

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sede Amministrativa: Università degli Studi di Padova
DIPARTIMENTO DI FISICA E ASTRONOMIA

SCUOLA DI DOTTORATO DI RICERCA IN
ASTRONOMIA
Ciclo XXIII

Tree-SPH code EvoL: profiling and optimization

Direttore della scuola: Ch.mo Prof. Giampaolo Piotto
Supervisore: Ch.mo Prof. Cesare Chiosi
Correlatore: Dott. Emiliano Merlin

Dottorando
Umberto Buonomo

Ai miei genitori

Abstract

This thesis is part of a larger project devoted to modeling galaxies via numerical hydrodynamical simulations, from their formation within their cosmological background throughout their evolutionary history.

The thesis has two primary objectives, strictly linked with one another. The first one, of astrophysical nature, is to develop a purely theoretical scenario for the formation and evolution of galaxies in the cosmological context. Given the complexity of the targeted physical systems, it is necessary to rely upon numerical simulations which in turn require an appropriate numerical code, able to follow the complex physics of galaxies. Therefore, the second objective is the optimization of such code, according to the paradigms of modern computational science. Indeed, given the complexity of the involved physical phenomena the code must be enriched with a number of complex algorithms, which at the same time slow down its performance to such a point that it is difficult to extensively use it to explore the whole scenario in detail, unless sophisticated optimization techniques are applied to drastically reduce the computational time.

In this thesis I have studied the problem of the optimization of numerical codes in serial and parallel architectures and I will present the different strategies I have adopted to make them faster, with particular attention to `EvOL`, the N-body Tree-SPH code entirely written in the house and widely used by the Padua group. To develop and optimize a code like `EvOL`, specific skills are needed, ranging from classical astrophysics to computer science. To cope with this, in the thesis I first introduce some general concepts on the functioning of a computer machine, its structure and architecture, the various reasons why a code, although correct from a mathematical and logical point a view, does not in practice work at the maximum level of efficiency and does not exploit the formal computing power to disposal in an optimal form. To this aim, I examine some important aspects of the general behaviour of a code and highlight several strategies that can be followed to optimize it. In particular, I analyze the code response in different situations highlighting the limitations, the routines with the heaviest load, and the importance of the communications between CPUs in parallel environment.

Given these premises, we present `EvOL`, a lagrangian code based on the classical NB-TSPH formalism, and describe in some detail its structure and

physical content. In particular we report on the new physical treatment of the interstellar medium and the companion codes ROBO and MaNN we have recently developed, and how they are implemented in EvOL. Then we describe the various optimizations we have applied to EvOL in order to improve its efficiency. Finally we report on the physical results we have obtained using EvOL, and ancillary codes ROBO and MaNN. In brief, we describe the physical content of the last release of EvOL, and the many hydrodynamical test we have performed to validate it. Second we describe the two studies dedicated to the treatment of the interstellar medium with the state of the art description of its chemistry and thermodynamical properties, and how these results can be implemented in EvOL, thanks to an extensive use of the artificial neural networks. Finally we present a set new models of early type galaxies obtained from cosmological initial conditions at varying the initial mass and density contrast with respect to the cosmological background and their subsequent evolution into well behaved galaxies. Particular attention is paid to the different role played by the density contrast of dark matter in the proto-halo generating the galaxy and the density of the baryonic matter (gas) required to initiate the star formation process. In general, while proto-galaxies of high mass give origin to a single dominant episode of star formation taking place very early on in a galaxy's history, proto-galaxies of low mass have a more complicated history of star formation, which a series of episodes of variable intensity often interrupted by periods of quiescence.

Abstract (Italian)

Questa tesi è parte di un più ampio progetto dedicato alla comprensione e alla modellizzazione della formazione ed evoluzione delle galassie da diversi punti di vista: formazione di strutture cosmologiche a tutte le scale e formazione ed evoluzione della singole entità fatte di materia oscura e barionica.

La tesi ha due obiettivi fondamentali intimamente legati. Il primo di natura astrofisica è già stato toccato e riguarda lo sviluppo di uno scenario puramente teorico per la formazione ed evoluzione delle galassie in contesto cosmologico.

Data la complessità dei sistemi fisici studiati, il lavoro è fortemente basato su simulazioni numeriche che richiedono appropriati codici in grado di seguire la complessa fisica delle galassie.

Pertanto, il secondo obiettivo riguarda l'ottimizzazione del codice numerico secondo i paradigmi della moderna scienza computazionale. Infatti, la complessità dei fenomeni fisici considerati, rende il codice più dettagliato da un punto di vista fisico, ma allo stesso tempo, lo rallenta al punto che non può essere direttamente utilizzato in modo sistematico, a meno che non vengano applicate sofisticate tecniche di ottimizzazione per ridurre drasticamente i tempi di calcolo.

In questa tesi ho studiato il problema della ottimizzazione di codici numerici in architetture seriali e parallele e le strategie adottate per renderli più veloci, con particolare attenzione al codice EVOL interamente scritto in casa e ampiamente utilizzato il gruppo di Padova.

Per sviluppare e ottimizzare un codice come EVOL sono necessarie competenze specifiche che vanno dalla astrofisica classica alla “computer science”. In sintesi, ho esaminato alcuni aspetti importanti del comportamento generale del codice e illustrato le diverse strategie applicate per ottimizzarlo. In particolare ho analizzato la risposta di un codice in situazioni diverse mettendo in evidenza i limiti globali, le routine di calcolo che spendono più tempo, e l'importanza delle comunicazioni MPI. Viene poi presentato il codice evolutivo e il suo contenuto fisico: EVOL è l'ultima versione del codice numerico lagrangiano sviluppato a Padova da C. Chiosi e collaboratori (Carraro et al. 1998; Merlin et al. 2010) basato sui classici algoritmi Tree-code (Barnes and Hut 1986) e SPH (Lucy 1977). Riporto il nuovo trat-

tamento fisico del mezzo interstellare attraverso i codici recentemente sviluppati ROBO e MANN, e l'implementazione del modello in EvOL. Ancora, vengono descritte le metodologie usate per valutare l'efficienza complessiva di EvOL, e le varie ottimizzazioni applicate; si descrivono anche i risultati ottenuti utilizzando il codice. Prima sono illustrati i diversi miglioramenti fisici che abbiamo incluso in EvOL, e i relativi tests idrodinamici fatti per validare il suo contenuto fisico. Successivamente si descrivono i due studi dedicati al trattamento del mezzo interstellare, della sua chimica, delle sue proprietà termodinamiche. Infine presentiamo una nuova serie di modelli di galassie “early type”, ottenute da condizioni iniziali cosmologiche variando la massa iniziale e il contrasto di densità rispetto al fondo cosmologico, e la loro successiva evoluzione. Particolare attenzione è rivolta al diverso ruolo svolto dal contrasto densità di materia oscura nel proto-alone e la densità della materia barionica (gas) necessaria per avviare il processo di formazione stellare. In generale, mentre le proto-galassie di grande massa danno origine ad un singolo episodio di formazione stellare che avviene precocemente nel corso della storia evolutiva della galassia, le proto-galassie di piccola massa hanno una storia di formazione stellare più complicata, con una serie di episodi di intensità variabile spesso interrotti da periodi di quiescenza.

Contents

1	Aims and plan of the thesis	21
1.1	Aims and plan	21
1.2	Introduction	23
2	Basic computer science and terminology	27
2.1	Why optimizing a code? How fast can we go?	28
2.2	A computer in a nutshell	28
2.2.1	Memory hierarchy	29
2.3	Parallel programming paradigms and machines	32
3	Fundamentals of optimization	39
3.1	Using the Cache	39
3.2	The pipeline	40
4	Serial and parallel optimization	47
4.1	The optimization cycle	47
4.2	Pre-processing and conditional compilation	48
4.2.1	Including files	49
4.2.2	The preprocessor C: #if - conditional inclusion	50
4.2.3	Macros	50
4.2.4	Conditionals	51
4.3	Make Utility	52
4.4	Compilation of a code	56
4.4.1	What a compiler cannot do	61
4.5	Algorithms	62
4.6	Scientific libraries	64
4.7	Evaluation of parallel performance	65
5	Debugging a code	71
6	Profiling	75
6.1	Time command	78
6.2	Profilers	79

7	Scientific Visualization	97
7.1	How does visualization adapt numbers to humans?	98
7.1.1	Visualization can reveal anomalies in data	99
7.2	AstroMD and VisIVO	100
7.3	SPLASH	103
8	N-body Tree-SPH code: EvoL	105
8.1	N-body technique	105
8.2	Description of the code: Gravity	106
8.2.1	Softening of the gravitational force	107
8.3	Description of the code: Hydrodynamics: The Baryon Physics	108
8.4	Smoothed Particles Hydrodynamics	109
8.4.1	Smoothing and softening lengths for SPH particles . .	112
8.4.2	Discontinuities	112
8.5	Time Integration	113
8.5.1	Leapfrog integrator	113
8.5.2	Time-stepping	113
8.6	Comoving coordinates	114
8.7	Periodic boundary conditions	115
8.8	Cooling and chemical composition	116
8.8.1	Radiative cooling	116
8.9	Cooling and chemical composition: ROBO	117
8.9.1	ROBO: the ISM model and companion code	117
8.9.2	The Artificial Neural Networks	119
8.9.3	Star formation	119
8.9.4	Energy Feedback	121
8.9.5	Energy feedback in EVOL	123
8.9.6	Chemical enrichment	125
8.10	Resolution	125
8.10.1	Open issues Jeans mass and resolution	125
8.10.2	Artificial pressurization	126
8.11	Parallelization	127
8.11.1	Data structures	128
8.12	Simulations of galaxy formation: setting the initial conditions	129
9	Profiling EvoL: actions and measurements	133
9.1	The initial conditions	133
9.2	Overview	134
9.3	Impact of profiling on the computational time	135
9.4	CORE vs. CPU	135
9.5	Weak and strong scalability	138
9.5.1	Planning the measurements	139
9.6	Computation time and communication	140
9.6.1	Serial fraction	144

9.6.2	Increasing problem size	150
9.7	Study of the MPI communications	155
9.7.1	Factors Affecting MPI Performance	155
9.7.2	Load imbalance	161
9.7.3	MPI communication time	164
9.7.4	Latency	169
9.8	Problems of cache misses	172
10	Optimizing EvoL	175
10.1	Compiler Flags	175
10.2	Preprocessor and Makefile technique	176
10.3	Periodism	179
10.4	soft_phi and soft_grav	184
10.5	Functions fper and pper	184
10.6	Back to the compiler	185
10.7	Total Optimization obtained in summary	186
11	Modelling galaxies with EvoL	191
11.1	Scientific Rationales of the selected papers	192
12	What to do next	195
13	General discussion and conclusions	197
A		211
A.1	Flags compiler	211
A.2	Machines and tools used	213
A.2.1	Machine	213
A.2.2	Tools	213
A.3	List of simulations	213

List of Figures

2.1	Cache Memory structure in modern computers.	31
2.2	Scheme of a shared memory architecture.	33
2.3	Scheme of a distributed memory architecture.	35
2.4	Scheme of a distributed shared memory architecture.	35
3.1	No pipeline vs 5 stage pipeline: The processing of an instruction by a processor: IF (Instruction Fetch): Read the instruction from memory, ID (Instruction Decode): Decoding and reading instruction operands from registers, EX (Execution): Executing, MEM (Memory): Activation of memory (only for certain instructions), WB (Write Back): Write the result in the register appropriate.	41
4.1	Maximum speed-up reached as a function of parallel fraction.	68
4.2	Combined effect of computing and communications on the speedup.	69
6.1	Xprofiler: entire call tree, including the calls to the libraries. .	83
6.2	Zoom-in view of main program	83
6.3	Example of a Trace Analyzer screenshot. Blue: time spent in calculations; red: time spent in communications.	85
6.4	Example of a Trace Analyzer screenshot with some Charts. Top: “event timeline” chart. Bottom left: “function profile”. Bottom middle: “message profile”. Bottom right: “collective operations profile”.	86
6.5	SCALASCA overview.	88
6.6	Example of visualization with Kcachegrind.	95
6.7	Part of the call to the Tree routine in EvoL displayed with Kcachegrind. Data have been collected with Callgrind. In the boxes, the percentage of time spent and the number of calls in the given branch are shown.	96

6.8	Kcachegrind screenshot displaying data collected with Cachegrind. On the right side are the routines with the larger number of cache-miss, divided by type. Bottom left: the line in the source code where the trouble is localized.	96
7.1	Kepler law: correctly display the data allows us to easily discover data relationships.	98
7.2	Graphically displaying data helps finding out troubles. A cosmological box from a wrong simulation, with a problem with the masses of particles. VisIVO help us to clearly see the two particles with wrong mass (red).	99
8.1	Data structures in EvoL. Void cells are included to allow for new particles to be created and/or imported from other CPUs. Ghost structures are reallocated at each time-step. . .	129
9.1	Same as in Tab. 9.2, in graphical form.	137
9.2	Scalasca profiling in CORE and CPU modes. As for the gprof data 9.3, the results show that the CPU mode does not change the computational time ripartition among the routines. . . .	138
9.3	Same data of 9.4 compared with the ideal speed-up.	139
9.4	Same data of 9.5, in graphical form.	141
9.5	SCALASCA call-tree for the “46” run on 128 processors. . . .	142
9.6	SCALASCA call-tree for the HDMM run on 128 processors. . .	143
9.7	Speed-up and ideal speed-up up to 128 processors with data from Scalasca. Data are the same as in Tab. 9.8, left column. . .	144
9.8	Efficiency up to 128 processors with data from Scalasca. Data are the same as in Tab. 9.8, right column.	145
9.9	Approximate crossing of the lines for the computation of the serial fraction with the simple graphic method described in the text. Data are from Trace Analyzer on MONSTER. . . .	146
9.10	Computational time in seconds, using as serial fraction the one obtained with the graphic method and the one obtained with the rigorous method. The differences are negligible for our purposes.	153
9.11	Time steps duration (in units of the Hubble time) during a typical run.	154
9.12	Percentage of MPI communication for three simulation at varying of the number of adopted CPUs.	157
9.13	Different weight in MPI communications in the same simulation on different machines at varying the number of adopted CPUs. The difference is in the performances of the high performance net subsystem on SP6.	158

9.14	Total number of bytes exchanged between processors at varying the number of CPUs for three simulations.	159
9.15	Ripartition of time spent MPI in communications: total (blue), synchronizations (orange), total communications divided into point to point (green) and collective (purple), the time for initialize the MPI execution environment (cyan) and the temporal cost of the calls to the MPI routines (brown and negligible). The dominant components are the P2P communications and the synchronizations.	160
9.16	Ratio between P2P and collective communications. the Y axis is logarithmic. Increasing the number of processors, in the small simulations (cyan) the collective communications are not negligible, and using a large number of CPUs (256 or more) this contribution can become important.	162
9.17	Variation of the fractional weight of the various components of MPI communications with the number of CPUs. Orange: total communications; blue: synchronizations.	163
9.18	Variation of the fractional weight of the various components of MPI communications with the number of particles. Orange: P2P; Yellow: collective; blue: synchronizations.	164
9.19	Load imbalance as fraction of the total computational time at varying the number of used particles on a constant number of CPUs (32).	165
9.20	Load imbalance as fraction of the total computational time at varying the number of used CPUs. The simulation is the 18^3	166
9.21	Ratio computation/communications as displayed with Trace Analyzer, for the 18^3 simulation on MONSTER with 16 CPUs. 166	
9.22	Load imbalance in the computation. Same simulation as in 9.21 167	
9.23	Load imbalance in the communications. Same simulation as in 9.21	167
9.24	As in 9.23 but with communications divided by type. Total times spent in Barriers are shown.	168
9.25	MONSTER data from Tab. 9.14. When the dimensions of the transmitted data are too small the total time of communication is essentially due to the latency.	170
9.26	Trace Analyzer can easily highlight the differences in the communication times among processors. Here the structure of the MONSTER cluster is made evident (8 processors per node). The communication times between processors belonging to the same node are much lower (blue) then those relative to processors belonging to different nodes.	171
9.27	Cachegrind on the 18^3 particles simulation on BUD (on a single processor). The routine periodism is highlighted.	173

10.1	gprof on a 18^3 particles, single CPU simulation on BUD. Routines are ordered per fraction of total inclusive time spent (i.e., time spent to call the routine plus time spent in all the depending subroutines).	180
10.2	gprof on a 18^3 particles, single CPU simulation on Bud. Routines are ordered per fraction of exclusive (SELF) time spent.	180
10.3	All the routines which call the routine periodism.	181
10.4	Final improvements of the routine periodism with respect to the preprocessed version.	183
10.5	Total optimization obtained.	187
10.6	Same as in Fig. 10.5 excluding the first column	188
10.7	Cache use after optimization	189

List of Tables

2.1	Latencies for different memory levels.	30
2.2	Cache miss for Power 3.	31
2.3	Myrinet vs. Gigabit Ethernet. Peak and Sustained in GFLOPS	36
2.4	Comparison of Latency vs. Bandwidth pros and cons.	36
2.5	Latency vs. Bandwidth for different hardware networks.	37
3.1	Average cycles per instruction, for the IBM-Power3 processor.	44
4.1	Evolution of the complexity of different algorithms for solving 3D elliptic equations.	62
4.2	Cycles spent on average for different functions, measured on an old system (HP EV6 to 500 Mhz.)	64
4.3	Average time spent on different functions normalized to the addition, measured on BUD	65
4.4	Average time spent on different functions normalized to the addition, measured on SP6	65
6.1	Example of time ripartition of the computational time be- tween different procedures.	77
6.2	Timers.	78
6.3	Example of a prof output.	80
8.1	<i>Cosmological parameters adopted in the simulations.</i>	130
9.1	Impact on the computational times of the different tools un- der examination. The times are normalized to the run without profiling.	135
9.2	Computational times in CORE and CPU modes. In the CORE mode the number of used processors is equal to the number of tasks, while in the CPU mode it is half this number. The num- ber of time-steps per hour is only slightly higher in the CORE mode, even with a doubled hardware (data from HDMM sim- ulation)	136

9.3	Profiling with gprof in CORE and CPU modes, using the same number of tasks. The profilings are very similar, so the Cpu modality does not substantially change the computational time ripartition between the routines in EvOL.	136
9.4	Scalability: number of particles per second and speed-up, in the “46” strong scalability test.	138
9.5	Particles per second on a single processor, progressively increasing the dimension of the problem (weak scalability). In the right column, data are normalized to the largest number.	140
9.6	Computation and communication times of simulations in physical (HDMM) and comoving (“46”) coordinates. Two cases with different numbers of processors are shown.	140
9.7	Fractional ripartition between computation and communication for the 2×18^3 particles simulation on MONSTER (data are obtained with Trace Analyzer).	141
9.8	Speedup and efficiency obtained with Scalasca for the “46” run on 128 processors.	143
9.9	The increasing function $k(n)$ indicates the growth of the computational time in the parallel mode as the number of particles increases. Data are from Scalasca on 32 CPUs.	152
9.10	α and β parameters. See text for details.	152
9.11	Percentage of MPI communication for three simulation at varying of the number of adopted CPUs.	156
9.12	Total number of trasmitted bytes among the processors for three simulations(with 2×18^3 , 2×46^3 particles and the HDMM simulation) divided into P2P and collective communications.	161
9.13	Time spent in communications, number of communications, and estimation of the average duration of a single communication.	169
9.14	Comparison of the MPI transmission times (in μsec) at varying of the number of bytes transmitted, on SP6 and on MONSTER.	169
9.15	Comparison among the average times for a single communication and the latency, for MONSTER and SP6.	171
10.1	Improvements of EvOL with respect to the originary code.	179
10.2	Improvement of periodism with respect to preprocessor version	182
10.3	Further improvements of the routine periodism with respect to the previous version.	184
10.4	Final improvements of the routine periodism with respect to the preprocessed version.	184
10.5	Improvements of the soft_phi and soft_grav routines with respect to the Period IV version.	185

10.6	Improvements of the routines fper and pper with respect to the SoftPG version.	185
10.7	After the final optimization procedure has been added by the compiler.	186
10.8	Percentage of reduction of cache-miss after optimization . . .	187
A.1	Simulations on MONSTER with the old code.	214
A.2	Simulations on MONSTER with the new code.	214
A.3	BUD/HAL9002: only the most relevant ones; 18^3 particles on 1 processor.	214
A.4	Simulations at CINECA ISCRA	215
A.5	Simulations at CINECA CNE0IN05	216

Chapter 1

Aims and plan of the thesis

1.1 Aims and plan

This thesis is part of a large project devoted to understanding and modeling the formation and evolution of galaxies from different points of view: cosmological formation of structures at all scales, dynamical formation of single entities made of dark and baryonic matter and their subsequent evolution from the initial epochs to the present, i.e. the galaxies of different morphological type, the thermodynamics of the interstellar medium inside galaxies, the chemical enrichment of the gas and stellar content, and finally the spectral and photometric evolution of the stellar populations composing a galaxy.

The thesis has two primary goals intimately related to each other. The first one of astrophysical nature has been already touched upon and is the elaboration of a mere theoretical scenario for galaxy formation and evolution in cosmological context. Given the complexity of the target physical systems, it heavily stands on numerical simulations requiring adequate numerical codes able to describe and follow the complex physics of galaxies. Therefore, the second target is of mathematical and computational nature, i.e. the optimization of the code according to the paradigms of modern computational science. Indeed the complexity of the physical phenomena involved make the codes in use more and more articulate and hence physically detailed but at the same time they slow down it to a point that it cannot be straightforwardly systematically used to explore the parameter space of the theoretical models unless sophisticated techniques of optimization are applied to drastically reduce the computational time.

In this thesis I have studied the problem of the numerical optimization of codes in parallel architectures and the adopted strategies with particular attention to the code EvoL entirely written in the house and widely used by the Padova group. Developing and optimizing a code like EvoL requires specific expertises that go from classical astrophysics to computer science and

therefore a large effort has been made to acquire familiarity and expertise in this area of research. At the same time the primary goal of the project has been targeted participating to the development of theoretical models of galaxy formation and evolution.

The plan of the thesis is as follows. Chapters 2 through 7 are meant to provide the reader with a short review of some key points in Computer Science and are essentially devoted to technical aspects of computers, programming, software developing and profiling tools. The second part of the thesis (Chapters 8 through 13) are related to the scientific side of the whole project, with particular attention paid to the optimization of the performance of the code used in our studies, and provides a summary of the scientific results obtained so far.

Chapter 2 introduces the basic concepts and information required to attack the problem from the point of view of computer science, presenting in some detail the many reasons why a numerical code cannot perform at the desired level of efficiency and how to cure it. The whole problem can be reduced to the statement that numerical codes must be suitably designed and optimized to benefit of the unprecedented computing power offered by modern computers, so there is ample room for optimization strategies. In Chapter 3 we present the fundamental concepts of optimization and in Chapter 4 we discuss them in the context of serial and parallel architectures. In Chapter 5 we describe the so-called debugging strategy and in Chapter 6 we define and discuss the associated tools to get the performance profile of a code in a given environment. Chapter 7 deals with the widely adopted technique of visualizing the results of a numerical code to pin down possible inconsistencies in addition to understanding the physical problem underneath. Chapter 8 describes the NB-TSPH code *EvOL* and its physical content. In particular we report on the new physical treatment of the interstellar medium by means of the codes *ROBO* and *MaNN* we have recently developed and it is implemented in the *EvOL*. Chapter 9 describes the performance profile of *EvOL*. In brief, we examine some important aspects of the general behaviour of the code and several strategies we have followed to optimize it. In particular we analyze the code response in different situations highlighting the limitations, the routines with the heaviest load, and the importance of the MPI communications. In Chapter 10 we describe the various optimizations we have applied and we evaluate the overall efficiency of *EvOL*. In Chapter 11 we describe the results we have obtained using *EvOL*. First we report on the many physical improvement we have included, and the companion test sets. Second, we report two studies dedicated to the treatment of the interstellar medium with the state of the art description of its chemistry and thermodynamical properties, and how these results can be implemented in *EvOL* thanks to an extensive use of the artificial neural networks. Finally we present a set new models of early type galaxies obtained from cosmological initial conditions at varying the initial mass and density contrast

with respect to the cosmological background and their subsequent evolution into well behaved galaxies. Particular attention is paid to the different role played by the density contrast of dark matter in the proto-halo generating the galaxy and the density of the baryonic matter (gas) required to initiate the star formation process. In general while proto-galaxies of high mass give origin to a single dominant episode of star formation taking place very early on in a galaxy's history nearly independently of their initial density, proto-galaxies of low mass have a more complicated history of star formation, which indeed is gradually turned into a series of episodes of variable intensity often interrupted by periods of quiescence, and on the average delayed in time, at decreasing mass and/or initial density of the galaxy. In Chapter 12 we shortly mention the future steps to be undertaken to improve both the physics and the numerical performances of *EvoL*. Finally, in Chapter 13 we draw some general conclusions. The thesis is also accompanied by a number of appendices of technical nature reporting some information and/or details of the many numerical tests we have made.

1.2 Introduction

The importance of numerical simulations in modern science is constantly growing, because of the complexity, the multi-scaling properties, and the non-linearity of many physical phenomena. Numerical simulations of three-dimensional self-gravitating fluids have become an indispensable tool of modern astrophysics. They are widely used in the study of nonlinear gravity of dark matter, the formation of clusters of galaxies, the interaction between isolated galaxies, the evolution of the intergalactic gas. When analytical predictions are not possible, we are forced to compute the evolution of physical systems numerically. Without numerical techniques, the immense progress made in this field would have been impossible, since the analytical treatment is often restricted to ideal problems. In modern astrophysics, the numerical simulation of systems has become a widely used tool on nearly all length scales, ranging from problems on cosmological distances, to galaxy formation and evolution to star and planet formation. The numerical simulation of a self-gravitating and/or gas dynamical system is a basis common to all those problems, no matter what length scale they belong to. The Universe, our Galaxy, but also a small group of planets, or even a single star, whose complexity is beyond the capabilities of analytical models, show such a deep complexity that analytical investigation can only help explaining general trends, often returning little more than approximated qualitative results. At present, numerical simulations are the best tool we have to investigate such a complex phenomena. That is why more and more accurate and detailed simulations are still needed.

Cosmological and galaxy-sized simulations of structure formation have

helped in casting light on a number of issues concerning how cosmic systems formed and evolved along the history of the Universe. Conversely, they raised an equivalent number of new questions, especially at the light of many recent observations that often appear at odds with well established theories. Numerical simulations have contributed much to our current understanding of the physical processes which lead to the universe we observe.

To simulate the dynamical evolution of a limited region of the Universe, it is necessary to properly model the main different components it is made of, namely Dark Matter and Baryonic Matter (gas and stars), representing them with different species of particles. Moreover, a physically robust model of the fundamental interactions is required, at the various scales of interest. Finally, a suitable cosmological framework is necessary, together with a coherent setting of the boundary conditions and with an efficient algorithm to follow the evolution of the simulated system through time using small but finite *time-steps* to approximate the equations of motion to a finite-differences problem.

The simulation techniques for such systems can be grouped into two different schemes: (i) The grid based methods (Eulerian View) which divide the space into finite sized cells and compute the physical quantities such as temperature, pressure, etc., inside those cells (see e.g. Stone and Norman (1992); Ryu et al. (1993); O'Shea et al. (2004), and references therein). (ii) The particle based methods (Lagrangian View) that describe a physical system (made of Dark and Baryonic Matter) by means of a large sample of "particles" to which physical quantities such as mass, position and velocity are assigned or computed (see e.g. Hernquist and Katz (1989); Dave et al. (1997); Springel et al. (2001); Wadsley et al. (2004); Springel (2005), and references therein). In the lagrangian approach to the problem, no reference grid is superposed to the computational volume, while particles are forced to move under mutual (or external) interactions. Which approach is best for modeling a particular system depends on the problem to be modeled.

For a system evolving only under the influence of gravity, the particle based approach leads to the classical N-body problem. A set of N particles evolves according to the force on each particle exerted by all the others. A particle-based simulation naturally concentrates the computational work in the most interesting areas of the flow where many particles are used. However the treatment of regions with very low densities of matter (much fewer particles in turn) can be a problem if such low density regions must be studied in detail (see e.g. O'Shea et al. (2005), for a comparison of grid and particle based codes for cosmological simulations). The efficient computation of the gravitational forces is a longstanding numerical problem.

To model gas dynamics, the Smoothed Particle Hydrodynamics technique has been developed over the years (SPH, Lucy (1977); Gingold and Monaghan (1977); Benz (1990); Monaghan (1992)). The method has achieved great success by incorporating the gas dynamics in the framework of the par-

ticle method. In brief, SPH describes the gas by means of particles (which makes the method so useful in combination with N-body methods), which are assumed to sample the local hydrodynamic quantities of the underlying flow. Each particle can possess, in addition to mass, position and velocity, also a list of physical features such as internal energy, density, temperature, chemical composition, etc. The volume (or surface) density is reconstructed for each particle on the basis of the positions of nearby “neighboring” particles. Thus, these gas particles can feel not only the gravitational forces that all particles in the simulation do, but also pressure forces and other gas dynamical effects.

Modeling dynamical and hydrodynamical systems using particles relies on the sufficiently accurate computation of both the gravitational and hydrodynamical forces of the particles on each other, then advancing them forward in time according to those forces. Thus a time integration of the particles-equations of motion and additional equations for hydrodynamic quantities such as internal energy, is the problem to be solved. Constraints on the numerical code designed to this aim are the fair reproduction of the evolution of well understood real systems that the simulation is supposed to model, and the efficiency and minimum cost in terms of computational time.

In this thesis we present *EvOL*, the Padova Tree-SPH code, designed to respond to many of these requirements. The core of the PD-TSPH code was written during the '90s by C. Chiosi, G. Carraro, C. Lia and C. Dalla Vecchia (Lia et al. 2002). In its original release, the PD-TSPH was a basic Tree + SPH code, written in Fortran90, conceptually similar to *TREESPH* by Hernquist and Katz (1989). Schematically, PD-TSPH used an early formulation of SPH (Benz 1990) to solve the equations of motion for the gas component, and the Barnes and Hut (1986) Tree algorithm to compute the gravitational interactions. In the following, we describe in some detail the version of *EvOL* (Merlin et al. 2010), the ultimate release code currently in use: in brief, the Tree-SPH (i.e. Oct-Tree plus Smoothing Particle Hydrodynamics) formalism, its parallel architecture, the physical content in addition to gravity, e.g. radiative cooling functions, chemical evolution, star formation, energy feedback), the methods and the tools used to debug and to measure its *PROFILE* in the serial and in parallel performance. The aim is to identify the most computationally expensive parts of the code.

EvOL is written in Fortran 95 and is designed to be versatile, flexible and extensible, with modular options that can be selected either at the stage of compilation or in the course of a run by means of suitable text input files.

The code includes a number of pre-built sub-makefiles from common commercial and open source compilers, which are selected automatically. Particular care is paid to “double precision” (8 bytes on most systems), for which about 15 digits of precision are retained at the cost of doubling the required memory relative to single precision. While even this level of preci-

sion does not eliminate finite precision errors, they will at least be of much smaller amplitude. In any case, when dealing with high resolution simulations, single precision should be avoided, even if this latter would provide a factor of two reduction in computational time. For example, when particles are close to each other in space, errors in the position can grow simply because the particles share all but the last few digits of their coordinate values. In such cases, and during long lasting runs, catastrophic cancellation can grow to levels that become important for the correct evaluation of the problem under investigation

EvOL can run on single processors or in parallel environment, using the OpenMPI paradigm on large scale, distributed memory parallel machines.

Chapter 2

Basic computer science and terminology

Super-computing is a key technology of modern science and engineering, indispensable to solve critical problems of high complexity. As a prerequisite for the productive use of today large-scale computing systems, powerful and robust performance analysis tools are needed in order to effectively optimize algorithms and parallel applications.

In order to maximize the scientific productivity, a numerical code must be able to perform simulations with the lowest possible computational expenses while still maintaining an accurate description of the evolution.

In this thesis, we focus the attention on the techniques which can make calculations more efficient. Because efficient calculations require both efficient algorithms and efficient implementation of these in the computational hardware, we describe in some detail the minimum optimization that must be implemented to improve the performance of the code itself on modern, microprocessor based computers.

We perform an extensive series of performance tests and analyze the effects of the code optimizations.

We also discuss in detail the optimizations techniques that we should use to maximize the performance of numerical codes, EvOL in particular.

How to make a program run faster? How to get answers to our problems on a reasonable time? The purpose is to describe both the theory and practical support for the optimization of scientific/numerical codes. We start introducing the concept of “high performance computing”, and examining a number of examples from the point of view of the theoretical formulation and practical realization, showing the problems and possible solutions. Optimization of a code requires full acquaintance not only with the physical nature and the mathematical description of the problem to study but also of the hardware and software of the computer and its operating system in use.

All the examples will refer to codes written in Fortran for Unix/Linux system, because they are the most widely used language and environments for scientific numerical codes.

2.1 Why optimizing a code? How fast can we go?

The performance of any code critically depends on how the code is written. Different ways to plan and write the code, and different algorithms or implementations of the same algorithm, can produce the desired result with very different time and computing costs. Optimization consists in removing by as much as possible all causes slowing down the execution of a program.

There are various ways to speed up the execution time of a program. The most obvious one is to buy a faster computer. The alternative is to better write (organize) the whole code or part of it. This option is often extremely slow and expensive, but once accomplished is very rewarding in terms of performances. Often it is just sufficient to change only part of the code or to improve the underlying mathematical algorithm: this is the typical approach in High Performance Computing (HPC).

The first thing to check is the effective utilization of available resources: HPC requires more than just writing a code and making it running on a machine if this is left largely under-used.

Once a code has been tested, the goal is facilitated by using one of the available tools spotting the weak points of the code. The task is trivial if the code is small, but it can be difficult if the code is long and/or complex (and not sufficiently documented). Only after this preliminary step, the performance of a code can be assessed and if required changes of the code can be attempted to reduce the computational time. The task however cannot be undertaken without knowing in some detail the memory architecture and the processor of the computer in use.

2.2 A computer in a nutshell

In its basic components, a typical computing machine can be reduced to:

- A CPU (central processor unit) that performs the operations. It is composed by several different and distinct logical units that perform floating point operations (flops), integer operations, and operations of data loading/storing. The CPU is characterized by the “clock cycle” which is the minimum time interval required to make a generic elementary operation. Each operation performed by the CPU necessarily requires an integer multiple of clock cycles.
- A System Memory: it manages and stores the data on different media, and provides data to the CPU to perform the requested operations.

It is organized in a hierarchical structure, going from the hard disk to the RAM memory and to intermediate memory units, called caches, characterized by different size and speed of access. A RAM memory is characterized by the latency parameter, i.e. time required to retrieve a data.

- An Operating Systems: the most important program that runs on a computer. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers. An operating system (OS) is a set of programs that manage computer hardware resources, provides common services for application software, and acts as an intermediary between application programs and computer hardware.

2.2.1 Memory hierarchy

The most important limit set by the hardware is the one imposed by the memory latency (Hennessy and Patterson 2007). Processors are becoming faster and faster, according to the so-called Moore's Law.¹ The RAM, however, does not increase in proportion to their performance. The universal solution has been to incorporate relatively small caches of memory into the processor itself, from which data can be accessed very quickly. The memory hierarchy in modern processors is composed of different levels of memory, each characterized by the access speed which is inversely proportional to the size: the bigger the size, the longer is the time required to retrieve the data it contains.

So there are two important issues to point out: first, accessing cached data is very fast compared to accessing the main memory; second, the cache memories are typically much smaller than the total memory required by a typical numerical problem. Normally, cache memory includes two or more levels, labeled "L1", "L2" etc, which are divided into a number of "lines" and "sets" The L1 cache is the smallest and fastest memory; higher cache levels are slower and larger. A cache line consists of ~ 32 -128 bytes of memory; this is the smallest increment of information that can be loaded into or retrieved from that cache at a time.

Even if a program requires only a single value from a given range of memory, say a single integer (4 bytes) or a single double precision (8 bytes) real value, that load step also brings several additional, possibly unneeded values into the cache. Any given address in main memory can be loaded into any one of exactly ns lines, where ns is the number of sets (the set "asso-

¹Moore's law is an empiric relation describing a long-term trend in the history of computing hardware: the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every 18 months.

Table 2.1: Latencies for different memory levels.

Level	Dimensions	Access Time
Registers	32	≤ 1 cycle
Cache L1	64 + 32 (KB)	1-10 cycles
Cache L2	4 MB	10-100 cycles
RAM	> 1 GB	> 100 cycles
Disc	> 10 GB	> 10000 cycles

ciativity”). Typically, $ns \sim 2-4$. This means that less immediately needed quantities may be overwritten by loads of other quantities if other calculations require them. Information may need to be loaded into cache from main memory many times if it is required for many calculations. We are therefore well advised to reuse previously cached data to the fullest extent possible before discarding it for other more immediately useful data.

The scheme below schematically shows an example of memory hierarchy, consisting of the disk, where data is stored on magnetic media, RAM memory, and 1 to 3 levels of cache memory.

The last level of the hierarchy is composed of registers, memory cells that are contained within the CPU. Each calculation, whether integer or in floating point, is done by reading the values from the registers, supporting any intermediate values still on the registers and writing, always on the registers, the final result. Then the final value will be copied back into memory. For example, a typical machine with 2 GB of RAM rests on a 4 MB second level cache (L2) which insists on a first-level cache (L1, on-chip) of only 64KB for data and 32KB for instructions, the latter relying on a small number of registers (typically 32 or 64). Each data-read operation is done moving from RAM to the registers, each time passing for the various cache levels before getting to the registers themselves. Similarly, each data-write operation must follow the opposite path. Clearly, the access times reflect the same disproportion in the size of various levels of memory. The latencies and the typical size of the memory levels expressed in clock cycles are shown in the following table.

To conclude, an inefficient use of the memory hierarchy forces the CPU to rest and wait for the data to retrieve, causing a significant loss of performance.

The use and benefit of a cache rests on the principle of locality, i.e. reusing data and instructions that were recently used. To be more precise the cache memory is beneficial when the spatial and temporal locality of data are at work, that is

- Temporal locality: once it is used as a generic A, this will be reused several times over a short period of time;

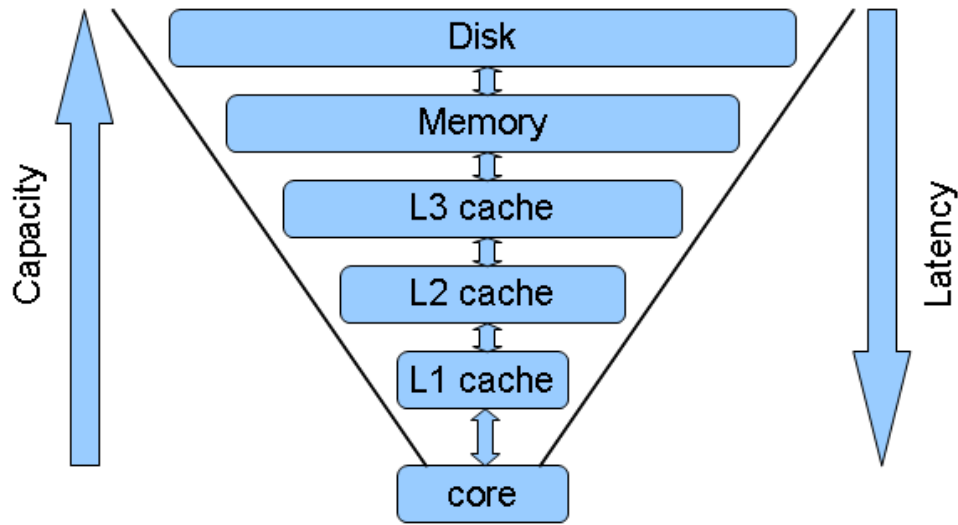


Figure 2.1: Cache Memory structure in modern computers.

Table 2.2: Cache miss for Power 3.

Livello	cache miss
da cache L1 a registri	1 ciclo
da cache L2 a cache L1	7 cicli
da Ram a L1 o L2	36 cicli

- Spatial locality: used a generic A another generic B, stored in the memory location next to A, is also used within a short time.

In addition to this, due to spatial locality, every time a data is loaded from memory to the cache, also neighbouring data in the so-called “cache line”, composed of 64-128 bytes, are loaded. When requesting a particular data, not only the requested data is moved, but also the entire line that contains the data. If an efficient use of the spatial and temporal locality is practiced the latency seen by an actual process will be that between the cache and the CPU which is much shorter than that between the CPU and RAM.

The parameter measuring the efficiency of utilization of the cache is the “cache miss”, or the percentage of requested data not found in the cache. In this case, the CPU is stalled until the missing information is not found and loaded from the cache level or from RAM. To get a quantitative indication of the effect on the performance of cache misses, using as reference the IBM Power3 processor latencies, some data are presented in Table. 2.2.

For example, consider a code that for every 100 data requests in L1

produces 5 cache misses. Under ideal conditions, the time taken will be $(95 \times 1) + (5 \times 7) = 130$ clock cycles, assuming that the missing data are all contained in the L2 cache, which has latency of 7 clock cycles.

An even more critical situation would happen if the requested data should be retrieved from the L2 memory, and the same percentage of 5 % are cache misses and are written on the RAM memory. In this case, the total time would be equal to $95 \times 36 \times 1 + 5 = 275$ clock cycles, more than doubling the case of the same percentage of cache misses of the previous case.

In the dramatic case that all data is contained only in RAM, then the total time would be equal to $100 \times 36 = 3600$ clock cycles, i.e. 3600 % slower than the ideal case.

On a more general level, we will frequently require that the same values are accessed repeatedly. For example, in SPH calculations spatially adjacent particles will have nearly identical lists of neighboring particles and nodes of interaction for gravity calculations. All calculations involving those lists will load the same quantities for these particles many times.

Note that all microprocessors share the constraint that loading or storing a value to or from memory is much more expensive than, for example, adding them together. Depending on the processor, a single calculation may take one or at most a few clock cycles, and several calculations may be processed at the same time on the same processor. On the other hand, loading and storing may take as many as hundreds or thousands of clock cycles.

Although the particle/node data corresponding to these entries may total only a few tens or hundreds of kilobytes (quite a small amount given the memory sizes of modern computers), it will certainly be much larger than any amount that can be allocated in the fastest level of the computer's cache hierarchy.

As the summation for each particle proceeds, data for each particle/node must be retrieved either from a page of main memory or from secondary/tertiary caches, stored in primary cache and operated upon by the processor, only to be evicted by later data in the interaction list and retrieved again for the next particle, an effect known to computer scientists as “cache or TLB thrashing” (where TLB is for Translation Look-aside Buffer, see Sect. 3.2).

As (Warren and Salmon 1995) point out, if we could arrange that data remain in cache after their first load from the main memory, the calculations would proceed much faster.

2.3 Parallel programming paradigms and machines

Even with the fastest processors available, the time required to perform detailed astrophysical simulations on a single machine would exceed any

reasonable time for a response to come. Fortunately, very often the task can be shared among many processors thus drastically reducing the computing time. Parallel computing architectures are commercially available at affordable costs.

To this aim, the body of calculations has to be divisible into independent sub-units so that they can be run in parallel on different processors. First, the data to be supplied to each sub-unit must not change before the task is completed and all relevant results from previous tasks must be communicated to each processor before a new task is started. Furthermore, the subunits should be of comparable size, so that they can be evenly distributed among all the processors and so that little time is wasted while some processors sit idle and others complete disproportionate shares. Secondly, communication must be fast, so that relatively little time is spent synchronizing the results of different processors. Finally, the amount of work that cannot be broken into independent parts (the so-called “serial fraction”) should be made as small as possible.

For our purposes, parallel computers can be roughly classified according to memory architecture and processor-processor communication. Different memory architectures imply different programming models (Dongarra 2003).

Shared Memory-OpenMP. Each processor can access all memory as global address space, processors operate independently, sharing the same memory resources and a change to a location in memory due to a processor is visible to all others,

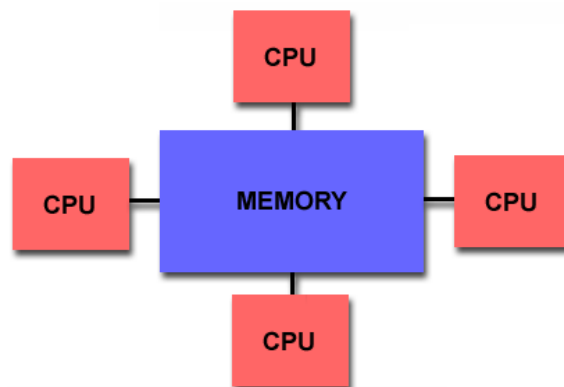


Figure 2.2: Scheme of a shared memory architecture.

For shared memory systems, partitioning naturally occurs at a very low level in the code, often at the level of individual loops themselves. An important factor, limiting the performance in this case, is that the work per loop iteration can be extremely small, so that the overhead required to distribute work among processors becomes significant.

Multi-Threaded Model. A single process can create some threads (execution units), that the operating system will schedule (perhaps on different CPUs). Each thread shares all resources associated with the process that generated it (including its memory space), has its own private memory area, and communicates with other threads through the global memory space. Therefore, synchronization operations are required to prevent more than one thread tries to update a global memory location at the same time.

The model is usually associated with multi-threaded shared memory architectures. Possible implementations are a library of primitives for programming multi-specific threads or a set of compiler directives to be included in the serial code:

- POSIX Threads (Pthreads) is a library used in C code for management of parallelism. It is completely explicitly (by the programmer);
- OpenMP is based on compiler directives. It can be very simple to use, because one can make an "incremental parallelization" from the serial code. OpenMP (see e.g. OpenMP Tutorials) uses the fork-join model of parallel execution. All OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region of a code is encountered.

FORK: the master thread then creates a team of parallel threads. The statements in the program that are enclosed by the parallel region are then executed in parallel among the various team threads;

JOIN: When the team threads complete the statements in the parallel region, they synchronize and terminate, leaving only the master thread.

Architecture Distributed Memory-MPI. Each processor has an area of local memory; processors can exchange data only through the communication network (see e.g. Open MPI Documentation; Gropp et al. 1994). There is no concept of a global address space among the processors: the programmer must explicitly define when and how to communicate the data and to synchronize tasks residing on different nodes. The adopted communication infrastructure is very important.

Message Passing Model. A group of processes use their own local memory during the run and can reside on the same machine or on several different machines connected via a communication network. Data exchange is co-operation between processes: inter-process communication occurs via the explicit exchange of messages. The implementation of message passing programming model requires a library of functions to be called in the code. It may happen that a set of tasks work collectively on the same set of data, each task works on a different portion of the same data structure or each task performs the same operation on its own data partition. On

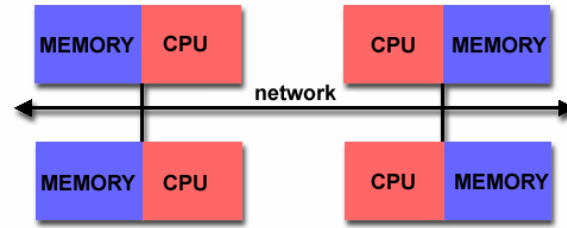


Figure 2.3: Scheme of a distributed memory architecture.

shared memory architectures tasks access the data structure through shared memory, while on distributed memory architectures the data structure is divided into chunks and each of them resides in local memory of the task.

Hybrid architectures: Distributed Shared Memory. The most common clusters are composed of shared memory nodes connected by a network. The single node is a shared memory system. The nodes are interconnected via a communication network.

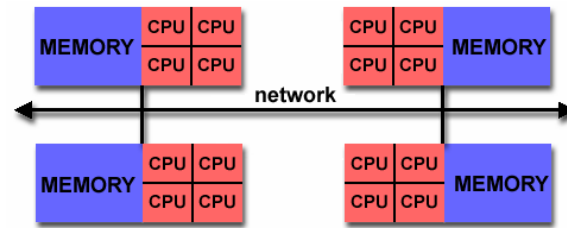


Figure 2.4: Scheme of a distributed shared memory architecture.

Hybrid Parallel Programming Model. They can be combined together two or more parallel programming models for example: it is possible to combine in hybrid architectures, the shared memory model (OpenMP) with a message passing model (MPI):

Within the node the communication between the processes on the same node is realized via OpenMP, while the communication between processor residing on different nodes is realized via MPI protocol.

Importance of the interconnection network. Unlike in serial machines, for cluster computers the interconnection network is very important, as the following tabel clearly illustrates.

Why this difference? We note that the performance of systems composed by identical processors can be very different because of the different performance of the network connecting the nodes of the machine (Myrinet Vs. Gigabit Ethernet).

A network is defined by:

Table 2.3: Myrinet vs. Gigabit Ethernet. Peak and Sustained in GFLOPS

Nome	Vendor	Cpu	N.Cpu	Peak	Sustained
K.I.S.T.	IBM	Intel Xeon@2.4 Ghz	1024	4915	3067
TotalFinaElf	IBM	Intel Xeon@2.4 Ghz	1024	4915	1755

- Topology: The structure of the interconnection network:
 - ring
 - hypercube
 - tree
- Latency (L): time needed to “send” an empty message (time of start-up)
- Bandwidth (B): data transfer rate (MB/sec.)

Latency vs. Bandwidth. The latency is the time needed to exchange a message of minimum size (0 bytes) between two processes. It is the time to ‘activate’ the communication between two processes; it has the typical value of 1-5 μ s. The bandwidth is the amount of data that can be communicated per unit of time. A typical bandwidth is 1-2GB/s (1-2 KB/ms). So the communication time (T) for a message of N MB is $T = L + N/B$. Sending many small messages may result in communication overhead (due to latency). Furthermore, the synchronization also induces an overhead because before proceeding further with the run, it is necessary to wait for the slower task to reach the synchronization instruction. Many communication operations require explicitly or implicitly synchronization among processes. It is therefore necessary to minimize the synchronization operations with an optimal load balancing.

Table 2.4: Comparison of Latency vs. Bandwidth pros and cons.

narrow bandwidth		large bandwidth
high latency	worst case	few large communications
short latency	many small communications	best case

Table 2.5: Latency vs. Bandwidth for different hardware networks.

Name	Latency (μs)	Bandwidth (MB/s)	Cost
Fast-Ethernet	100	12	<10
Gigabit-Ethernet	100	12 5	>10
Myrinet	7	250	>100
QSWnet	5	360	>100
InfiniBand	4	750	>1000

Chapter 3

Fundamentals of optimization

3.1 Using the Cache

Data allocation in Fortran/C. Row-major order and column-major order describe methods for storing multidimensional arrays in linear memory. Following the standard notation for matrices, rows and columns are indicated by the first and second index of a two dimensional array, respectively. Array layout is critical for performance because accessing array elements that are contiguous in memory is usually much faster than accessing elements which are not contiguous. In row-major order the rightmost indices vary faster as one steps through consecutive memory locations, while in column-major order the leftmost indices vary faster.

Fortran and C do storing of array element in a different logical order, so the same sequence of elements is stored differently in memory. C makes use of the row-major order, Fortran goes by column-major order. When an array of data is accessed, it is important to know in what order the language we are using sort the arrays in physical memory (for topics of these technical chapters, see e.g. IBM Optimization and Tuning Guide 1996; IBM Optimization and Programming Guide 2010; Intel Optimization Reference Manual 2011).

Cache thrashing. Thrashing (see Sect. 2.2.1) can occur, in general, when dimensions of vectors and matrices are larger than the first level cache, and these quantities must be mapped in more levels of cache or RAM. In practice, the presence of matrices and/or “commons” may cause thrashing because contiguous allocation of variables is forced. Thrashing is avoided by placing unused memory locations (array padding) in order to prevent mapping of quantities on the same cache line. Modern compilers, with high levels of optimization and with obvious limits, can partially cope with this problem.

Cache blocking. Cache blocking prevents extensive use of the cache, especially as far as temporal locality is concerned. Cache blocking requires structuring data blocks so that they conveniently fit into a portion of the L1 or L2 cache. By controlling data cache locality, an application can minimize performance delays due to memory bus access. The application controls the behavior by dividing a large array into smaller blocks of memory so a thread can make repeated accesses to the data while the data is still in the cache. Compilers often use the same technique, by grouping related blocks of instructions close together so they execute from the L2 cache. The effectiveness of the cache blocking technique depends on data block size, processor cache size, and the number of times the data is reused. Cache sizes vary based on processor. An application can detect the data cache size and dynamically adjust cache blocking tile sizes to maximize performance. As a general rule, cache block sizes should target approximately one-half to three-quarters the size of the physical cache.

Cache pre-fetching. Efficient use of cache and memory system can achieve high performance. As the access latency to data not contained in the cache is very high, it is advantageous, when possible, copy the data in the cache before serving, so they are readily available when needed. While the CPU runs iteration “i”, the memory system loads the data iteration “i+1”, using the independence and competition between different sub-systems.

Data alignment. Another important issue is how the data is aligned in memory. The data is moved between the various levels of memory through the so-called “memory bus”. The bus is aligned with the memory, which can move around the block: bus-width “n bits” to read the address from a given $n - 4$ and $n + 4$ should read 2 times: first the block including between 0 and $n - 1$, and then between them $2n - 1$. So if the data is not aligned, that is, its address is not a multiple of its size (e.g. single or double precision) it may take longer to read/write.

Alignment problems may arise when using the “common” instruction. The basic rule is to order the variables by type: the first all variables of type REAL*8, then the REAL*4, and finally the logical ones.

3.2 The pipeline

Consider a generic task, such as a floating point operation: it can be logically divided into different subsequent stages (e.g. mantissa alignment, sum, normalization and rounding). The transaction is considered completed only after passing all the stages.

Dividing a generic task in three steps, i.e. *fetching* (the stage where the data are retrieved), *decoding* and *executing* - and assuming that each of these will require the same amount of time - CPUs can be ranked according to how they can exploit the different phases.

Sequential CPU. Suppose to assign a frequency of 1 GHz and that each stage or sub-step completes a task, such as a floating point calculation, in 3 cycles. This hypothetical computer has a maximum power output, also known as *peak*, amounting to $1,000,000,000 * 1 / 3 = 333$ MFLOPS.

Pipelined CPU. Suppose the same operation is again made in three steps and assume that different independent parts of each chip in the same cycle of operation can fetch, decode and execute one step without interfering with the others. This would constitute a pipelined architecture, which can overlap the execution of different steps of different instructions.

Thus, in steady state (i.e. full pipeline), the processor produces an output per cycle and thus the achieved peak power (at the same clock) is 3 times higher than in the case of sequential CPU.

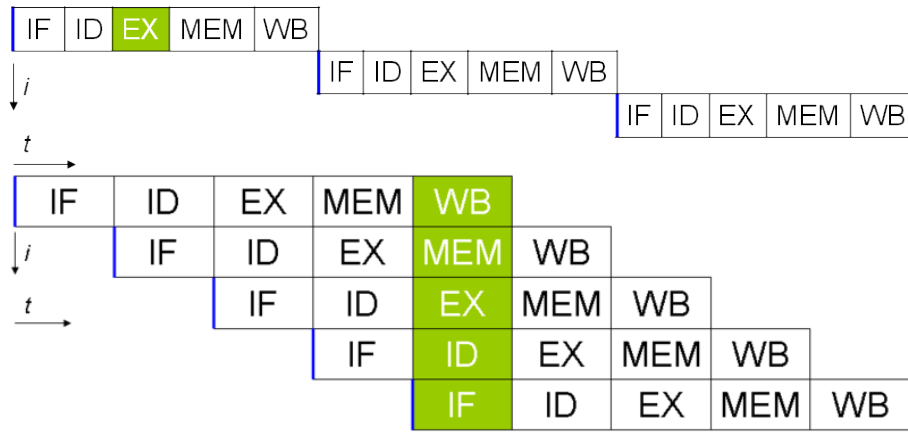


Figure 3.1: No pipeline vs 5 stage pipeline: The processing of an instruction by a processor: IF (Instruction Fetch): Read the instruction from memory, ID (Instruction Decode): Decoding and reading instruction operands from registers, EX (Execution): Executing, MEM (Memory): Activation of memory (only for certain instructions), WB (Write Back): Write the result in the register appropriate.

Loop unrolling. A method to extensively fill the pipeline is the “loop unrolling”, in which an explicit loop can be blown up. An example of it is as follows:

```
do j = 1, nj
do i = 1, ni
a(i,j)=a(i,j)+c*b(i,j)
enddo
enddo
```

that can be rewritten as

```

do j = 1, nj
do i = 1, ni, 2
a(i,j)=a(i,j)+c*b(i,j)
a(i+1,j)=a(i+1,j)+c*b(i+1,j)
enddo
enddo

```

In this way, the pipeline can provide more data to be processed in parallel, since the two operations $a(i, j)$ and $a(i + 1, j)$ are independent from one another. Of course the development of the loop can not be done when there is a dependence between the nested instructions. However in some cases false dependencies are present, that can easily be removed. The fundamental problem is that the way a loop is written may prevent the compiler to perform loop unrolling. For example, a loop like the following one:

```

do j = 1, nj
do i = 1, ni
a(l(i),m(j)) = a(l(i),m(j)) + b(i,j) * c(i,j)
enddo
enddo

```

does not provide any information about the dependence of the data, and thus prevents the compiler from unrolling (even if the iterations are effectively independent).

Another technique to fill the pipelines and reduce the number of load/store is the “total loop merging”, where more loops are merged into a larger loop. Sometimes the reverse operation, called “loop splitting” (which replaces a single loop with several statements with multiple loops containing fewer instructions) is to be preferred, so one can later apply the unrolling optimizations.

The advantage or disadvantage in terms of time offered by this kind of optimizations depends on the particular type of loop in question and on the CPU. In general there is no general rule providing the most appropriate solution to the specific case under examination.

Register usage. A careful use of the registers can improve the performance of a numerical code. For example, it is always convenient to use scalar quantities to store intermediate results, because in this way we suggest to the compiler, which manages the access to the registers, to keep them instead of copying them on the RAM, reducing the total number of load/store operations.

The usage of support vectors as temporary variables is generally not a good approach. This is because important data of the problem are stored in RAM and the use of a vector suggests to the compiler to write the result immediately in RAM by forcing synchronization, even if the result is quickly changed.

On the other hand, an excessive use of variables can lead to a process called “spilling”. In many programming languages, the programmer has the illusion of arbitrarily allocating many variables. However, the compiler must decide how to allocate these variables to a small, finite set of registers. Not all variables are in use at the same time, so some registers may be assigned to more than one variable. However, two variables in use at the same time cannot be assigned to the same register and must be kept in RAM and loaded in/out for every read/write step. All this slows down the execution speed of the compiled program.

“Register pressure” is the term used when there are fewer hardware registers available with respect to the optimal number; higher pressure usually means that more spills and reloads are unavoidable.

Cycles and instructions. Not all operations between integers or real have the same computational weight (also depending on whether one uses single or double precision variables). The table below shows the average cycles required to perform different operations between real and integer variables. The sums and products, both in single and double precision, require a few cycles, and in conditions of full pipeline a CPU is able to complete one for each clock cycle. Other operations do require more cycles (e.g., iterative processes or performing a complex series of calculations). For example, the arithmetical division is an iterative process and therefore does not allow to fill the pipeline since the i -th iteration requires the result of the iteration $(i - 1)$ -th¹. The same is true for operations such as square roots, exponentials, trigonometric functions, exponentiation with real exponent². Usually this kind of transformation, known as “strength reduction”, can be performed by the compiler. As a general rule of thumb, one should avoid, if possible, the use of complex functions and precompute all the quantities he expects to re-use more than once: in such cases it is often convenient to construct a look-up table loading the pre-calculated data and reading them when needed, instead of repeating the calculations many times.

IF-THEN-ELSE Clauses. Another limit to the filling of the pipeline calculation can result from improper use of the IF-THEN-ELSE clause. The clauses IF, of which it is unknown the outcome at compiling time, could end up invalidating the entire pipeline, resulting in a degradation in performance. As a general rule, by using the IF-THEN-ELSE construct it is convenient to put the **less likely** possibility in the THEN case, and the most probable one in the ELSE statement. For example, suppose we want to run the following instructions:

```
IF(option.eq.TRUE) THEN
```

¹Although modern computers often have units specifically dedicated to division

²This is why it is always more convenient to recast the operation “ x^2 ” as a product ($x * x$), rather than as an exponent ($x ** 2$) or even worse ($x ** 2.0$), that is with a real exponent.

Table 3.1: Average cycles per instruction, for the IBM-Power3 processor.

Instruction	32 bit	64 bit
Product of integers	3-4	3-9
Integer division	21	37
Sum or product of real	3-4	3-4
Sum and product of real	3-4	3-4
Divisions	14-21	18-25
Square root of real	14-23	22-31

```

instruction 1
ELSE
instruction 2
ENDIF

```

If for most of the time the expression `option` is TRUE, then then instructions should be re-written in the form:

```

IF(option.eq.TRUE) THEN
instruction 2
ELSE
instruction 1
ENDIF

```

Moreover, it is strongly recommended to avoid IF-THEN-ELSE clauses in extended loops, because their presence tends to empty the pipeline and inhibit the unroll operations.

Branch predictions. Currently, processors implement two different strategies to identify the logical path (“branch prediction”), to limit the penalties induced by IF loops: the “speculative execution” and the “predictive execution”.

With the speculative execution the processor determines which of the two options is most often used and executes it, in case returning on its choice if needed. This is usually implemented in the hardware and it is advantageous only when one of two options is dominant over the other and occurring with some regularity.

With the predictive execution, all possible branch paths are executed. This takes advantage from the fact that in modern CPUs multiple independent logical units are present, that perform in parallel the two possible options associated with the IF clause. Therefore, whenever possible and basing on designed tests, the correct path is kept and all others are eliminated. Furthermore, the condition IF-THEN-ELSE statement has a loop-sided probability in which case other approaches should be considered. The elimination

of branching is an important concern with today's deeply pipelined processor architectures. The reason is that a "mispredicted" branch often costs many cycles.

A possible optimization is like from:

```
if( Condition )
    Case A;
else
    Case B;

to

Case B;
if( Condition)
    not (Case B);
    Case A;
```

Clearly this only works if "not (Case B)" is possible. However, if this is the case, it can be optimized according to the Condition Probability. "not (Case B)" and "Case A" must be merged together for a higher efficiency.

Virtual Memory and Translation Look-aside Buffer. For larger scale memory requirements, the memory access of the processors require rather long times. In practice, direct access to the entire memory does not occur because a translation must be made between the virtual and the physical address of any quantity. The processor allocates a finite number of such translations to a "page" of the main memory which, depending on the system, can be as small as 4 kB or as large as 32 MB. In many cases the page size can be decided by the the code itself.

The processor stores the address conversions for a number of virtual/physical pages in a special cache called the "Translation Look-aside Buffer" (TLB). Accessing a value resident on a page already mapped by the TLB, but not currently resident in cache, may take as many as a few hundred clock cycles, due to the slower speed of main memory relative to the processor. Accessing a value from a page not resident in the TLB requires in addition that a new address translation be calculated, replacing one of those currently resident in the TLB. While some processors have special circuitry to assist in the calculation so that it adds only a small additional delay, others require an intervention by the operating system, resulting in additional delays of a few hundred cycles.

Input & Output. Even the operations of input/output may limit the performance of a code. Some general rules to follow for non-performance penalties are:

- preferentially use write/read instructions in binary or unformatted mode: writing in formatted mode needs a specific function, and hence more time
- write/read an array in one operation and not element by element;
- group the variables to be written/read in order to reduce the I/O;
- remove all read/write in computationally intensive loops;
- always remember to remove all writing statements introduced when debugging!

Chapter 4

Serial and parallel optimization

4.1 The optimization cycle

Regardless of whether an application should be optimized for single-core performance or for scalability, the basic approach is very similar. First, the behavior of the application needs to be monitored. Then, the recorded behavior can be evaluated to draw conclusions for further improvement. This is an iterative process that can be described by a cycle, the so-called performance optimization cycle. When broken down into phases, it can be written as follows:

- Instrumentation
- Measurement
- Analysis
- Presentation
- Evaluation
- Optimization of the code

The user starts with the original (i.e., un-optimized) application, which enters the optimization cycle in the instrumentation phase. Instrumentation describes the process of modifying the application code to enable measurement of performance relevant data during the application run. This can be achieved by different mechanisms, such as source-code instrumentation, automatic compiler-based instrumentation or linking with pre-instrumented libraries. When the instrumented code is executed during the measurement phase, performance data is collected. This can be stored as a profile or an event trace, depending on the desired level of information needed.

Obviously, any additional instruction inserted during the instrumentation stage, and the associated measurement storage, require resources (memory as well as CPU time). However, the perturbation by the additional measurement instructions may be small enough to get a fairly accurate view of the application behavior.

After analyzing the collected data, the result needs to be presented in an analysis report. At this stage, it is important to reduce the complexity of the collected performance data to ease evaluation by the user. If the presented data is too abstract, performance critical event patterns might not be recognized by the user; if it is too detailed, the user might drown in too much data.

In the evaluation phase, conclusions are drawn from the presented information, leading to optimization hypothesis. The user proposes optimization strategies for the application, which are then implemented in the following optimization phase. Afterwards, the effectiveness of the optimization has to be verified by another pass through the performance optimization cycle. When the user is satisfied with the application performance during evaluation, and no further optimization is needed, the instrumentation can be disabled. In the following sections I introduce concepts and tools that form the “toolbox” for the optimization phase.

4.2 Pre-processing and conditional compilation

A pre-processor is a program that processes its input data to produce output to be used as input to another program. The output is said to be a pre-processed form of the input data, which is often used by some subsequent programs (like compilers). Basically it is a stand-alone text processor. The amount and kind of processing done depends on the nature of the pre-processor; some pre-processors are only capable of performing relatively simple textual substitutions and macro expansions, while others have the power of full-fledged programming languages.

Let us now examine the main features that are useful to our purposes. A common example from computer programming is the processing performed on a source code before the compilation. For our aims, the conditional compilation is particularly interesting. This method allows the compiler to produce slight differences in the executable program depending on parameters that are provided during the compilation. This technique is commonly used when these differences are needed to run the software on different platforms, or with different versions of required libraries or hardware. Many programming languages support various methods of doing this. For example, in C (and some languages with a similar syntax) this is done using the C pre-processor and the `#ifdef` directives (all instructions and pre-processor commands begin with a `#` symbol). Many Fortran compilers allow passing

the source code through a C preprocessor (CPP; sometimes also called the Fortran preprocessor, FPP) to allow for conditional compilation. On systems with case-preserving file names, the pre-processor is automatically invoked via the filename extension (.F, .FOR, .FTN, .fpp, .FPP, .F90, .F95, .F03 or .F08). The pre-processor also provides its own language, which can be a powerful tool for programmers.

The use of the pre-processor is advantageous, since it makes programs easier to develop, to read and to edit; moreover, the code is more portable to different machine architectures. The pre-processor also allows the user to “customize” the code through its own language, including directives to be executed, and macros to be expanded. Its primary functions are:

- inclusion of header files
- macro expansion (see Sect. 4.2.3)
- conditional compilation
- line control
- diagnostic

The most common “directives” of the pre-processor are `#define`, `#undef`, `#include` `#if` (conditional inclusion).

4.2.1 Including files

One of the most common task of a pre-processor is to include other files, for example:

```
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\\n");
    return 0;
}
```

The “`#include`” directive brings in (pastes) text from different files during compilation. The `#include` command is very unintelligent and unstructured; the entire text of the file ‘`stdio.h`’ replaces the `#include` directive. There are two possible forms:

```
#include <file> or
#include "file".
```

The first form tells the compiler to look for include-files in the system directories; UNIX stores the files in `/usr/include`; while the form “`file`” looks for a file in the current directory (the one in which the program runs).

4.2.2 The preprocessor C: #if - conditional inclusion

#if evaluates a constant integer expression; one must use #endif to mark the end of the construct; one can also have else (with #else) and else if (with #elif). A common use of #if is as follows:

```
#ifdef - if defined
#ifdef - if not defined
```

These instructions are useful for checking if macros are set (see below), maybe from different program modules and header files or make-files, for example:

```
#ifdef USESTRINGDOTH
    #include <string.h>
#else USESTRINGDOTH
    #include <strings.h>
#endif USESTRINGDOTH
```

or to set the size of the integer between two different environments. We must proceed as follows:

```
#ifdef ENV1
    #define INT_SIZE 16
#else
    #define INT_SIZE 32
#endif
```

Or, one may need to include the file1.h in place of the default.h file when running the program on a specific machine. A macro "SYSTEM" is set to the type of system:

```
#if SYSTEM == MONSTER
    #include <file1.h>
#else
    #include "default.h"
```

4.2.3 Macros

A macro is a fragment of code with a name. Whenever that name is used, it is replaced by the content of the macro. Macros are created with the "#define" directive. There are two types of macros, object-like and function-like. Object-like macros do not take parameters; function-like macros do.

The generic syntax for declaring an identifier as a macro of each type is, respectively:

`#define <identifier> <replacement token list>` (object-like macro) and
`#define <identifier>(<parameter list>) <replacement token list>` (function-like macro)

Examples of two classes of macros:

1. Object-like macros, resembles data objects:

```
#define BUFFER_SIZE 1024
Foo = (char *) malloc (BUFFER_SIZE)
```

Is replaced by:

```
Foo = (char *) malloc (1024)
```

2. Function-like macros, resemble function calls:

```
#define min(X, Y) ((X)<(Y) ? (X):(Y))
x=min(a,b);
=> x=((a)<(b) ? (a):(b));
z=min(a+28, *p);
=> z=((a+28)<(*p) ? (a+28):(*p));
```

4.2.4 Conditionals

Conditionals are directive which make the pre-compiler include or not chunks of code. This in order to:

- adapt the compiled code to the system
- support different versions of the same code (e.g for production run or for debug)
- exclude some code at compiling time but keeping it, as a comment, in the source

Examples of conditionals directives:

```
# if expression (true/false 1/0), # if defined MACRO
```

Expression can include integer constant, arithmetic and logical operators, macros. They can be combined (differently from `#ifdef`):

```
#if defined (__SP5__) || defined (__CLX__)
#if defined BUFSIZE && BUFSIZE >= 1024
# ifdef MACRO
```

defined previously in the code or at compiling time with `-D` option.

```
# ifndef MACRO
# else
# elif expression, # elif defined
# endif
```

The `#if` test can be used at compile-time to look at those symbols and turn on and off which lines the compiler uses

```
#define FOO 1
...
#if FOO
aaa
aaa
#else
bbb
bbb
#endif
```

A useful different version is `#ifdef` / `#ifndef`:

```
#ifdef SP5
time_cpu_sp5(ttot1);
#else
time_cpu_linux(ttot1);
#endif
```

The combination of the pre-processor the Makefile instructions can generate slim and portable programs, and speed up codes substantially, like in the case of EVOL.

4.3 Make Utility

The Make Utility is an intelligent program manager that maintains the integrity of a group of program modules, a collection of programs or a complete system. Make was originally developed for UNIX, but is currently available on most systems.

Consider the problem of maintaining a large number of source files:

```
main.c f1.c ... fn.c
```

so we usually compile with the command:

```
cc -o main main.c f1.c ... fn.c
```


However, if we are aware that some files have already been previously completed and their sources have not changed since we have tried, and we want to save these files to compile the total linking object code, the command is as follows:

```
cc -o main main.c f1.c ... fi.o ... fj.o ... fn.c
```

We can use the option “-c” for C compiler (or Fortran) to create a file “.o” in the provided module.

For example:

```
cc -c main.c
```

will create a file main.o.

In this case there is no need to provide any link to libraries, as this will be fixed at the link stage of compilation. However, compiling a whole code in this way may take quite a long time:

- it is time consuming to fill out a form.c file: if the module has already been compiled before, and it has not been changed, there is no need for a new compilation. It is sufficient to link the object files. However, book-keeping of files may not be easy.
- It is rather difficult (and error-prone) to type a long string of compiling command line. Usually, many user-files and library files must be linked: it can be difficult to remember the correct sequence of operations.

If we use the Make Utility, all this is automatically done. In general, this utility provides only a recompilation of the modules that have the object files older than the source files.

Programming the Make-file.

Programming the Make-file is straightforward. Basically, we need to write a sequence of commands that describe how our program (or system of programs) can be built from the source files. The construction sequence is described in a file “Makefile”, which contains dependency rules and rules of interpretation.

A dependency rule has two parts (a left and a right side, separated by “:”):

```
left\_side : right\_side
```

The left side is formed by the name of a target (name of program or system files) that must be created, while the right side provides the names of the files upon which the target file (for example, source files, header files or data file) is made. If the target file is not updated with respect to its constituent parts, the rules of interpretation (or construction) and addition are applied. For instance, in a typical C program, when running the Make-file the following actions are undertaken:

1. reading the makefile: this shows what items and library files need to be linked and what header files and source must be compiled to create each object file
2. date and time of each object file are checked with those of each source file and header files on which it depends. If any source file or header file is newer than the object, then the files have been modified since the last compilation, and therefore the object file is recompiled
3. once all the object files have been checked, are controlled date and time of all object files and compared with those of the executable file. If any of the files is more recent, then the object file will be recompiled.

We point out that the make file can obey any command that is typed on the command line. So we can use the Make-file to do more than compiling a source module of the system. For example, we could make backups of files, run programs if data files are changed, or clean directories.

Creating a makefile. Here is an example Make-file:

```
prog prog.o f1.o f2.o

c89 prog.o f1.o f2.o -lm etc..

prog.o: header.h prog.c

    c89 -c prog.c

f1.o: header.h f1.c
c89 -c f1.c f2.o: ... ..
```

Make will interpret the file as follows:

1. prog depends on 3 files: prog.o, f2.o, and f1.o. If any object file has been modified since the last compilation, the file must be linked again.
2. prog.o depends on 2 files. If these are changed, prog.o must be recompiled, and the same goes for f1.o and f2.o.

The last 3 commands in makefiles are called “explicit rules”, since the files are listed in the commands with their own name.

We can also use “implicit rules”, which allow us to generalize our rules and store what has been typed. For example one can take:

```
f1.o: f1.c

    cc-c f1.c
```

```
f2.o: f2.c

cc -c f2.c
```

and generalize it with the following command:

```
.c .o: cc -c \<
```

This reads like “Source `_extension`.target `_extension`: command”. “`<`” is the abbreviation for the name file “`.c`”. One can add comments in a makefile using the symbol “`#`”; in this way, all the characters following `#` line are ignored.

Make, as anticipated, has many internal commands similar or even identical to those in UNIX¹. An important feature for our purposes is the ability to define variables and macros.

Creating Macros with the Make-file. Using Make one can define macros to store the names of source files, object file names, compiler options, and link libraries. Macros are easy to define, for example:

```
SOURCES = main.c f1.c f2.c

CFLAGS = -g -C

LIBS = -lm

PROGRAM = main
OBJECTS = (SOURCES: .c=.o)
```

where `(SOURCES: .c=.o)` transforms extensions “`.c`” of the files listed in `SOURCES` in extensions “`.o`”.

Reference to a macro in Make is via the command type `$(macro_name)`; for example:

```
$(PROGRAM) : $(OBJECTS),$(LINK.C) -o $@ $(OBJECTS) $(LIBS)
etc.
```

It is important to note that:

- `$(PROGRAM) : $(OBJECTS)` - creates a list of dependencies and objects;
- using a internal macro, ie `$`.

There are many internal macros, among which we recall

¹Some examples are: `break` dates, `mkdir`, `type`, `chdir`, `mv` (move or rename), `cd`, `rm` (remove), `cp` (copy), `path`

- `$*` - part of the file name in the current directory (without .suffix)
- `$@` - full name of the current object
- `$<` - file object `.c`

Note that one can also merge two or more text variables to create a new one.

I have successfully used these instructions to create a Makefile for our code easily readable and accessible by a generic compiler.

Running Makefile. Typing “make” in the command line, UNIX automatically searches for a file named “Makefile” (note the capital letter, while everything else is tiny). The Makefile in the current directory will be executed. One inhibit this search by simply typing the command “make -f make_filename”.

4.4 Compilation of a code

In this section we describe the compilation procedure and how this affects the optimization process. The compiler translates a code from a high-level language such as Fortran or C into assembler instructions directly executable by the processor.

Transformations performed by the compiler. Starting from the original code, the compiler produces an executable through different steps that are summarized as follows.

- Pre-compiling: the code undergoes minor changes, such as the removal of comments, the inclusion of the “include” file and, if required, the conditional compilation.
- Lexical analysis: identifies the operations to be done, which are the variables to use, etc.
- Parsing: the code is broken into individual statements and the syntax is checked.
- Generation of the intermediate code: an intermediate code is written in which most of the optimization is made.
- Generation of the assembler (Back-end): the final assembler is written.

Generally, this sequence is identical for different languages (e.g. Fortran or C). The most critical step for a good performance is the generation of the intermediate code where the individual blocks of instructions are changed and rearranged to obtain the best performance.

A generic compiler typically performs the following operations to maximize the performance:

- Dead and redundant code removal
- Copy propagation
- Code motion
- Strength reduction
- Common subexpression elimination
- Register allocation
- Loop pipelining/unrolling
- Cache blocking
- Loop interchange/reordering
- Function in-lining

In the following I will try to describe advantages and disadvantages of these actions.

Dead code removal. This is one an easy operation: removing all the lines in the code that are not accessed during the run (for example, old versions, redundant commands, etc.). This way, all unnecessary jumps in the program are removed leaving having an instruction stream as linear as possible.

Copy propagation. By changing the assignments of constants some dependencies between instructions are removed. For example, the instructions

```
B = A
D = C
E = B*D
```

have a dependency between the data: the third statement must be performed after the second instruction is completed. Renaming the constants as:

```
B = A
D = C
E = A*C
```

the dependence on the third statement is removed, thus better filling the pipeline.

Code motion. When there are no dependencies, the compiler move instructions from one part of the code to another (code motion). In this way one can extract from a loop those operations that do not depend on the index of the loop itself. For example the loop:

```
do i = 1, n
sum = sum + a(i)*b(j)*c(k)
enddo
```

can be rewritten as:

```
temp = b(j)*c(k)
do i = 1, n
sum = sum + a(i)*temp
enddo
```

This will save n products $b(j)*c(k)$ using the temporary support variable `temp`, which is calculated only once outside the loop. It is evident that an incorrect use of parentheses can inhibit the code motion. For instance writing the previous loop as:

```
do i = 1, n
sum = sum + (a(i)*b(j))*c(k)
enddo
```

prevents the compiler from computing $b(j)*c(k)$, unless high optimization options (that can alter the order of operations established by the parentheses) are used.

Strength reduction. Operations such as $B=A**3$ are computationally expensive because the operation actually performed is $x^y = \exp(y \ln x)$. If the above expression is written as $B = A*A*A$ only two products are needed to perform the computation, instead of two calls to complex functions (logarithm and exponent). Typically, a compiler is able to make these changes, at least if the exponent is not too large. Note that writing the same expression as: $B=A**3.0$ the compiler may have problems because now the exponent is not an integer but a real quantity.

Finally, typing

```
C = 3.0
.....
B = A**C
```

strength reduction is completely inhibited.

Another typical example is the strength reduction division. As a rule of thumb, it is always better to pre-compute all divisions in advance. For instance, when normalizing quantities it is best practice to multiply by the reciprocal, instead of dividing by the normalization factor.

Common sub-expressions elimination. Variables are often used more than once in the same algorithm, for intermediate calculations. A compiler is able to perform the necessary transformations, however only up to a certain

extent, because it works on “windows” (or pieces of code) of finite size. So if a loop is very long, the compiler cannot determine all the expressions that are reused. This will reflect onto loss of performance. In addition, the compiler cannot consider functions as repeated variables: calling twice the same function, even with the same argument, will not always return the same result (typical example are the functions generating random numbers). Another example are the conversions from polar to cartesian coordinates:

```
x = c*sin(a)*cos(b)
y = c*sin(a)*sin(b)
x = c*cos(a)
```

They are more efficiently written as follows:

```
temp = c*sin(a)
x = temp*cos(b)
y = temp*sin(b)
z = c*cos(a)
```

where a product and a call to a trigonometric function are replaced by an additional load and a store operation, the variable `temp`. In general the use of functions inhibits the motion code and/or loop unrolling and merging. In such cases, the “code motion” is entirely left to the programmer ability.

Register allocation. The compiler also takes care of allocating the registers for operations. The registers in the CPU are limited in number and must be suitably used and allocated. As already mentioned, introducing too many intermediate quantities, or using incorrect bracketing, can generate the so-called “register spilling” (number of required registers larger than physically available). In this case, the “missing registers” are stored in the first-level cache, thus increasing the number of load and store operations. Obviously the changes that bring benefits in terms of time, by definition, are not portable, as they are based on a particular compiler and the existing number of registers. Changing compiler and/or architecture can easily lead to drastic changes in the computing time and performance.

Loop unrolling, splitting & merging. Another class of transformations that the compiler is able to do is the unrolling, merging or splitting of the loops. Obviously, these transformations are allowed only if there are no correlations among the elements of the iterations. In this case the presence of a subroutine, a function or an I/O within the body of the loop inhibits this type of transformations. Loop merging is also inhibited by the presence of instructions or I/O functions timing between the two loops to be fused together. High levels of compiler optimization usually yield the correct unrolling, especially for loops with few instructions. But in the case of loops very full-bodied with lots of sub-loops or routines, the compiler is not able to edit/rewrite the loop.

Cache blocking. A good compiler is also able to do the blocking of a loop. Obviously, for very complex loops or indirect addressing in most cases the compiler is not able to produce good results. Operations such as blocking or unrolling are performed by a compiler only at high levels of optimization.

Loop reordering. In the absence of dependencies between the instructions of a loop, and if the loop body is quite simple, the compiler is able to alter the order of the loop in order to facilitate access to data, minimizing the stride. Sometimes it is enough to have a number of loops indented as a few units, to inhibit this type of transformation.

In-lining. Sometimes a simple loop that contains a `CALL` statement to a subroutine may not allow optimizations such as “common subexpression elimination” or unrolling. Furthermore, the jump in the flow of instructions (following the call) can also empty the pipeline thus slowing down the process. To maintain a modular structure in the code and not lose too much performance at the same time, the compiler should be instructed to make in-lining of subroutines: this is to replace the call of the subroutine’s body itself, and then compile it all together. As a drawback, in-lining produces very large executables large, very long compilation times, and often worse performance.

Accuracy of results. Operations such as “strength reduction” or “common subexpression elimination” alter the numerical results: this is because the algebra has finite precision, and is not associative with respect to the sum. Therefore, in the expression: $A + B + C \neq A + C + B$ the two sums may differ in the least significant digits due to different rounding, and therefore changing the order in which operations are executed may change, even significantly, the final result.

Altering the order of operations may result in errors and loss of accuracy in case of cancellation. Putting $A = -1$, $B = 1$, $C = 1e - 9$ for single precision, we have: $(A + B) + C = 1e - 9$ while $A + (B + C) = 0$. This is a classic example of a cancellation. If the divisor is the result of another quantity, in the second case we would have an indeterminate result (overflow) due to the reordering of instructions.

This problem often causes a code to crash. At low level of optimization, typically used to develop and validate a code, the order of operations is not changed and so the problem does not occur. At high levels, however, operations are reordered, and cancellation may take place. In this case it may be helpful to instruct the compiler to force that the operation order does not alter the code and the result in turn (e.g. `-qstrict` command in SP6²).

²Cluster at CINECA, Bologna, Italy

4.4.1 What a compiler cannot do

. In this section we will list the operations that a compiler cannot perform:

- Analyze and optimize very long codes or routines, unless ones enables the IPA (but this may take some time to be completed);
- Understand and explain dependencies between data with indirect addressing;
- Strength reduction of non-integer powers;
- Common sub-expression elimination functions;
- Unrolling / Merging / Blocking functions / subroutines;
- Unrolling / Merging / Blocking loops with boundaries unknown at the time of compilation;
- Altering the order of explicit parentheses if high levels of optimization;
- Make in-lining of functions, if not explicitly stated.

In general, it is wise to apply all the possible simplifications, in order to allow the compiler to do only the ones that are too complex, and those necessary to build the assembler.

Portability. By definition, the changes that facilitate or help the compiler to produce an efficient executable are non-portable, since they rely on the goodness of the compiler, and the hardware features of the machine in use. So it is unlikely that changes on a machine will produce the same benefits on another. On the other hand it is true that, in general, the optimization of a specific architecture does not involve major disadvantages with different architectures (provided they are comparable in technology).

Summary. We have examined a number of aspects that when suitably implemented, cured, avoided or taken into account can foster high performance for a given numerical code. To summarize:

- Access to data with unit stride;
- Simplify the addressing in memory;
- Reduce computationally expensive operations (division, square roots,...);
- Use temporary variables as scalars rather than vectors;
- Restrict the use of IF THEN ELSE constructs, (and use them properly)
- Restrict the use of subroutine and function within the loop;
- Avoid the I/O in the critical loop;

- Use whenever possible, the system libraries that are well optimized.

It should be emphasized that in principle a compiler is able to achieve high performance, and in most cases only the poor writing of the code limits the ability of the compiler to optimize it.

4.5 Algorithms

In this section we discuss about algorithms and how their choice may have a heavy impact on performance.

Complexity of algorithms. If from a “mathematical” point of view different algorithms to solve a problem can be completely equivalent, from the computational point of view they can require extremely different amounts of time. Accurate implementations can reduce the computational time even by several orders of magnitude. To illustrate the point, we discuss the case of algorithms suited to solve 3D elliptic equations:

Table 4.1: Evolution of the complexity of different algorithms for solving 3D elliptic equations.

Method	Year	Complexity
Gaussian Elimination	1945	N^7
SOR (suboptimal)	1954	$8N^5$
SOR (optimal)	1960	$8N^4 \log(N)$
Cyclic Reduction	1970	$8N^3 \log(N)$
Multigrid	1978	$60N^3$

The total complexity between two methods such as Gaussian elimination and multigrid decreased as N^4 . Therefore, it is wise to choose among the available algorithms the one which is best suited to the problem under examination.

Shape of the data structure. The data structure that is chosen can have an effect on the performance of a code. Here follows a classic example in dynamics, which is however valid in various fields. Suppose there are a number of particles, each characterized by three functions f_x, f_y, f_z (one per coordinate in the ordinary 3D space) and undergoing the variation dx, dy, dz . The operation can be written in the following way

```

real fx(1:n), fy(1:n), fz(1:n)
....
do i=1, n
  ii = index(i)
  ....
  fx(ii) = fx(ii) + dx....

```

```

fy(ii) = fy(ii) + dy....
fz(ii) = fz(ii) + dz....
enddo

```

This involves access to three different data streams, one for each spatial component. Access to each data stream will probably generate a cache miss, since the vectors $\mathbf{fx}$, $\mathbf{fy}$, $\mathbf{fz}$ are, for realistic cases, larger than the cache, and the particles in their evolution will also mix with each other, completely losing the spatial locality. In contrast, a data structure like this:

```

real f(1:3,1:n)
....
do i=1, n
  ii = index(i)
  ....
  f(1,ii) = f(1,ii) + dx....
  f(2,ii) = f(2,ii) + dy....
  f(3,ii) = f(3,ii) + dz....
enddo

```

generates only one cache miss per iteration, having a single data stream. Of course, it must be noted that using an efficient data structure for part of the code may be inefficient in another part of the same code. Therefore, one should evaluate case by case whether and how to change the data structure and evaluate the overall effect.

Size of the problem. During the development and validation stages, it is generally convenient to work on “small” problems. However, if such problems are “too small” they may be completely worked out in the cache, thus hiding possible problems of cache-misses or cache-thrashing which will reveal when dealing with larger datasets. In these situations, the use of an efficient algorithm may generate the impression of being fast enough, while the speed is problem-dependent.

Store or recalculate? There is no pre-determined rule saying when to keep a quantity in memory or when to recalculate it. In general, if the calculation of this quantity is simple, then it is probably advantageous to recalculate if all the ingredients are already in the cache.

Algorithms and precision In some cases, relaxing the constraints on the precision can lead to some advantages in terms of computing time. For example, math libraries that compute complex functions are implemented in order to obtain maximum precision for all values of the argument for which this function is defined. Assuming that the range that the variable will assume is known *a priori*, one can replace the library function with another approximate, but faster, function. For instance, if one heavily uses trigonometric functions in the neighborhood of a fixed value x_0 , it is possible to develop a series of power up to the desired accuracy, i.e. one can replace

```

do i=1, n
....
a(i) = sin(x)
....
enddo

```

with its series expansion, in a small interval around $x_0 = 0$:

```

do i=1, n
....
a(i) = a1*x+a3*x*x*x.....
....
enddo

```

This method may indeed benefit of symmetries or special formulas for trigonometric functions, or cancellation of terms in the series expansion. However, mathematical functions are in general not very efficient and the time required for their execution changes a lot. The situation is illustrated in the tables below.

Table 4.2: Cycles spent on average for different functions, measured on an old system (HP EV6 to 500 Mhz.)

Function	Cycles
$\sqrt{x}$	16
$1/x$	4
$x^{1.5}$	175
$\sin(x)$	83
$\log(x)$	85
$\exp(x)$	5

It is clear that it may be cheaper, in terms of spent cycles, to avoid the sequence $y=x**1.5$ and to choose instead the equivalent form $y=\text{SQRT}(x*x*x)$. This requires, according to the table, about 20 cycles instead of 175. Obviously the two expressions give the same result in infinite precision, but not in finite precision, so these optimizations might produce slightly different final results.

4.6 Scientific libraries

In scientific computing there are a large number of libraries, i.e. collections of functions and subroutines that can help solve many different problems. These collections provide the user with ready-to-use pieces of code, and with high degree of modularity. They greatly simplify the work, are generally

Table 4.3: Average time spent on different functions normalized to the addition, measured on BUD

Function	<i>REAL</i> * 4	<i>REAL</i> * 8
+	1.00	1.00
*	1.03	1.01
$\sqrt{x}$	1.60	2.50
$\frac{1}{x}$	1.46	2.50
$x^{1.5}$	37.86	33.54
$\sqrt{xxx}$	1.47	2.42
$\sin(x)$	7.09	8.30
$\cos(x)$	6.36	8.81
$\tan(x)$	12.21	13.72
$\text{atan}(x)$	7.72	11.70
$\log(x)$	8.56	16.74
$\exp(x)$	5.97	6.35

Table 4.4: Average time spent on different functions normalized to the addition, measured on SP6

Function	<i>REAL</i> * 4	<i>REAL</i> * 8
+	1.00	1.00
*	1.10	1.10
$\sqrt{x}$	5.39	5.19
$\frac{1}{x}$	1.02	1.06
$x^{1.5}$	20.38	18.55
$\sqrt{xxx}$	6.11	5.17
$\sin(x)$	7.04	6.32
$\cos(x)$	5.16	4.96
$\tan(x)$	9.09	8.37
$\text{atan}(x)$	6.82	5.94
$\log(x)$	8.05	6.85
$\exp(x)$	8.88	8.33

quite cured, tested and highly performing. However, they may sometimes have some problems of portability.

Optimized math libraries. There are also optimized math libraries, extremely fast in calculating mathematical functions and sometimes vectors (in the sense that they very efficiently calculate multiple values simultaneously). See for instance

- Intel vml
- IBM mass & vmass (automatically used -O5.)

However, one should keep in mind that speed is often achieved at the expense of accuracy!

4.7 Evaluation of parallel performance

The goal of the “scaling theory” is the evaluation of the performance of parallel systems. Some natural questions arise when you first approach these topics: how to measure the performance of different parallel systems (eg. IBM SP Power 5, SGI Altix, Linux or clusters)? How to compare the performance obtained on different machines? Is the evaluation the same if the program adopts the MPI or OpenMP paradigm? In addition to this, requests concerning the maximum size of the problem to be solved, the accuracy of the solution, the execution time, the speed-up, and the efficiency of computer exploitation are present.

Parallel systems. When speaking of a “parallel system”, we mean the implementation of a parallel algorithm on a specific parallel architecture.

Speed Up and Scalability. Some definitions:

Speedup: speed increase (i.e., decrease of computational time) due to the use of n processors:

$$S(n) = \frac{T(1)}{T(n)},$$

where $T(1)$ is the total serial time, and $T(n)$ is the total time using n processors. If $S(n) = n$ then the speed-up is linear (ideal case), $S(n) < n$ is the speed-up of a real case, and if $S(n) > n$ is super-linear speed-up (due to cache effects).

Efficiency. Efficiency is the effective exploitation of the parallel machine and is defined as follows:

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{(T(n) * n)} \quad (4.1)$$

where $E(n) = 1$ is the ideal case, $E(n) < 1$ is the real case, and $E(n) \ll 1$ when problems are present.

Scalability. It is the ability of a code to keep its efficiency on a parallel machine at increasing size of the problem and number of processors in use, while maintaining unchanged the calculation time.

Parallel overhead. The efficiency of the “real parallel systems” is not maximized, because there are always sources of overhead such as

- Extra computations parallel algorithm, than the best sequential algorithm;
- Inter-processor communication needs;
- Imbalance of workload;
- Communication / synchronization: messages and synchronization in a parallel program introduce a computational cost absent in the serial case.

Granularity of Computation. Granularity is the amount of work in the parallel task, a qualitative measure of the relationship between computations and communications. The design goal is to determine the right granularity for the parallel tasks while avoiding load imbalance and communication overhead. The amount of work for each parallel task, or granularity, of a parallel application greatly affects its parallel performance.

When threading an application, the first step is to partition the problem into as many parallel tasks as possible. The second step is to determine the necessary communication in terms of data and synchronization. The third step is to consider the performance of the algorithm. Since communication

and partitioning are not free operations, the operations often need to combine partitions. The combination step is the process of determining the best granularity for the application.

Fine-grained parallelism. Few computational operations between communications (small value): individual tasks are relatively small in terms of code size and execution time. The data is transferred among processors frequently, in amounts of one or a few memory words. It can be easy to balance the load but there is a significant overhead of communication. It is easier to balance the workload of a large number of small tasks but too many small tasks can lead to excessive parallel overhead. If granularity is too fine performance can suffer from increased communication overhead.

Coarse-grained parallelism. Many computational tasks between communications (great value): individual tasks are relatively big in terms of code size and execution time. It is the opposite of the previous: data are communicated infrequently, after larger amounts of computation. If granularity is too coarse performance can suffer from load imbalance. It may be difficult in turn to balance the computational load. If communications are negligible, then there will probably be an increase in overall performance. Therefore, coarse granularity is usually best. Increasing granularity too much can create load imbalance.

Serial and parallel fraction. The “serial fraction” of a program (SF), is the ratio between the time spent during the execution in the intrinsically sequential code and the total execution time. On the other hand, the “parallel fraction” of a program (PF) is the ratio between time spent in the parallel section and the total time. Obviously $SF = (1 - PF)$.

Amdahl and Gustafson Laws. They are related to the definitions of speedup and scalability and seek to answer the following question from two different points of view: What is the maximum achievable speed-up in a parallel code?

Amdahl's law: Defined $T = T_S + T_P$ the total execution time, where T_S is the time spent in the serial part of the code (not parallelized/parallelizable), T_P is the time spent in the parallel part, and $T_n = T_S + \frac{T_P}{n}$ the execution time on n processors, the speed-up with n processors will be: $S_n = T_1/T_n = (T_S + T_P)/(T_S + T_P/n)$. For $n \gg 1$, $S_n \simeq \frac{(T_S + T_P)}{T_S}$. Therefore, there is an “asymptotic limit” to the speed-up!

In reality, this maximum speed-up overestimates the scalability, because this model does not take into account:

- Time spent in communication/synchronization;
- Resource contention (latencies and communication time);
- Any idle time due to load imbalance;
- Improvements of the memory hierarchy and the number of processors;

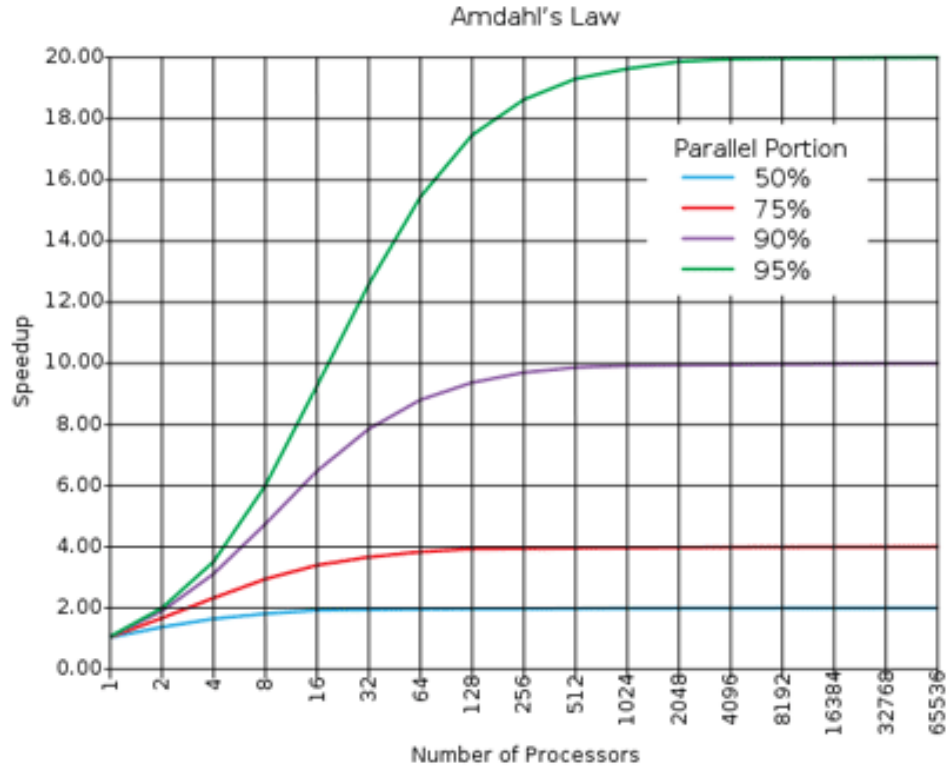


Figure 4.1: Maximum speed-up reached as a function of parallel fraction.

Amdahl's law (Amdahl 1967) is based on fixed workload or fixed problem size. It implies that the serial part of a program does not change with respect to the machine size (i.e., the number of processors). If we take into account the weight of the communications, the real situation is shown in the Figure 4.2. In general, the importance of computations and communications decrease/increase with the number of processors.

Gustafson's Law: Gustafson's Law (Gustafson 1988) says that problems with large, repetitive datasets can be efficiently parallelized. Therefore, Gustafson's Law contradicts Amdahl's law, which poses a limit on the speed-up that parallelization can provide. Set $S_n = n - T_S \times (n - 1)$, where n is again the number of processors, S is the speedup, and T_S the serial (i.e., non-parallelizable) part of the process. Gustafson's law addresses the shortcomings of Amdahl's law, which does not scale the availability of computing power as the number of machines increases. Gustafson's Law proposes that programmers set the size of problems in relation to available equipment in order to solve problems within a certain (reasonable) amount time. Therefore, if faster (more parallel) equipment is available, larger problems can be solved in the same time. The impact of Gustafson's law was to shift research

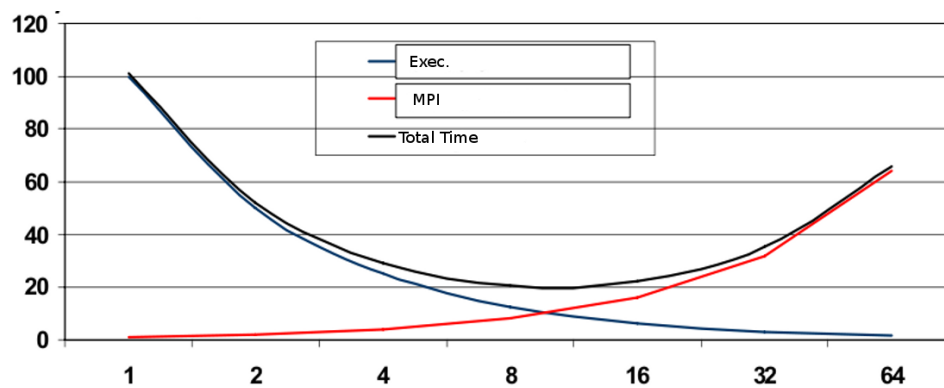


Figure 4.2: Combined effect of computing and communications on the speedup.

goals to select or reformulate problems so that solving a larger problem in the same amount of time would be possible.

How to parallelize a code? Given these limitations it is clear that parallelizing a code is not always enough to obtain faster results, and it is also clear that lots of work are left to the skilful programmers (Dongarra 2003), requiring both scientific and computer science expertises. In short, one has to evaluate

- Data independence;
- Order of operations;
- Order (and efficiency) of memory access;
- Data/information moving between tasks;
- Communication efficiency;
- Synchronization;
- Load Balancing.

Chapter 5

Debugging a code

Let us suppose that the parallel code suited to solve our physical problem is ready and running, and we realize that the computing resources at disposal seem to be inadequate. Is this always true? Is our code actually exploiting all the computer power, or does it use a fraction of it? To answer these questions, we must evaluate the performance in an objective way.

Of course, first of all one must be sure that the code runs correctly and properly in different situations and with different inputs. It is only through detailed tests that hidden pitfalls can be brought into evidence. To give an example, our code, even when producing correct results, turned out to be affected by an incorrect deallocation of arrays. Since the simulations often lasted for months, they were interrupted soon beyond the phase of “dynamic formation and strong star formation activity”. The error in the array deallocation was not noticed because it showed out only in certain circumstances and in the simulations carried out to the end.

The basic tools to pin down most common types of errors are the standard serial debugging routines available in any development environment. A “debugger” is a series of programs that are used to test and debug the “target” program. The code to be examined is run on a simulator of instructions (ISS), whose greatest merit is to stop when specific conditions are encountered. It goes without saying that running a code under debug is much slower than executing the same code directly. Some debuggers offer two modes of operation - full or partial simulation, to limit this drawback.

A “crash” happens when the program cannot normally continue because of a “bug” in the target code is met. When the program crashes, or reaches a preset condition, a typical source-level or a symbolic debugger show the position of the crashing point in the original code. Most popular debuggers also offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping the program to examine the current state at some event or specified instruction by means of a breakpoint, and tracking the values of some variables.

Some debuggers can modify the state of the program while it is running, rather than merely watch it. It may also be possible to continue execution bypassing a crash or logical error. The lack of a debugger, once accustomed to using one, has been often said to “make you feel like a blind man in a dark room looking for a black cat that is not there”.

However, even with a good debugger at disposal, it is often very difficult to track down runtime problems in complex multi-threaded or distributed systems. Most current mainstream debuggers (or debugging engines), such as GDB and DBX, provide console-based command line interfaces. Debugger front-ends are popular extensions of debugging engines that provide IDE integration, program animation, and visualization features.

Serial debuggers. These programs can be either a command line or a functional graphic tool. To mention a few, we recall GDB, DBX, IDB (Intel debugging), VALGRIND for memory check. Some of these are commercial with open source GNU licenses. Among others we recall

- DDD is the standard front-end of the GNU Project. This is a complex tool that works with most common debuggers (GDB, JDB, Python debugger, Perl, Tcl, and others);
- KDBG - Part of the KDE development tools;
- XXGDB - X-Window front end for GDB and DBX debugger, Eclipse etc..

The methods used in this study are very similar to the above ones, and are of immediate use (the graphical ones in particular). An error in the code can be easily located, displaying variables, by advancing the program step by step, or stopping it at a given line, or when certain conditions are verified (e.g., a variable assumes a specific value).

Parallel MPI/OpenMP debuggers. For parallel computing, things are complicated by the presence communications (MPI in our case) and in this case one needs a special “parallel debugger”. There are only two commercial parallel debuggers, none of which is open source: TOTALVIEW and DDT. I used them both on the CINECA facilities, and the second one on my personal computer.

The primary difference of serial debugging is the ability to follow individual processes, to track the values taken by a variable on different processors, to monitor the communications, to check whether the bug happens regularly or if there are blockages (deadlocks), which could hardly be detected unless they are explored by hand throughout the entire source code. With the aid of these tools, one can find and solve all the major problems faced by parallel codes.

The greatest drawback with using this software is that the problem to be solved often occurs in advanced stages of a simulation, so that monitoring

the code in the debug mode is mandatory. Debugging slows down the code and for long lasting calculations debugging turns out not to be viable.

Typical debugging procedure. Normally, the first step in debugging is to try to reproduce the bug itself. This can be a non-trivial task, for example in the case of parallel processes or some unusual software bugs. After the bug is reproduced, the input of the program may need to be simplified to make it (if possible) easier to debug. One can examine a code line by line (values of variables, plus the call stack) and track down the origin of the problem(s). Alternatively, tracing can be used. In simple cases, tracing is just a few print statements, which output the values of variables at certain points of program execution.

Compiler flags for debugging. It is often said that adding a `-g` flag to the compiling list in the Makefile is enough to trace all the errors. Unfortunately the command `-g` is not enough. This “naive” approach does not guarantee success. Furthermore, if well-compiled with the correct options, a crashing code still provides enough information to solve the errors without needing deep debugging. For a correct debugging procedure one has to:

- Compile with the `-g` flag to save symbols table in the executable program.
- Use the option `-O0` to eliminate the optimization¹
- Enable the capture of floating-point exception. The IEEE² is able to manage the following exceptions:
 - Invalid operation: result *NaN*
 - Division by zero: result $\pm \text{inf}$
 - Overflow: result $\pm \text{inf}$ or $\pm N_{\text{max}}$
 - Underflow: result ± 0 or $\pm N_{\text{min}}$ or subnormal
 - Inexact: result is “properly (according to the rules)” rounded.
- Check the limit of the array (bounds check)
- Use a “backtrace” to know sequence: call the routine that in turn calls a function that goes wrong at a certain line.

Usually all these options are not enabled, for reasons of speed, so one must explicitly enable them. To set up a code to debug using INTEL, IFORT and GFORTRAN, a command line is for example

¹The compiler may alter the semantics of the code to high levels of optimizations. With the `-O0` flag we have the correspondence between the source code and the compiled version. So, in case of error (or error message) we know the exact line of source where the problem occurs.

²Institute of Electrical and Electronics Engineers defining the standard for Floating-Point Arithmetic.

```
gfortran -O0 -g -fbacktrace -fbounds-check -ffpe-trap=invalid,  
zero, overflow, underflow prog.f90 -o prog.x
```

and

```
ifort -O0 -g -traceback -fp-stack-check -check bounds -fpe0  
prog.f90 -o prog.x
```

Without the right debugging options (compiler flags), even wrong codes can run quietly yielding erroneous results. In particular, codes with divisions by zero and/or arrays with indices exceeding the limits may have a bizarre behavior, but in most cases will not crash. In fact, writing or reading past the last element of an array, overwriting or reading the memory that was reserved for other data etc., causes the program to behave differently at each run, giving erroneous results or crashing at different points each time for no obvious reason, or, even worst, seeming to work properly. Great care should be paid to these controls, which should always be enabled during the development phase and also during the later stages. Only after some time of testing in different conditions, the code can be considered reasonably free from errors and these flags can be removed to gain speed.

Chapter 6

Profiling

When developing a code, one of the last steps is to make it as fast as possible. To avoid waste of time optimizing routines which will be rarely used, one needs to know which parts of the code require most of the computing time. This is done with a technique called “profiling”. The program is run under control of a profiling tool, which gives the time distribution among the various functions executed in the run. Examining the code’s profile, one may know where to optimize. The success of the optimization is verified by another profiling step and the process is iterated.

‘Inclusive cost’ and ‘Self/Exclusive cost’. In general, profiling distinguishes between time spent in a procedure and time spent in sub-procedures called by the latter. As functions can call each other, it makes sense to distinguish the cost of the function itself (‘Self Cost’, sometimes also referred to as ‘Exclusive’ costs) and the cost including all called functions (‘Inclusive Cost’).

Profiling Tools. A variety of profiling and execution analysis tools exist for both serial and parallel programs. They range widely in usefulness and complexity:

- Simple command line timing utilities
- Fortran and C timing routines
- Profilers
- Execution trace generators
- Graphical execution analyzers - with/without trace generation
- Both real-time and post-execution tools

Most of the more sophisticated and useful tools have an associated learning curve, and would require a long presentation. In the following we limit ourselves to briefly review a important aspects of this topic.

Motivation. Writing large-scale parallel and distributed scientific applications that make optimum use of computational resources is a challenging problem. Very often, resources are under-utilized or used inefficiently. The factors which determine a program's performance are complex, interrelated, and often hidden to the programmer. Some of them are listed by category below.

- Application Related Factors:

- Algorithms
- Dataset Sizes
- Memory Usage Patterns
- Use of I/O
- Communication Patterns
- Task Granularity
- Load Balancing
- Amdahl's Law

- Hardware Related Factors:

- Processor Architecture
- Memory Hierarchy
- I/O Configuration
- Network

- Software Related Factors:

- Operating system
- Compiler
- Preprocessor
- Communication protocols
- Libraries

Because of these challenges and complexities, performance analysis tools are essential to optimizing an application's performance. They can help in understanding what your program is "really doing" and suggest how program performance should be improved.

Performance Considerations and Strategies. The most important goal of performance tuning is to reduce a program's wall clock execution time. Reducing resource usage in other areas, such as memory or disk requirements, may also be a tuning goal. Performance tuning is an iterative process used to optimize the efficiency of a program. It usually involves finding program's *hot spots* (parts of the code that use a disproportionately

high amount of processor time) and eliminating the *bottlenecks* (parts of the code that use processor resources inefficiently and therefore cause unnecessary delays) in them.

It is wise to use profiling tools and techniques to find out which parts of the code offer the greatest potential performance increase *before* starting the tuning process and targeting the most time consuming, frequently executed portions of a program for optimization. Also, it is wise to optimize the underlying algorithms: an extremely fine-tuned $O(N * N)$ sorting algorithm may perform significantly worse than a untuned $O(N \log N)$ algorithm. For data dependent computations, benchmark based on a variety of realistic (both in size and in values) input datasets is also recommended. Furthermore, it is wise to maintain consistent input data during the fine-tuning process, and to take advantage of compiler and preprocessor optimizations when possible. Last but not least, one should keep in mind when to stop - there are diminishing returns in successive optimizations. As an example, consider a program with the following breakdown of execution time percentages for the associated parts of the program:

Table 6.1: Example of time ripartition of the computational time between different procedures.

Procedure	% CPU Time
main	13%
procedure1	17%
procedure2	20%
procedure3	50%

A 20% increase in the performance of procedure3 results in a 10% performance increase overall. A 20% increase in the performance of main results in only a 2.6% performance increase overall.

So, properly measuring the performance of a code is a crucial step as the change of the code should be applied according to the measures accumulated. Errors or incomplete information may nullify all subsequent optimization work.

As a general summary, based on experience, it is recommended to:

- Use multiple test cases;
- Use different profiling tools;
- Choose realistic reference cases;
- Make tests with various dimensions of the problem of how to estimate the relative weight changes of the various subroutines;

- Note the weight of the input/output in the test;
- Possibly work on dedicated processor;
- Possibly work on several different architectures.

Timers. In addition to the difference of time taken from the start and end date, and the running time of the file, a wide variety of timers are available on different platforms. The table below compares a number of these. A detailed description for each one follows in Tab. 6.2.

Table 6.2: Timers.

Timer	Usage	Languages	Portable?
time-command	shell/script	any	yes
gettimeofday	subroutine	C/C++	yes
MPI_Wtime	subroutine	C/C++, Fortran	yes
AIX Trace Facility	shell/script/subroutine	any	no

6.1 Time command

The time command returns the total execution time of your program. The format of the output is different for the Korn shell and the C shell but the basic information is similar¹:

- Real time: the total wall clock (start to finish) time your program took to load, execute, and exit.
- User time: the total amount of CPU time your program took to execute.
- System time: the amount of CPU time spent on operating system calls in executing your program.

Example of Korn shell time output:

```

real    0m2.58s
user    0m1.14s
sys     0m0.03s
```

gettimeofday() The gettimeofday() routine returns the time in seconds and microseconds since the reference epoch (Midnight, January 1, 1970). It can be inserted anywhere within a C program and used to determine the

¹The system and user times are defined in different ways across different computer architectures.

start and end times of code fragments. Fortran programs must perform an interlanguage call.

MPI Timing Routines This timer is provided by the MPI implementation, called `MPI_Wtime`, which returns the elapsed wall clock time in seconds (double precision) on the calling processor. MPI also includes a routine, called `MPI_Wtick`, which returns the resolution in seconds (double precision) of `MPI_Wtime`. For example, the resolution on IBM SP systems varies by model but is microsecond level or better.

6.2 Profilers

Profiling is achieved by instrumenting either the program source code or its binary executable form. Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, instruction set simulation, and so on. The methodology of the profiler itself classify the profiler as event-based, as statistical, as instrumentation, or as simulation.

The output of a profiler may be a summary profile of the observed events, often shown against the source code statements where the events occur, or a stream of recorded events (a trace). For sequential programs, a summary profile is usually sufficient, but performance problems in parallel programs (waiting for messages or synchronization issues) often depend on the time relationship of events, thus requiring a full trace to get an understanding of what is happening.

The size of a (full) trace is linear to the program's instruction path length, making it somewhat impractical. The results of measurements collected profiling can be presented in different formats:

Flat profiler computes the average call times, from the calls, and do not break down the call times based on the callee or the context.

Call-graph profiler shows the call times, and frequencies of the functions, and also the call-chains involved based on the callee.

The most common methods of data gathering are the *statistical profiler* (or sampling profiler), and the *event-based profiler*.

A sampling profiler probes the target program at regular intervals using operating system interrupts. The resulting data are not exact, but a statistical approximation. The actual amount of errors is usually more than one sampling period. Sampling profiles, though typically less numerically accurate and specific, allow the target program to run (almost) at full speed.

In practice, sampling profilers can often provide a more accurate picture of the target program's execution than other approaches, as they are not intrusive to the target program, and thus don't have as many side effects (such as on memory caches or instruction decoding pipelines).

Others profilers *instrument* the target program with additional instructions to collect the required information. Instrumenting the program can

Table 6.3: Example of a prof output.

Name	%Time	Seconds	Cum.secs	#Calls	msec/call
.fft	51.8	0.59	0.59	1024	0.576
.main	40.4	0.46	1.05	1	460.
.bit_reverse	7.9	0.09	1.14	1024	0.088
.cos	0.0	0.00	1.14	256	0.00
...	...	...	...	...	...

cause changes in the performance of the program, potentially causing inaccurate results and typically always slowing it. GPROF is an example of a profiler that uses both instrumentation and sampling: instrumentation is used to gather caller information and the actual timing values are obtained by statistical sampling.

Instrumentation. As already mentioned “instrumentation” refers to the inserted additional instructions to collect data. There are several ways to do the instrumentation of a program:

- Manual: Performed by the programmer, e.g. by adding instructions to explicitly calculate runtimes, simply count events or calls to measurement APIs such as the Application Response Measurement standard.
- Automatic source level: instrumentation added to the source code by an automatic tool according to an instrumentation policy.
- Compiler assisted: for example, “gcc -pg ...” for GPROF
- Runtime instrumentation: the code is instrumented directly before execution. The program run is fully supervised and controlled by the tool. Examples: VALGRIND.

prof utility. The **prof** utility is included in most UNIX systems. It is used to profile the program execution at the procedure level. **prof** displays the following information:

- the name of each procedure - in descending order of processing activity
- the percentage of the program’s CPU time used by each procedure
- the execution time in seconds for all references by each procedure
- the number of times the procedure was called
- the average time in milliseconds for a call to each procedure.

Sample of a **prof** output (abbreviated):

How to use **prof**:

1. Compile your program with the `-p` option.
2. Run the program. When the run ends, a file called `mon.out` should have been created, containing runtime statistics. If you are running a parallel program (compiled with `mpcc` or `mpxlf`) you will have multiple files differentiated by the task ID which created them, such as `mon.out.0` `mon.out.1` `mon.out.2`, etc.
3. For serial usage, view the profile statistics with `prof` by typing `prof` at the shell prompt in the same directory where the program is run
4. For parallel usage, run `prof` with the `-m` option followed by the names of the desired `mon.out.X` files. You may view any single file or any combination.

gprof utility. The **gprof** utility is included in most UNIX systems. Like **prof**, it is used to profile program execution at the procedure level. Unlike **prof**, it profiles procedures according to their call graphs. **gprof** displays the following information:

- The parent of each procedure.
- An index number for each procedure.
- The percentage of CPU time taken by that procedure and all procedures it calls (the calling tree).
- A breakdown of time used by the procedure and its descendents.
- The number of times the procedure was called.
- The direct descendents of each procedure.

ngranularity: Each sample hit covers 4 bytes. Time: 1.17 seconds

index	time	self	descendents	called/total called+self called/total	parents name index children
[1]	99.1	0.44	0.72	1/1	__start [2]
		0.44	0.72	1	.main [1]
		0.59	0.13	1024/1024	.fft [3]
		0.00	0.00	256/256	.cos [6]
		0.00	0.00	256/256	.sin [7]
		0.00	0.00	8/8	.gettimeofday [11]
		0.00	0.00	7/7	.printf [16]
		0.00	0.00	1/1	.atoi [31]
		0.00	0.00	1/1	.exit [33]

6.6s					<spontaneous>
[2]	99.1	0.00	1.16		__start [2]
		0.44	0.72	1/1	.main [1]

How to use gprof:

1. Compile the program with the `-pg` flag.
2. Run the program. When the run is completed file called `gmon.out` should exist, containing runtime statistics. If you are running a parallel program you will have multiple files differentiated by the task ID which created them, such as `gmon.out.0` `gmon.out.1` `gmon.out.2`, etc.
3. For serial usage, view the profile statistics with `gprof` by typing `gprof` at the shell prompt in the same directory where the program is run.
4. For parallel usage, view the profile statistics by typing `gprof` followed by the name of your executable and the `gmon.out.X` files you wish to view. You may view any single file or any combination.

The primary difference between `prof` and `gprof` timings are: for `prof`, CPU time assigned to each procedure does not include the CPU time used by procedures further down the calling tree; for `gprof`, this CPU time is included. Thus `gprof` should be used for programs which make calls to library procedures. Both `prof` and `gprof` show CPU time, not real time. Disk delays due to input and output are not included.

Xprofiling. Xprofiler is an X Windows based profiling tool distributed with IBM's Parallel Environment software. It is based upon the `gprof` profiling utility, however it provides a graphical representation of profile data in addition to all of the usual `gprof` reports. Graphical information is organized into 3 main components:

- Library Cluster Boxes
- Function Boxes
- Call Arcs

Xprofiler can profile at both the subroutine level, and at the source statement level. Profiling also includes any calls made to library functions. Filters, zooming, and other options allow you to limit displays to only those portions of the call tree you are interested in analyzing.

Fig. 6.1 below illustrates the displays and reports of an ideal case:
Using Xprofiler:

1. Compile and link your program with both of the options: `-g -pg`. The `-g` option enables source statement profiling and `-pg` turns profiling on.
2. Run your serial or parallel code as usual. When it has completed, you will find one statistics file for each task. Serial codes produce `gmon.out`. For parallel jobs, the files will be called `gmon.out.0`, `gmon.out.1`, `gmon.out.2` and so on.

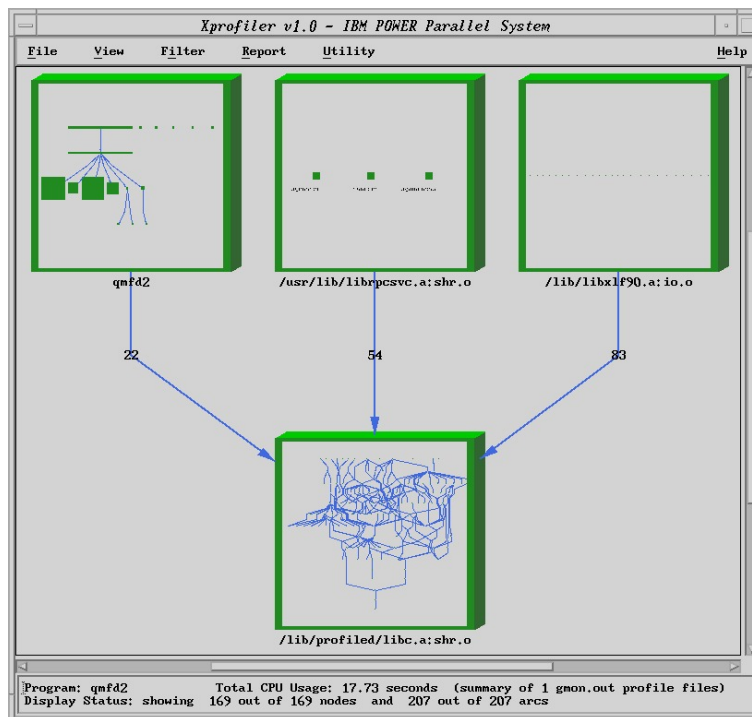


Figure 6.1: Xprofiler: entire call tree, including the calls to the libraries.

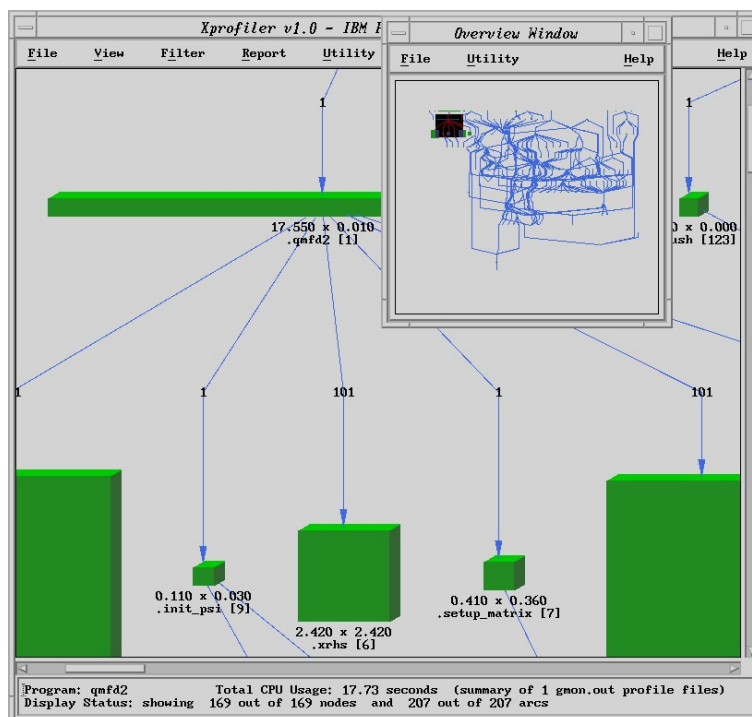


Figure 6.2: Zoom-in view of main program

3. Invoke Xprofiler
4. Use Xprofiler's pull down menus and hidden menus to accomplish desired actions, such as: zooming-in/out, examining function statistics, producing reports, collapsing/hide library information and more.

Intel Trace Analyzer and Collector. *Intel Trace Collector* for MPI applications produces tracefiles that can be analyzed with Intel Trace Analyzer performance analysis tool (see Intel[®] Trace Analyzer and Collector Documentation). It records all calls to the MPI library and all transmitted messages, and allows arbitrary user-defined events to be recorded. This will enable the tracing of all calls to MPI routines, as well as all explicit message-passing. Instrumentation can be switched on or off at runtime, and a powerful filtering mechanism helps to limit the amount of the generated trace data.

Intel Trace Analyzer is a graphical tool that displays and analyzes event trace data generated by the Intel Trace Collector. It helps detecting performance problems, programming errors and understanding the behavior of the application.

Using Trace Collector for MPI is straightforward: relink the MPI application with the appropriate profiling library and run it following the usual procedures of the system. This will generate a trace-file suitable for use with Intel Trace Analyzer, including records of all calls to MPI routines as well as all point-to-point and collective communication operations performed by the application. Intel Trace Collector is automatically initialized within the execution of the `MPI.Init()` routine. During the execution of the `MPI.Finalize()` routine, the trace data collected in memory or in temporary files is consolidated and written into the permanent trace file(s), and Intel Trace Collector is terminated. To get more detailed information about the application, Intel Trace Collector provides several methods to instrument the application. This will allow arbitrary user-defined events to be traced; in practice, it is often very useful to record the applications entry and exit to/from subroutines or regions within large subroutines. In the following, we explain how to compile, link and execute MPI applications with Intel Trace Collector. The easiest way is to use the compile scripts provided by the Intel MPI Library: simply use the command line argument `-trace`. All it requires is having sourced/executed the respective `itacvars` script. MPI programs linked with Intel Trace Collector as described in the previous sections can be started in the same way as conventional MPI applications. The trace data is stored in memory during the program execution, and written to disk at MPI finalization time. By default the data are stored in several "Structured Trace File" (STF). Normally if a MPI application fails or is aborted, all trace data collected so far is lost: the Trace Collector needs a working MPI to write the trace file, but the MPI standard does not guarantee that MPI is still operational after a failure. In practice most MPI implementations just abort

the application because of Premature Exit: one or more processes exit without calling `MPI_Finalize()`. Intel Trace Collector is able to gather and store statistics about the function calls and their communication.

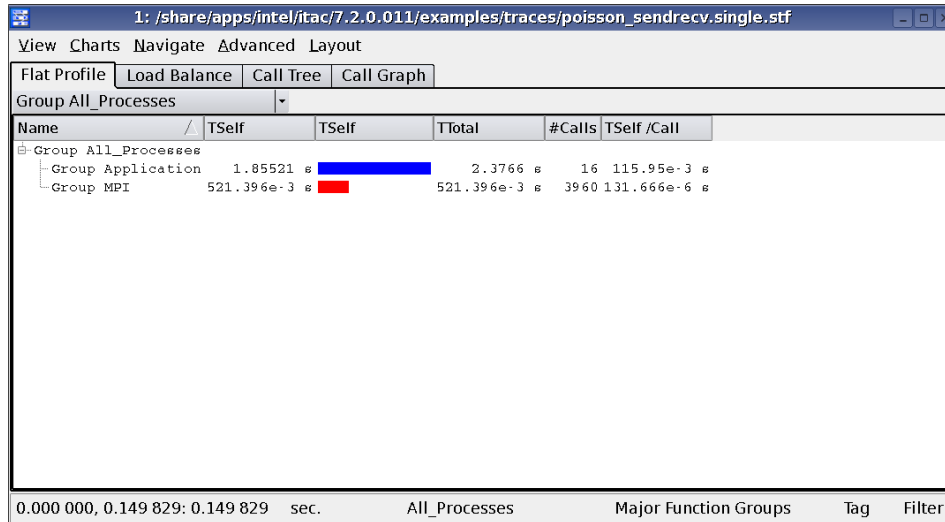


Figure 6.3: Example of a Trace Analyzer screenshot. Blue: time spent in calculations; red: time spent in communications.

Trace Analyzer: Charts. In the Intel Trace Analyzer, the graphical tool - “Charts” - that can be generated and modified according to user’s needs are of paramount importance.

The Charts are graphical or alphanumerical diagrams that are parameterized with a time interval, a process grouping, a function grouping and an optional filter. Together they define the structure in which data is presented and the amount of data to be displayed. The Charts supported by Intel Trace Analyzer are divided into:

1. Timelines:

- the Event Timeline
- the Qualitative Timeline
- the Quantitative Timeline and
- the Counter Timeline.

2. Profiles:

- the Function Profile
- the Message Profile
- the Collective Operations Profile.

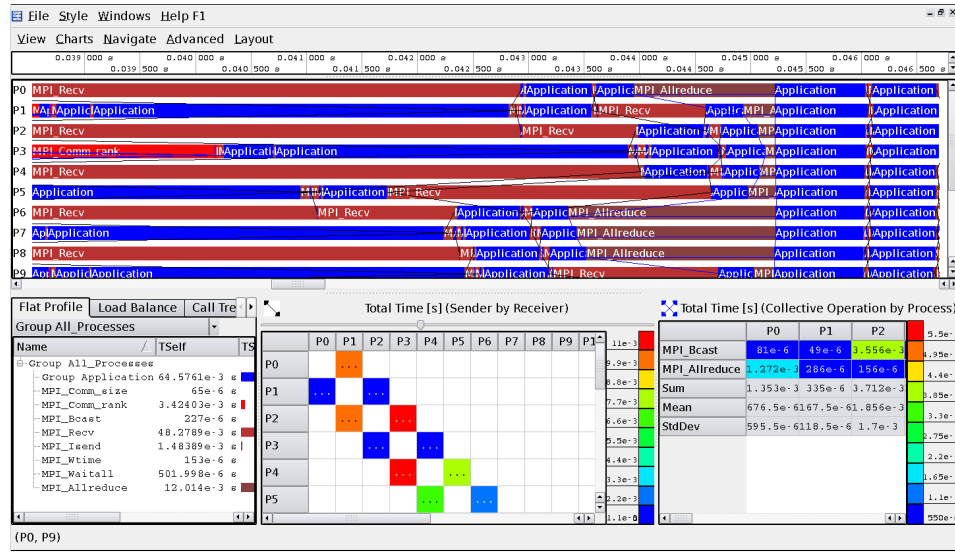


Figure 6.4: Example of a Trace Analyzer screenshot with some Charts. Top: “event timeline” chart. Bottom left: “function profile”. Bottom middle: “message profile”. Bottom right: “collective operations profile”.

While the Timelines show trace data in graphical form over a horizontal axis representing the runtime, the Profiles show statistical data. To see, for example, which particular MPI functions are used in the program, one can do right-click on MPI in the Event Timeline and choose Ungroup Group MPI. The Function Aggregation of the View changes so that the MPI functions are no longer aggregated into the Function Group MPI, but are shown individually. To compare two trace files, Intel Trace Analyzer offers the so-called Comparison View. A new Comparison View is opened that shows an Event Timeline for each file and a Comparison Function Profile Chart that shows a profile computed from both trace files. One can create a Comparison View that compares one time interval and one process group against another time interval and another process group of the same trace file. Other useful applications of the Comparison View include **scalability analysis** where one compares two runs of the same unmodified program with different processor counts and try to find out which functions scale well and which suffer from the Amdahl’s Law.

Scalasca is an open-source toolset for the analysis of scalable performance of large-scale parallel applications (see Scalasca User Guide). It measures and analyzes the runtime behavior. The analysis identifies potential performance bottlenecks, in particular those concerning communication and synchronization, and offers guidance in exploring their causes.

SCALASCA targets mainly scientific and engineering applications based

on the programming interfaces MPI and OpenMP, including hybrid applications based on a combination of the two. The tool has been specifically designed for use on large-scale systems.

Making applications run efficiently at larger scales is often thwarted by excessive communication and synchronization overheads. Especially during simulations of irregular and dynamic domains, these overheads are often enlarged by wait states that appear in the wake of load or communication imbalance when processes fail to reach synchronization points simultaneously.

Overview of the Scalasca Functionality. SCALASCA supports measurement and analysis of the MPI, OpenMP and hybrid MPI/OpenMP programming constructs most widely used in highly scalable HPC applications written in C, C++ and Fortran on a wide range of current HPC platforms. To evaluate the behavior of parallel programs, SCALASCA takes performance measurements at runtime to be analyzed after program termination. The user of SCALASCA can choose between two different analysis modes: performance overview on the call-path level via runtime summarization, and in-depth study of application behavior via event tracing.

In the **profiling mode**, SCALASCA generates aggregate performance metrics for individual function call paths, which are useful to identify the most resource-intensive parts of the program, the resulting reports can be interactively explored in a graphical browser (see Fig. 6.5).

As an alternative, the resulting traces (**event trace mode**) can be visualized in a third-party time-line browser such as “Vampir” to study the detailed interactions among different processes or threads.

The use of SCALASCA involves three phases: program instrumentation, execution measurement and analysis, and analysis report examination. The SCALASCA command provides action options that invoke the corresponding commands `skin`, `scan` and `square`. These actions are:

1. `scalasca -instrument` is used to insert calls to the SCALASCA measurement system into the application’s code, either automatically, semi-automatically or by linking with pre-instrumented libraries
2. `scalasca -analyze` is used to control the measurement environment during the application execution, and to automatically perform trace analysis after measurement completion if tracing was enabled
3. `scalasca -examine` is used to postprocess the analysis report generated by the measurement runtime summarization and/or post-mortem trace analysis, and to start SCALASCA’s analysis report examination browser CUBE3.

Instrumentation. As usual, before any performance data can be collected, the target application needs to be instrumented, i.e., at specific important points (events) during the application run, special measurement calls

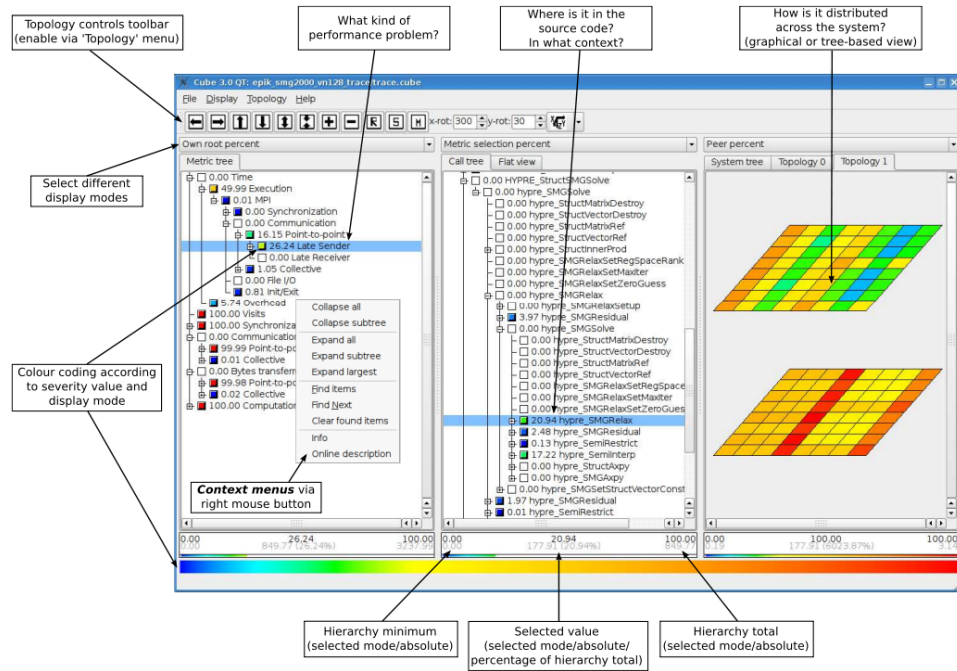


Figure 6.5: SCALASCA overview.

have to be inserted. All the necessary instrumentation of user, OpenMP and MPI functions is handled by the SCALASCA instrumenter, which is accessed through the `scalasca -instrument` command:

- Pre-pend `scalasca -instrument` (or `skin`) and any other instrumentation flags to the compile/link commands.
- By default, MPI and OpenMP operations are automatically instrumented, and many compilers are also able to instrument all routines found in source files.
- To enable manual instrumentation use EPIK.

For example, the original command

```
mpicc -c foo.c
mpicxx -o foo foo.cpp
mpif90 -openmp -o bar bar.f90
```

in the SCALASCA instrumentation command becomes:

```
\textsc{Scalasca} -instrument mpicc -c foo.c
\textsc{Scalasca} -inst -pomp mpicxx -o foo foo.cpp
skin mpif90 -openmp -o bar bar.f90
```

Measurement and Analysis. The SCALASCA measurement collection & analysis nexus accessed through the `scalasca -analyze` command integrates the following steps:

- measurement configuration
- application execution
- collection of measured data
- automatic post-mortem trace analysis (if configured)

When the measurement has completed, the measurement archive directory contains various log files and one or more analysis reports. For MPI measurements, MPI time and message and file I/O statistics are included. For OpenMP measurements, OpenMP-specific metrics are calculated. Hybrid OpenMP/MPI measurements contain both sets of metrics. If hardware counter metrics were requested, these are also included in the summary report.

- Prepend `scalasca -analyze` (or `scan`) to the usual execution command line to perform a measurement with `scalasca` runtime summarization and associated automatic trace analysis (if applicable).
- Each measurement is stored in a new experiment archive which is *never overwritten* by a subsequent measurement.
- By default, only a runtime summary (profile) is collected (equivalent to specifying `-s`).
- To enable trace collection & analysis, add the flag `-t`.
- To analyze MPI and hybrid OpenMP/MPI applications, use the usual MPI launcher command and arguments.

Examination of the Analysis Report. The results of the automatic analysis are stored in one or more reports in the experiment archive. These reports can be processed and examined using the `scalasca -examine` command on the experiment archive:

```
scalasca -examine epik_< title>
```

Post-processing is done the first time that an archive is examined, before launching the CUBE3 report viewer. The CUBE3 viewer can also be used on an experiment archive or CUBE file:

```
cube3 epik_<title>
cube3 <file>.cube
```

However, keep in mind that no post-processing is performed in this case, so that only a subset of SCALASCA analyses and metrics may be shown.

- To interactively examine the contents of a SCALASCA experiment, after final processing of runtime summary and trace analyses, use `scalasca -examine` (or `square`) with the experiment archive directory name as argument.
- To get a textual score output (using the `cube3` score utility) instead of the graphical user interface, add the `-s` flag.
- If multiple analysis reports are available, a trace analysis report is shown in preference to a runtime summary report: other reports can be specified directly or selected from the File/Open menu.
- Results are displayed using three coupled tree browsers showing
 - Metrics (i.e., performance properties/problems)
 - Call-tree or flat region profile
 - System location (alternative: graphical display of physical/virtual topologies, 1D/2D/3D Cartesian only)
- Analyses are presented in trees, where collapsed nodes represent inclusive values (consisting of the value of the node itself and all of its child nodes), which can be selectively expanded to reveal exclusive values (i.e., the node self value) and child nodes.
- When a node is selected from any tree, its severity value (and percentage) are shown in the panel below it, and that value distributed across the tree(s) to the right of it.
- Selective expansion of critical nodes, guided by the colour scale, can be used to hone in on performance problems.
- Each tree browser provides additional information via a context menu (on the right mouse button), such as the description of the selected metric or source code for the selected region (where available).

Metric severity values can be displayed in various modes:

- Mode Absolute: absolute value in the corresponding unit of measurement.
- Root percent: percentage relative to the inclusive value of the root node of the corresponding hierarchy.
- Selection percent: percentage relative to the value of selected node in corresponding tree browser to the left.

- Peer percent: percentage relative to the maximum of all peer values (all values of the current leaf level).
- Peer distribution: percentage relative to the maximum and non-zero minimum of all peer values.
- External percent: similar to Root percent, but reference values are taken from another experiment.

Manual source-code instrumentation. Region or phase annotations manually inserted in source files can augment or substitute automatic instrumentation, and can improve the structure of the analysis reports to make them more readily comprehensible. These annotations can be used to mark any sequence or block of statements, such as functions, phases, loop nests, etc., and can be nested, provided that every enter has a matching exit. Source files annotated in this way need to be compiled with the `-user` flag given to the SCALASCA instrumenter, otherwise the annotations are ignored. Fortran source files need to be preprocessed

Experiment archives. SCALASCA measurement and analysis artifacts are stored in unique experiment archive directories, with the `epik` prefix. Note that experiment archives are not overwritten or otherwise corrupted by subsequent measurements, an existing measurement archive will block new measurements that would have the same experiment archive name. The SCALASCA measurement & analysis nexus automatically generates a default experiment title from the target executable, compute node mode (if appropriate), number of MPI processes etc.

CUBE3. CUBE3 is a generic user interface for presenting and browsing performance and debugging information from parallel applications. The left panel shows performance properties of the execution, the middle panel shows the call-tree or a flat profile of the application, and the right tree either shows the system hierarchy consisting of machines, compute nodes, processes, and threads or a topological view of the application's processes and threads. All tree nodes are labeled with a metric value and a colored box which can help identify hotspots. The metric value color is determined from the proportion of the total (root) value or some other specified reference value. A click on a performance property or a call path selects the corresponding node. This has the effect that the metric value held by this node (such as execution time) will be further broken down into its constituents. That is, after selecting a performance property, the middle panel shows its distribution across the call tree. After selecting a call path (i.e., a node in the call tree), the system tree shows the distribution of the performance property in that call path across the system locations. A click on the icon left to a node in each tree expands or collapses that node. By expanding or collapsing nodes in each of the three trees, the analysis results can be viewed on different levels of granularity. In addition to interactive exploration of analysis reports with the SCALASCA

examiner, they can be processed with a variety of CUBE algebra tools and utilities.

CUBE3 algebra utility. CUBE3 also provides a number of algebra utilities which are command-line tools that operate on analysis reports. Multiple analysis reports can be averaged with `cube3_mean` or merged with `cube3_merge`. The difference between two analysis reports can be calculated using `cube3_diff`. Finally, a new analysis report can be generated after pruning specified call trees and/or specifying a call-tree node as a new root with `cube3_cut`. The latter can be particularly useful for eliminating uninteresting phases (e.g., initialization) and focussing the analysis on a selected part of the execution. As performance tuning of parallel applications usually involves multiple experiments to compare the effects of certain optimization strategies, CUBE offers a mechanism called performance algebra that can be used to merge, subtract, and average the data from different experiments and view the results in the form of a single experiment. The algebra is an ideal tool to verify and locate performance improvements and degradations likewise. The algebra includes three operators (`diff`, `merge`, and `mean`) provided as command-line utilities which take two or more CUBE files as input and generate another CUBE file as output. In addition, the `cube3_score` tool can be used to estimate trace buffer requirements based on a given CUBE file, typically from a previous summary experiment.

Valgrind: cachegrind/callgrind. Valgrind is an instrumentation framework for building dynamic analysis tools (see Valgrind User Manual). It comes with a set of tools each of which performs some kind of debugging, profiling, or similar task that helps improve the programs. A number of useful tools are supplied as standard. Those we have used are:

- Memcheck is a memory error detector. Useful to correct the programs.
- Cachegrind is a cache and branch-prediction profiler. It helps the programs run faster.
- Callgrind is a call-graph generating cache profiler. Gathers some information that Cachegrind does not.
- Massif is a heap profiler. It helps the programs use less memory.

Valgrind is designed to be as non-intrusive as possible. It works directly with existing executables. One does not need to recompile, relink, or otherwise modify the program to be checked.

Cachegrind. Cachegrind simulates how the program interacts with a machine's cache hierarchy and (optionally) branch predictor. It simulates a machine with independent first-level instruction and data caches (I1 and D1), backed by a unified second-level cache (L2). This exactly matches the configuration of many modern machines. Cachegrind gathers the following statistics (abbreviations used for each statistic is given in parentheses):

- I cache reads (Ir, which equals the number of instructions executed), I1 cache read misses (I1mr) and LL cache instruction read misses (ILmr).
- D cache reads (Dr, which equals the number of memory reads), D1 cache read misses (D1mr), and LL cache data read misses (DLmr).
- D cache writes (Dw, which equals the number of memory writes), D1 cache write misses (D1mw), and LL cache data write misses (DLmw).
- Conditional branches executed (Bc) and conditional branches mispredicted (Bcm).
- Indirect branches executed (Bi) and indirect branches mispredicted (Bim).

Note that D1 total accesses is given by $D1mr + D1mw$, and that LL total accesses is given by $ILmr + DLmr + DLmw$. These statistics are presented for the entire program and for each function in the program. You can also annotate each line of source code in the program with the counts that were caused directly by it. On a modern machine, an L1 miss will typically cost around 10 cycles, an LL miss can cost as much as 200 cycles, and a mispredicted branch costs in the region of 10 to 30 cycles. Detailed cache and branch profiling can be very useful for understanding how your program interacts with the machine and thus how to make it faster.

To run Cachegrind on a program prog, run:

```
valgrind --tool=cachegrind prog
```

and to get a function-by-function summary, type:

```
cg_annotate <filename>
```

To get annotate source files just use `-auto=yes` option.

Callgrind. Callgrind is a profiling tool that records the call history among functions in a program's run as a call-graph. By default, the collected data consists of the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions, and the numbers of such calls. Optionally, cache simulation and/or branch prediction (similar to Cachegrind) can produce further information about the runtime behavior of an application. The profile data is written out to a file at program termination. For presentation of the data, and interactive control of the profiling, two command line tools are provided:

callgrind.annotate. This command reads in the profile data, and prints a sorted lists of functions, optionally with source annotation. For graphical visualization of the data, we use KCachegrind, that makes it easy to navigate the large amount of data that Callgrind produces.

callgrind.control. This command enables you to interactively observe and control the status of a program currently running under Callgrind's control, without stopping the program. To run Callgrind on a program prog, type:

```
valgrind --tool=callgrind prog
```

for "the assembly code annotation" and to see even the conditional jumps, we need to add options

```
--dump-instr=yes
```

```
--trace-jump=yes
```

Currently the results can only be displayed by KCachegrind.

Kcachegrind. KCachegrind is a visualization tool for the profiling data generated by Cachegrind and Calltree (they profile data file format is upwards compatible). Despite being programmed first with browsing the data from Cachegrind and Calltree in mind, there are converters available to be able to display profile data produced by other tools. Besides a list of functions sorted according exclusive or inclusive cost metrics, and optionally grouped by source file, shared library or class. KCachegrind features various views for a selected function, namely:

- a call-graph view, which shows a section of the call graph around the selected function,
- a tree-map view, which allows nested-call relations to be visualized, together with inclusive cost metric for fast visual detection of problematic functions,
- source code and disassembler annotation views, allowing to see details of cost related to source lines and assembler instructions.

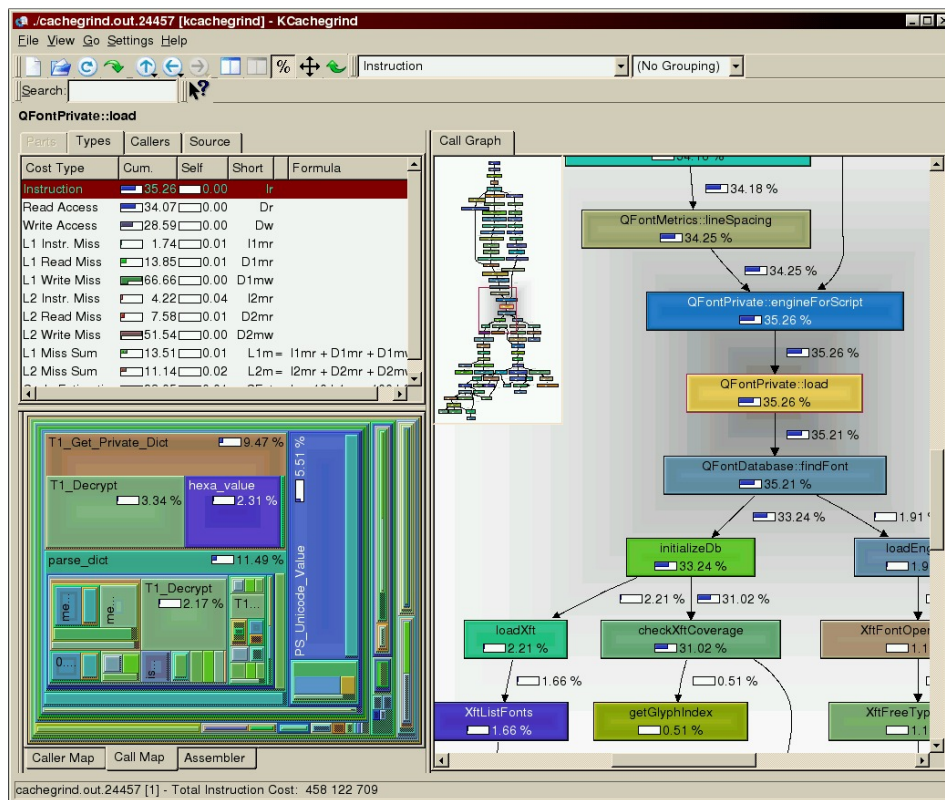


Figure 6.6: Example of visualization with Kcachegrind.

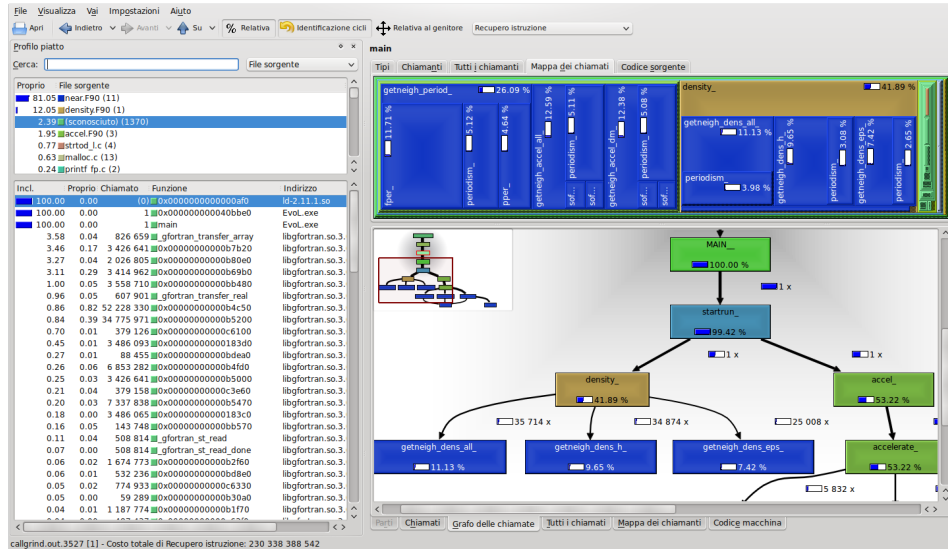


Figure 6.7: Part of the call to the Tree routine in EVOL displayed with Kcachegrind. Data have been collected with Callgrind. In the boxes, the percentage of time spent and the number of calls in the given branch are shown.

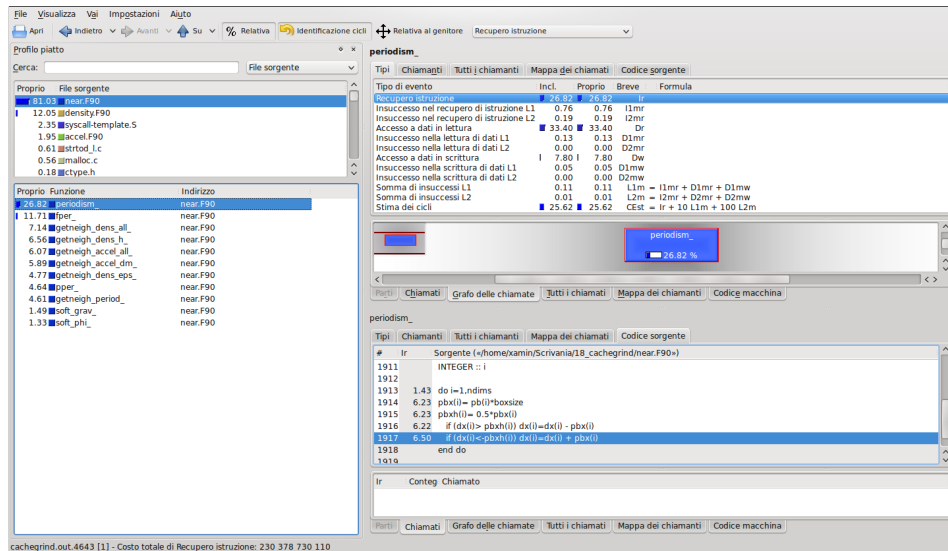


Figure 6.8: Kcachegrind screenshot displaying data collected with Cachegrind. On the right side are the routines with the larger number of cache-miss, divided by type. Bottom left: the line in the source code where the trouble is localized.

Chapter 7

Scientific Visualization

Over the past few years, the role of visualization for scientific purpose has grown up enormously. Astronomy makes an extended use of visualization techniques to analyze data, and scientific visualization has become a fundamental part of modern researches in Astronomy. The scientific community expressed a great interest in scientific visualisation tools. A strong boost in this direction was determined by the introduction of modern CCD detectors to collect observational data in a digital form. Today, almost all the standard measures are digital and each measure can be generally considered as a collection of images forming a multi-dimensional data set. In many cases extensive image processing is required to obtain meaningful images. Besides observational data, the improvement of technology and the availability of super-computing multiprocessor system, have led to a dramatic increase of the volume of data coming from numerical simulations. With the evolution of high performance computers, numerical simulations have assumed a great role in the scientific investigation, allowing the user to run simulation with higher and higher resolution.

Modern astrophysical simulations produce gigabytes of data which have to be efficiently visualized and analyzed. In fact, a simulation produces several outputs to be correlated and analyzed, corresponding to different temporal tags and associated computational meshes (sometimes irregular or structured grids). Data produced in these simulations are often multi-dimensional arrays with several physical quantities. These data are very hard to manage and to analyze efficiently. Visualization means different things to different people, but anyway visualization is used to well understand data, to make some preliminary analysis, to extract information for further analysis and to show results in an high publication quality: “One nice figure is worth one hundred words”.

7.1 How does visualization adapt numbers to humans?

“It is nice to know that the computer understands the problem. But I would like to understand it too” Eugene Wigner (Physicist, 1902-1995)

Using color, shape, interaction, spatial relationships and visual metaphors the visualization makes it easy to understand the relationships and characteristics of our data. Structures simplify perception; for example through a pre-attentive perception you can capture the regularity in the data. Trivial example: how many “3” are there in this numerical sequence?

1315484161844323891345658340425

and this other?

1315484161844323891345658340425

Highlighting, in some way, the elements with a special feature we make immediately perceive trends, correlations, structures, outliers, etc. A classical example to recall is the Kepler’s third law. If the orbital period P of a planet is plotted on log-log graph paper against its distance R from the Sun (taken as equal to the semi-major axis of the path ellipse), then the result is a straight line, the mathematical statement of Kepler’s third law.

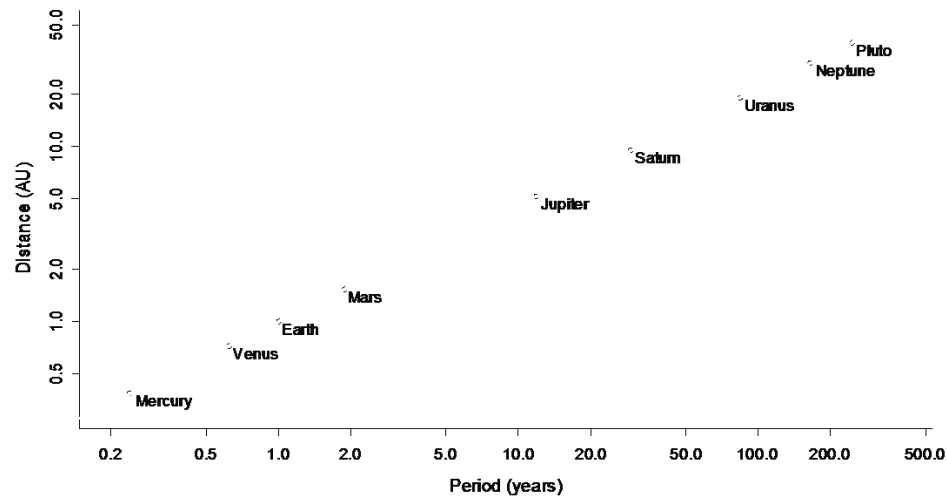


Figure 7.1: Kepler law: correctly display the data allows us to easily discover data relationships.

7.1.1 Visualization can reveal anomalies in data

The data display can be useful for exploring data, to grasp the general features, to easily understand the temporal evolution of the quantities involved, but also to be able to perceive with ease anomalies, non-physical discontinuities in the model adopted or detect simple errors in read/write files.

We used simple 3D visualization programs to explore the data, to verify the correctness of the temporal evolution, since the first computational step. With this method we were able to detect and correct errors in EVOL.

For example, in our simulations, all particles of dark matter should have, by construction, the same mass for the duration of the simulation, but in a particular case, due to a small error in the MPI communications, the masses changed. Viewing the entire system, and assigning the color of the particles as a function of mass, rotating and zooming the system, the problem is evident in a few seconds.

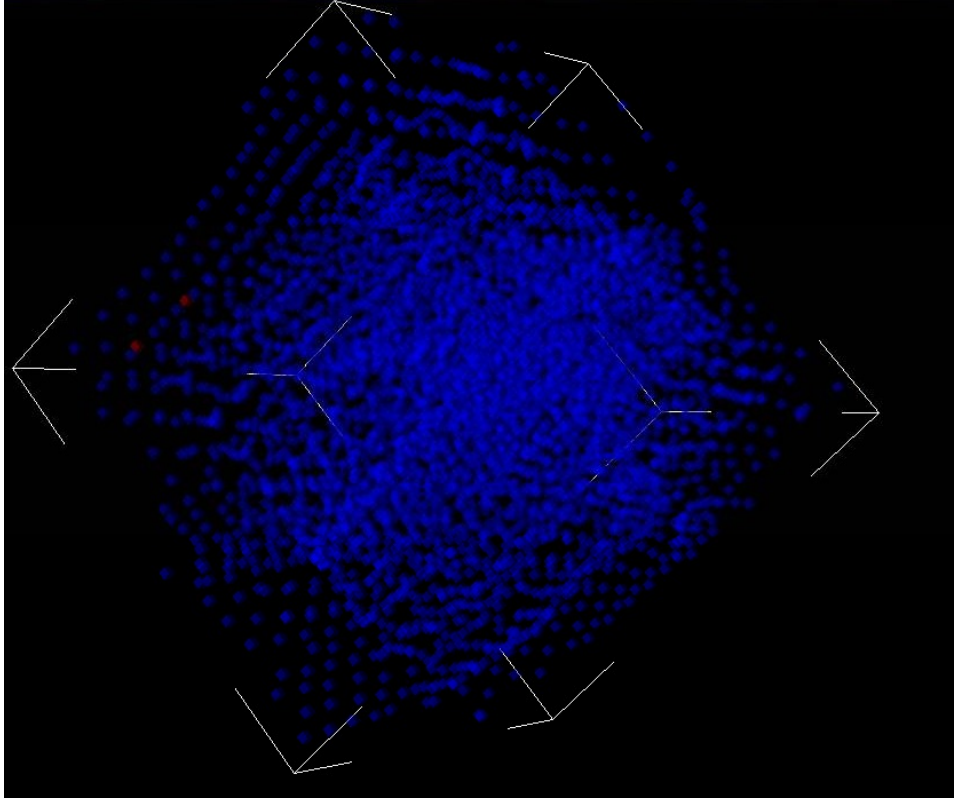


Figure 7.2: Graphically displaying data helps finding out troubles. A cosmological box from a wrong simulation, with a problem with the masses of particles. VisIVO help us to clearly see the two particles with wrong mass (red).

With the same approach have been discovered and corrected anomalies in the calculation of the density of gas particles, errors in the division of the computational domain between processors, and spatial irregularities in star formation. In the sections below, I will shortly describe the visualization software we have used.

7.2 AstroMD and VisIVO

We searched and tried many solutions, using codes specifically written for astronomical use, and software written for general purposes. Our search was mainly driven by practical requirements: we desired a freeware software that did not need too much memory or a powerful graphics processor, so it could be easily used on any PC (even the old ones) at our disposal.

AstroMD. This is an old software developed and distributed by CINECA (Bologna, Italy) for visualization and analysis of astrophysical data (Becciani et al. 2003). The software is able to handle various file formats, including ASCII, which makes it very flexible at zero cost.

The software relies on VTK package for three-dimensional graphics, and uses the Tcl/tk user interface. It can be used in different operating systems, provided the necessary libraries are present. I have successfully used it on both Linux and Windows. There are several possibilities of visualization and analysis, from spatially displaying a set of points, to views of volumes, of isodensity surfaces for arbitrary variables and to the animation of the time evolution of a system. This latter possibility is particularly interesting, because one can easily see how the problem under investigation evolves with time (large scale structures and galaxies in our case). The procedure is as follows. First the files are read in sequence and a special list is built which AstroMD interpret as the temporal sequence of files. Better performance is achieved using binary output files. Second, a software must be written to interface EVOL and AstroMD, selecting only the variables of interest, in order to save memory usage. Finally, a program capturing what is displayed on the monitor, compressing it with a proper video codec, must be adopted. Although this may look like a mere aesthetical exercise, using different colors for different ranges of the variables displayed, one can follow the spatial and temporal evolution of different quantities, such as temperature, density, chemical elements etc, thus providing a useful tool to analyse the global properties of the system under examination.

AstroMD can manage different physical quantities. It can find structures having a not well defined shape or symmetries, and perform quantitative calculations on a selected region or structure. It uses virtual reality techniques giving the sensation of travelling through a computer-based multi-dimensional model. This techniques are particularly effective for understanding the three dimensional structure of a system, its geometry, topology and

specific patterns. The display of data gives the illusion of a surrounding medium into which the user is immersed.

The main features can be summarized as follows:

Baryonic and dark matter have different physical properties and they have to be treated with different numerical approach. Dark matter is usually described by N-body algorithms, while simulation carried out with baryons have a fluid-dynamical description (either Eulerian or Lagrangian). Further components, like stars or different chemical species, can be introduced. These kinds of data require different types of analysis and different kind of visualization techniques. Dark matter needs the analysis of particle positions and velocity, while baryons require mesh based analysis and visualization. Furthermore particle associated quantities, like the mass density or the gravitational potential, require their calculation and visualization on a mesh.

To have a clear 3D representation and efficient and fast tools of navigation, zoom and rotation of a specific user-selected region are possible. A sub-box can be selected interactively inside the parent box with a different spatial resolution, so that the user can focus on the most interesting regions. Boxes can be translated, rotated, zoomed in and out with respect to selected positions. Colors and luminosities can be chosen freely by the user.

The opacity of the particles can be increased, so that low density regions can be more easily detectable, or decreased, so that the details of the high density regions substructures are better visualized. Different particles species (e.g. dark matter and baryons) can be visualized at the same time using different colours.

Images of different evolutionary stages can be combined in order to obtain a dynamic view of the behaviour of the system.

Finally, different time frames can be shown in a sequence. When particles are used, their positions are linearly interpolated between two available keyframes. This technique engenders a graphic animation in real time, giving the impression of a continuous movement of particles, generally a randomized sample of the global data set. Both the single images and the whole sequence of time-step can be saved in bitmap format. The program allows one to treat both particles and fields related data.

VisIVO The limitations of AstroMD, poor memory management, as well as the fact that Visual is under constant development, convinced me to try out this software. Visualization Interface for the Virtual Observatory (VisIVO) is a visualization package developed in collaboration between INAF and CINECA. VisIVO (Becciani et al. 2006, 2010) is similar to AstroMD, with the possibility of more sophisticated analysis; it is better implemented; also, it can communicate with different astronomical software using a specifically designed protocol. It is thus possible to begin the analysis with any other software compatible with this protocol, and then communicate the results in real time for instant viewing. The communication is bidirectional so it is possible to select a part of the system and communicate the selec-

tion to the other software. In addition, the ability to create vectors from existing data using a predefined set of operations (the four basic arithmetic ones, trigonometry, logarithms, etc.) easily allows for a real-time analysis of data. In this work VisIVO has been used to evaluate the spatial and velocity evolution of the system and for rapid graphical displaying of Star Formation histories, Supernovae explosion rates, and chemical evolution in galactic systems.

Also, it is of primary importance the possibility to quickly identify errors or oversights, or to evaluate the distribution of many parameters characterizing a simulation. VisIVO is able to simultaneously handle and display up to six variables: three arbitrary variables on the three spatial axes, and a variable showing the particles in different colors; plus, one or two more variables can be shown, choosing the way in which the particles are displayed (using a spheric or cubic vision a single variable can be displayed, but choosing a cylindric or conic vision - solid characterized by just two parameters, height and base radius - two variables become available). While we can not replace other more specific programs in the analysis of qualitative results, I consider VisIVO an excellent and practical aid for initial analysis.

I summarize his main features:

- Free software, cross platform (Windows and Linux)
- 3D -2D Visualization
- N-Dim (>3) Visualization of Astrophysical data
- Point, Volume and Vector Visualization
- Zoom, lookup table, data selection, etc.
- Sample using a box or ISO-Surface (extract cluster)
- Astrophysical Data Analysis and Mathematical Operation
 - Randomizer: extract a random subset of points
 - Power Spectrum of a distribution of points
 - Correlation Filter in a point distribution.
 - Points Splatter: distribute scalars on a regular mesh
- VO (Virtual Observatory) compliance
- Interoperability: VOT, PLASTIC¹, Vizier

¹PLASTIC is a protocol for communication between desktop astronomy applications. It is simple for application developers to adopt and is easily extended. Through PLASTIC applications can share data, link views or instruct each other to load an image of a particular area of sky

7.3 SPLASH

SPLASH (Price 2007) is a software utility for the visualisation of outputs from (astrophysical) simulations using the Smoothed Particle Hydrodynamics (SPH) method. It is written in Fortran 90/95 and utilises the PGPLOT library. SPLASH is designed for this specific task - to use SPH tools to analyse SPH data and to make this a straightforward task such that publishable images and animations can be obtained as efficiently as possible from the raw data with a minimum amount of effort from the user.

What it does

In particular the following features are included:

- Rendering of particle data to an array of pixels using the SPH kernel
- Cross-sections through 2D and 3D data (as both particle plots and rendered images).
- Fast projections through 3D data (i.e., column density plots, or integration of other quantities along the line of sight)
- Surface renderings of 3D data.
- Vector plots of the velocity (and other vector quantities), including vector plots in a cross section slice in 3D.
- Rotation and animation sequence generation for 3D data.
- Automatic stepping through timesteps, making animations simple to produce.
- Interactive mode for detailed examination of timestep data (e.g. zooming, rotating, plotting particle labels, working out the gradient of a line, stepping forwards/backwards through timesteps).
- Multiple plots on page, including option to automatically tile plots if y and x limits are the same.
- Plot limits can be fixed, adaptive or particle tracking. Also simple to change axes to log, invert, square root or absolute of a quantity.
- Exact solutions for common SPH test problems (e.g. hydrodynamic shock tubes, polytropes).
- Calculation of quantities not dumped (e.g. pressure, entropy)
- Conversion of binary dump files to ascii format.
- Transformation to different coordinate systems (for both coordinates and vector components) and rescaling of data into physical units.

- Straightforward production of in different graphical format

I will not dwell further on the various features of this software, which is very useful to address all the problems of visualization of cosmological simulations. The software is under continuous development, the source is available and easily adaptable to your needs. The simplicity and the quality of the display produced make this software widely used not only in the preliminary stages but also in the production of high-quality graphics can be incorporated into published articles.

Chapter 8

N-body Tree-SPH code: EvoL

8.1 N-body technique

The basic idea of an N-body code is simple. A system of discrete point masses can be represented by a set of “bodies” or “particles”, moving in time and space under the force field exerted by the other bodies, or by external sources. Moreover, a fluid can be sampled by particles representing volume elements. The gravitational force exerted on a fixed test particle by the system of bodies can be obtained by the vectorial summation of all the particles’ contributions. This is simply the straightforward application of Newton’s universal gravitation law. Alongside with the force calculation, the gravitational potential energy of each particle can be obtained in the same way (S. Aarseth and A. Toomre pioneered this technique in the first 70s.)

EvoL (Merlin et al. 2010) is the new release of the Padova N-body code (PD-TSPH Carraro et al. 1998). It is a flexible, fully lagrangian, parallel, and self-adaptive N-body code, written in Fortran95, specifically designed for simulations of cosmological structure formation on cluster, galactic and sub-galactic scales. EvoL is based on the classical Oct-tree by Barnes and Hut (1986) and on the Smoothed Particle Hydrodynamics algorithm (SPH, (Lucy 1977)). It includes special features such as adaptive softening lengths with correcting extra-terms, and a modern formulation of the SPH technique. It is designed to be run in parallel on multiple CPUs to optimize the performance and save computational time.

The core of the Padova N-body Tree-SPH (Pd-Tsph) code was written during the 90s by C. Chiosi, G. Carraro, C. Lia and C. Dalla Vecchia (Carraro et al. (1998); Lia et al. (2002)). Over the years, many researchers added their contribution to the development of the code. In its original release, the Pd-Tsph was a Tree-SPH code, written in FORTRAN 90, conceptually

similar to the Tree-SPH by Hernquist and Katz (1989). Schematically, Pd-Tsph used a basic implementation of SPH to solve the equations of motion for the gas component, and the Barnes and Hut (1986) Tree algorithm to compute the gravitational interactions. The integration of the equation of motions could only be carried out in proper physical coordinates and void boundary conditions.

In the following Sections I will review the last release of EvOL. The code is described in details in Merlin et al. (2010) for what concerns its basic features and the reader is referred to that paper for an accurate description (we include the paper in the last part of this Thesis). In what follows I briefly summarize the main features of the code.

8.2 Description of the code: Gravity

Gravity is the leading force behind the formation of cosmic structures, on many scales of interest, from galaxy clusters down to individual stars and planets.

Classical gravitation is a well understood interaction. It is described by Newton’s law of universal gravitation:

$$\vec{F}_{ij} = G \frac{m_i m_j}{|\vec{r}_i - \vec{r}_j|^3} (\vec{r}_i - \vec{r}_j), \quad (8.1)$$

where G is the gravitational constant.

The gravitational force exerted on a given body (i.e., particle) by the whole system of bodies within a simulation can be obtained by the vectorial summation of all the particles’ contributions (without considering, for the moment, the action of the infinite region external to the computational volume; this can indeed be taken into account using periodic boundary conditions, see Sec. 8.7). This is simply the straightforward application of Eq. 8.1. Anyway, in practice this approach is inefficient and may lead to artificial divergences as explained in the following Sections.

A convenient approach are the so-called *tree* structures, in which particles are arranged in a hierarchy of groups or “cells”. In this way, when the force on a particle is computed the contribution by distant groups of bodies can be approximated by their lowest multipole moments. In this way, the computational cost for a complete force evaluation can be reduced to order $O(N \log N)$. In the classical Barnes and Hut (1986) scheme, the computational spatial domain is hierarchically partitioned into a sequence of cubes, where each cube contains eight siblings, each with half the side-length of the parent cube. These cubes form the nodes of an oct-tree structure. The tree is constructed such that each node (cube) contains either a desired number of particles (usually one, or one per particles type - DM, gas, stars), or is progenitor to further nodes, in which case the node carries the monopole

and quadrupole moments of all the particles that lie inside its cube. A force computation then proceeds by walking the tree, and summing up appropriate force contributions from tree nodes. In the standard tree walk, the global gravitational force exerted by a node of linear size l on particle i is considered only if $r > l/\theta$, where r is the distance of the node from the active particle and θ is an *accuracy parameter* (usually ≤ 1): if a node fulfills the criterion, the tree walk along this branch can be terminated, otherwise it is “opened”, and the walk is continued with all its siblings. The contribution from individual particles is thus considered only when they are sufficiently close to the particle i . Further refinement is possible (see Merlin et al. 2010).

8.2.1 Softening of the gravitational force

Close encounters between particles can cause numerical errors, essentially because of the limits in time and mass resolution. Moreover, dense clumps of particles steal large amounts of computational time. To cope with this, it is common practice to *soften* the gravitational force between close pairs of bodies: if the distance between two particles becomes smaller than a suitable *softening length* ϵ , the force exerted on each body is corrected and progressively reduced to zero with decreasing distance.

Different forms of the softened force function can be used. A possible expression is given by the following formula:

$$\frac{F(\vec{r}, \epsilon)}{Gm_i} = m_j \frac{\vec{r}}{|\vec{r}|} \times \quad (8.2)$$

$$\begin{cases} 1/\epsilon^2 [\frac{4}{3}u - \frac{6}{5}u^3 + \frac{1}{2}u^4] & \text{if } 0 \leq u < 1, \\ 1/\epsilon^2 [\frac{8}{3}u - 3u^2 + \frac{6}{5}u^3 - \frac{1}{6}u^4 - \frac{1}{15u^2}] & \text{if } 1 \leq u < 2, \\ 1/|\vec{r}|^2 & \text{if } u \geq 2, \end{cases}$$

where $\vec{r}$ is the distance between particles i and j , and $u = |\vec{r}|/\epsilon$. This expression for the force softening corresponds (via Poisson equation) to a density distribution *kernel* function proposed by Monaghan and Lattanzio (1985), which is widely adopted in Smoothed Particles Hydrodynamics algorithms (see Sec. 8.3)

$$W(r, \epsilon) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}u^2 + \frac{3}{4}u^3 & \text{if } 0 \leq u < 1 \\ \frac{1}{4}(2 - u)^3 & \text{if } 1 \leq u < 2 \\ 0 & \text{if } u \geq 2. \end{cases} \quad (8.3)$$

Note that in Eq. 8.3 the softening length ϵ is assumed to be the same for the two particles i and j . In the more general situation in which each particle carries its own ϵ , a symmetrization is needed to ensure energy and momentum conservation: this can be achieved either using $\epsilon = \bar{\epsilon}_{ij} = (\epsilon_i +$

$\epsilon_j)/2$, or by symmetrizing the softened force after having computed it with the two different values ϵ_i and ϵ_j .

The softening lengths can be fixed in space and time, or may be let vary with local conditions. If the softening length is kept constant, the choice of its numerical value is of primary importance, for too small a softening length will result in noisy force estimates, while too large a value will systematically bias the force in an unphysical manner (Merritt 1996; Romeo 1998; Athanassoula et al. 2000; Price and Monaghan 2007).

Obviously, the accuracy can be greatly improved if ϵ is let vary according to the local particle number density (see e.g. Dehnen 2001). Moreover, in principle, if particles in a simulations are used to sample a continuous fluid (whose physics is determined by the Navier-Stokes or the Boltzmann equations) the properties of such points should always change accordingly to the local properties of the fluid they are sampling, to optimize the self-adaptive nature of lagrangian methods (on the other hand, if particles represent discrete objects (single stars, or galaxies in a cluster, etc.), their softening lengths might perhaps be considered an intrinsic property of such objects and may be kept constant, depending on their mass and not on the local density of particles).

In cosmological and galaxy-sized simulations, gas and Dark Matter particles are point-masses sampling the underlying density field, and stellar particles represent clusters of thousands of stars prone to the gravitational action of the nearby distribution of matter; thus a fully adaptive approach seems to be adequate to describe the evolution of these fluids. However, it can be easily shown that simply letting ϵ change freely would result in poor conservation of global energy and momentum, even if in some cases the errors could be small (see e.g. Price and Monaghan 2007; Wetzstein et al. 2009).

To cope with this, *EvOL* allows for the adaptive evolution of individual softening lengths, but includes in the equation of motions of particles self-consistent correcting additional terms. Such terms can be derived analitically (Price and Monaghan 2007; Merlin et al. 2010). Again, see Merlin et al. (2010) for further details.

8.3 Description of the code: Hydrodynamics: The Baryon Physics

Astrophysical gaseous plasmas can generally be approximated to highly compressible, unviscous fluids, in which anyway a fundamental role is played by localized violent phenomena such as strong shocks, high energy point-like explosions and/or supersonic turbulent motions.

EvOL follows the basic gas physics via Smoothed Particles Hydrodynamics (SPH, Lucy 1977; Monaghan 1992), in a modern formulation based

on the review by Rosswog (2009), to which the reader is referred for the details. Anyway, some different features are present in our implementation and we summarize them in the following, along with a short overview of the whole SPH algorithm.

8.4 Smoothed Particles Hydrodynamics

The basic idea of SPH is to model hydrodynamics of fluids through a discretization of the continuous fields which describe the properties of the media, following the equations of motion and conservation for a finite (possibly large) number of particles sampling the fluid in a lagrangian point of view. In the SPH approach the properties of single particles are *smoothed* in real space through the kernel function W , and thus *weighted* by the contributions of neighbouring particles. In this way, the physical properties of each point in real space can be obtained by the summation over particles of their individual, discrete properties.

The integral interpolating in space a function $A(\vec{r})$ can be written as

$$A_i(\vec{r}) = \int A(\vec{r}') W(\vec{r} - \vec{r}', h) d\vec{r}', \quad (8.4)$$

(the interpolating kernel function W can be the density kernel Eq. 8.3).

The relative discrete approximation is

$$A_s(\vec{r}_i) = \sum_j \frac{A_j}{\rho_j} m_j W(\vec{r}_i - \vec{r}_j, h_i), \quad (8.5)$$

and the physical density of any SPH particle can be computed as

$$\rho_i = \sum_j m_j W(|\vec{r}|, h_i) \quad (8.6)$$

(here and throughout this Section, it is implied that all summations and computations are extended to SPH particles only).

The divergence of velocity is usually estimated using (e.g., Monaghan 1992)

$$\nabla \cdot \vec{v}_i = \sum_j m_j [(\vec{v}_j - \vec{v}_i) \cdot \nabla_i W_{ij}] / \rho_i, \quad (8.7)$$

where $\nabla_i W_{ij}$ denotes the gradient of $W(\vec{r} - \vec{r}', h)$ with respect to the coordinates of particle i .

The dynamical evolution of a continuous fluid is governed by the well known laws of conservation: the continuity equation which ensures conservation of mass, the Euler equation which represents the conservation of momentum, and the equation of energy conservation (plus a suitable equation of state to close the system). These equations must be written in discrete form to be used within the SPH formalism. In modern versions of SPH, this is usually obtained through a Lagrangian analysis.

The continuity equation is usually replaced by the interpolater Eq. 8.6.

The equation of motion and conservation of energy read, in the most general case (Merlin et al. 2010):

$$\begin{aligned} \frac{d\vec{v}_{i,hyd}}{dt} = & \sum_j m_j \times \\ & \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) \nabla_i W_{ij}(h_i) + \right. \\ & \left. \frac{P_j}{\rho_j^2} \left(1 + \frac{\zeta_j/m_i}{\Omega_j^*} \right) \nabla_i W_{ij}(h_j) \right], \end{aligned} \quad (8.8)$$

and

$$\begin{aligned} \frac{du_i}{dt} = & \\ & \sum_j m_j \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) (\vec{v}_j - \vec{v}_i) \cdot \nabla_i W_{ij}(h_i) \right]. \end{aligned} \quad (8.9)$$

In these expressions, the correcting terms are as follows:

$$\Omega_i^* = 1 - \frac{\partial h_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(h_i)}{\partial h_i}, \quad (8.10)$$

$$\Upsilon_i = \left[1 - \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(\epsilon_i)}{\partial \epsilon_i} \right], \quad (8.11)$$

$$\xi_i = \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial \phi_{ij}(\epsilon_i)}{\partial \epsilon_i}, \quad (8.12)$$

and

$$\zeta_i = \frac{\partial h_i}{\partial n_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i}. \quad (8.13)$$

The smoothing kernel W is given by Eq. 8.3. The standard equation of state is that of an ideal gas,

$$P = (\gamma - 1)\rho u, \quad (8.14)$$

where γ is the adiabatic index (5/3 for a monatomic gas); a more general equation of state will be presented in Merlin et al. (2012b, in preparation).

The smoothing length h may in principle be a constant parameter, but the efficiency and accuracy of the SPH method is greatly improved adapting the resolution lengths to the local density of particles. A self-consistent method is to obtaining h from

$$\frac{dh}{dt} = \frac{dh}{dn_{SPH}} \frac{dn_{SPH}}{dt}, \quad (8.15)$$

where n_{SPH} is the local number density *of SPH particles only*, and relating h to n_{SPH} by requiring that a fixed number of kernel-weighted particles is contained within a smoothing sphere, i.e.

$$h_i = \eta \left(\frac{1}{n_{SPH,i}} \right)^{1/3}. \quad (8.16)$$

Note that in this case, while still obtaining the mass density via summation Eq. 8.6, one can alongside compute the number density of particles at particle i 's location, i.e.:

$$n_{SPH,i} = \sum_{SPH,j} W(|\vec{r}_{ij}|, h_i), \quad (8.17)$$

It is worth recalling here that in the first versions of SPH with spatially-varying resolution the spatial variation of the smoothing length was generally not considered in the formulation of the equations of motion, and this resulted in secular errors in the conservation of entropy. The inclusion of extra-correcting terms (the so-called ∇h terms) is therefore important to ensure the conservation of *both* energy and entropy (see e.g. Serna et al. 1996; Alimi et al. 2003).

These ∇h terms were first introduced explicitly (Nelson and Papaloizou 1994), with a double-summation term added to the canonical SPH equations. Later, an implicit formulation was obtained (Monaghan 2002; Springel and Hernquist 2002), beginning with the Lagrangian derivation of the equations of motion and self-consistently obtaining correcting terms which accounts for the variation of h .

8.4.1 Smoothing and softening lengths for SPH particles

SPH particles under self-gravity have two parameters characterizing their physical properties: the softening length ϵ , related to their gravitational interactions with nearby particles, and the smoothing length h , used to smooth their hydrodynamical features.

In principle there is no need for these two quantities to be related to one another, because they refer to two separate fields of action, and have somehow opposite functions as noted above. However, Bate and Burkert (1997) claimed that a quasi-stable clump of gas could be un-physically stabilized against collapse if $\epsilon > h$, because in this case pressure forces would dominate over the strongly softened gravity, or on the other hand, if $\epsilon < h$, collapse on scales smaller than the formal resolution of SPH may be induced, causing artificial fragmentation. They recommend that gravitational and hydrodynamical forces should be resolved at the same scale length, imposing $\epsilon = h$ (requiring both of them to fixed, or ϵ to be adaptive like h , with the introduction of the correcting terms described above). Anyway, Williams et al. (2004) have studied the effects of such procedure, founding that in many hydrodynamical problems this can cause a number of problems.

They pointed out that, in contrast with the claims by Bate and Burkert (1997), the smoothing length should not be kept equal to the softening length, because in many physical situations (for example, shocks) hydrodynamic forces dominates over gravity, and may need to be properly resolved on size scales smaller than the softening length.

Finally, a numerical problem arises when trying to keep $\epsilon = h$ in cosmological simulations, since the presence of a Dark Matter component makes it impossible to keep constant the number of both hydrodynamic *and* gravitational neighbours.

Thus, in EVOL both ϵ and h can be let free to vary independently from one another.

8.4.2 Discontinuities

By construction, SPH works smoothing out local properties on a spatial scale of the order of the smoothing length h . Anyway, strong shocks are of primary importance in a number of astrophysical systems. Thus, to model a real discontinuity in the density and/or pressure field *ad hoc* dissipative, pressure-like terms must be added to the perfect fluid equations, to spread the discontinuities over the numerically resolvable length.

This is usually achieved with the introduction of an *artificial viscosity*, a numerical artifact that is not meant to mimic physical viscosity, but simply to reproduce on large scale the action of smaller, unresolved scale physics.

Among the many formulations that have been proposed, the most commonly used, with a suitable expression for such term as reported in Merlin

et al. (2010).

As Price (2008) pointed out, a similar approach should be used to smooth out discontinuities in *all* physical variables. In particular, an *artificial thermal conductivity* is necessary to resolve discontinuities in thermal energy, although very few published formulations of SPH account for that. A term P_i^u should therefore be added to the equation of energy conservation. See Merlin et al. (2010) for the complete discussion on the issue.

As in the case of artificial viscosity, it is worth noting that this conductive term is not intended to reproduce a physical dissipation; instead, it is a merely numerical artifact introduced to smooth out unresolvable discontinuities.

8.5 Time Integration

8.5.1 Leapfrog integrator

Particles positions and velocities are advanced by means of a standard leapfrog integrator, in the so-called Kick-Drift-Kick (KDK) version, where the K operator evolves velocities and the D operator evolves positions.

Predicted values of physical quantities at the end of each time-step are extrapolated at the beginning the step, to guarantee synchronization in the calculation of accelerations and in the outputs. Moreover, if the individual time-stepping option is adopted, non-active particles use predicted quantities to give their own contributions to forces and interactions.

8.5.2 Time-stepping

A clever midpoint between accuracy and efficiency in the choice of the time-step size is a fundamental issue. One would like to obtain robust results in a reasonable amount of CPU time. To do so, numerical stability must be ensured, together with a proper description of all relevant physical phenomena, at the same time reducing the computational cost keeping the size of the time-step as large as possible.

To this aim, at the end of each step each particle computes its own optimal time-step as the minimum between the following:

$$\begin{aligned}\delta t_{acc,i} &= \eta_{acc} \sqrt{\frac{\min(\epsilon_i, h_i)}{|\vec{a}_i|}} \\ \delta t_{vel,i} &= \eta_{vel} \sqrt{\frac{\min(\epsilon_i, h_i)}{|\vec{v}_i|}} \\ \delta t_{av,i} &= \eta_{av} \sqrt{\frac{|\vec{v}_i|}{|\vec{a}_i|}}\end{aligned}$$

with obvious meaning of symbols; the parameters η are of order 0.1. SPH particles also compute two more values, the first obtained through the well known *Courant condition* and the other constructed to avoid large jumps in thermal energy:

$$\delta t_{C,i} = \eta_C \frac{h_i}{h_i |\nabla \cdot \vec{v}_i| + c_i + 1.2(\alpha c_i + \beta \max_j |\mu_{ij}|)}$$

$$\delta t_{u,i} = \eta_u \left| \frac{u_i}{du_i/dt} \right|$$

where α, β and μ are the quantities defined in the artificial viscosity parametrization, see See Merlin et al. (2010), and c is the sound speed.

If the simulation is run with a standard *global* time-stepping scheme, all particles are advanced adopting the minimum of all individual time-steps calculated in this way; synchronization is clearly guaranteed by construction in this case.

If desired, a minimum time-step can be imposed. In this case, however, limiters must be adopted to avoid numerical divergences in accelerations and/or internal energies. This of course leads to a poorer description of the physical evolution of the system; anyway, in some cases a threshold may be necessary to avoid very short time-steps due to violent high energy phenomena.

8.6 Comoving coordinates

The total acceleration acting on a particle consists of two terms, $acc_{tot} = acc_{grav} + acc_{hyd}$. The term acc_{hyd} refers to the sum of all the non gravitational forces, and acts only on SPH particles. In general, acc_{hyd} shows a strong dependence on local physical parameters like temperature and (proper) density, rather than on large scale or global cosmic motions. Thus, this term is generally better evaluated working in proper physical coordinates.

On the other hand, the gravitational term acc_{grav} can be easily computed either in proper or in comoving coordinates. Previous release of the PD-TSPH code only allowed for the straightforward computation in proper coordinates. However, to properly simulate large portions of the Universe accounting for large scale motions within a consistent cosmological framework, it is more appropriate to integrate the equations of motion using comoving coordinates, possibly adopting periodic boundary conditions.

Only the gravitational part of the equations of motion can be optionally integrated in comoving frame in EVOL, leaving hydrodynamical accelerations and velocities in proper physical coordinates.

The comoving coordinates $\vec{x}$ are related to the physical coordinates $\vec{r}$ by

the cosmological expansion factor a , i.e.

$$\vec{x} = \vec{r}/a. \quad (8.18)$$

Deriving this relation one finds the comoving velocity as

$$\vec{w} = \dot{\vec{x}} = \frac{a\dot{\vec{r}} - \dot{a}\vec{r}}{a^2} = \frac{1}{a}\vec{u} - H_z\vec{x}, \quad (8.19)$$

where $H_z = H(z)$ is the Hubble constant at redshift $z = 1/a - 1$. For a flat Λ -CDM Universe, this is given by $H(z) = H_0\sqrt{[\Omega_M(1+z)^3 + \Omega_\Lambda]}$. In Eq. 8.19, the first term is the peculiar comoving velocity, while the second term accounts for the expansion of the Universe. The derivation of the comoving equations of motion is non trivial (for a detailed treatment the reader should refer to Peebles 1980, Chapters 7 and 8).

The Newtonian Euler equation for a non-collisional fluid in an expanding Universe, written in proper physical coordinates, is

$$\frac{\partial \vec{u}}{\partial t} \Big|_{\vec{r}} + (\vec{u} \cdot \nabla_{\vec{r}})\vec{u} = -\nabla_{\vec{r}}\Phi, \quad (8.20)$$

where $\vec{u} = \vec{v} + H_z\vec{r}$ is the proper total velocity, given by the proper peculiar velocity (due to mutual gravitational attraction) plus the Hubble flow velocity (compare this expression with Eq. 8.19), and Φ is the gravitational potential given by the Poisson equation,

$$\nabla_{\vec{r}}^2\Phi = 4\pi G\rho. \quad (8.21)$$

When the equations of motion are integrated in comoving frame, *EvOL* uses the cosmic time t , the comoving positions $\vec{x}$ and the comoving velocities $\vec{w}$ as phase-space variables. At each time-step, SPH particles add the hydrodynamical acceleration transforming comoving to proper velocity via Eq. 8.19, summing the $acc_{hyd} \times dt$ term, and going back to comoving variables. $H_z, a, acc_{hyd}, acc_{grav}$ are evaluated at the beginning of each time-step and are considered constant within it (see Quinn et al. 1997, for a discussion about this approximation).

8.7 Periodic boundary conditions

Periodic boundary conditions are needed to correctly account for large scale tidal forces, and more in general to avoid spurious border effects. In *EvOL*, they are implemented using the well known Ewald (1921) method, adding to the acceleration of each particle an extra-term due to the replicas of particles and nodes, expanding in Fourier series the potential and the density fluctuation, as described in Hernquist et al. (1991) and in Springel et al. (2001). If $\vec{x}_i$ is the coordinate at the point of force-evaluation relative to a

particle or a node j of mass m_j , the *additional* acceleration that must be added to account for the action of the infinite replicas of j results

$$acc_{per}(\vec{x}) = m_j \left[\frac{\vec{x}}{|\vec{x}|^3} - \sum_n \frac{\vec{x} - \vec{n}L}{|\vec{x} - \vec{n}L|^3} \times \right. \quad (8.22)$$

$$\left(erf c(\omega|\vec{x} - \vec{n}L|) + \frac{2\omega|\vec{x} - \vec{n}L|}{\sqrt{\pi}} \times e^{-\omega^2|\vec{x} - \vec{n}L|^2} \right)$$

$$\left. - \frac{2}{L^2} \sum_{\vec{h} \neq 0} \frac{\vec{h}}{|\vec{h}|^2} e^{-\frac{\pi^2|\vec{h}|^2}{\omega^2 L^2}} \sin \left(\frac{2\pi}{L} \vec{h} \cdot \vec{x} \right) \right]$$

where $\vec{n}$ and $\vec{h}$ are integer triplets, L is the periodic box size, and ω is an arbitrary number. Convergence is achieved for $\omega = 2/L$, $|\vec{n}| < 5$ and $|\vec{h}| < 5$ (Springel et al. 2001). The correcting acc_{per}/m_j terms are tabulated, and tri-linear interpolation off the grid is used to compute the correct accelerations. It must be pointed out that the adoption of this interpolation significantly slows down the tree algorithm, almost doubling the CPU worktime.

8.8 Cooling and chemical composition

8.8.1 Radiative cooling

We adopt the cooling functions described in Carraro et al. (1998), which are based on those elaborated by Chiosi et al. (1998). In brief, for temperatures greater than 10^4 K they lean on the Sutherland and Dopita (1993) tabulations for a plasma under equilibrium conditions and metal abundances $\log[Z/Z_\odot] = -10$ (no metals), -3, -2, -1.5, -1, -0.5, 0 (solar) and 0.5. The chemical properties of gas particles are followed in details, tracking the abundances of Fe, O, Mg, Si and C, and the global metallicity Z .

For temperatures in the range $100 < T < 10^4$ the dominant source of cooling is the H_2 molecule becoming rotationally and/or vibrationally excited through a collision with an H atom or another H_2 molecule and decaying through radiative emission; the data in use have been derived from the analytical expressions of Hollenbach and McKee (1979) and Tegmark et al. (1997).

For temperatures lower than 100 K, Sutherland and Dopita (1993) incorporate the results of Hollenbach and McKee (1979) and Hollenbach (1988) for CO molecule as the dominant coolant. The mean fractionary abundance of CO is given as a function of $[Fe/H]$ (see Carraro et al. 1998, for further details).

Also included are the contribution by inverse Compton cooling

$$\Lambda_{Compton} = 5.41 \times 10^{-36} x_e [T - T_{CMB}(1+z)](1+z)^4 \quad (8.23)$$

in erg/s/cm^3 , with the usual meaning of symbols (see e.g. Ikeuchi and Ostriker 1986).

The radiative cooling is computed separately from the SPH equation of energy conservation, due to its very short time-scales. During a dynamical time-step, the density and the metallicity of gas particles are kept fixed, as well as the mechanical heating rate. On the contrary, the temperature is let vary under the action of cooling, and the new cooling rates are simultaneously obtained as a function of the new temperature. The process is iterated until the end of the time-step.

8.9 Cooling and chemical composition: ROBO

In the range between 10^2 and 10^4 K, the Carraro et al. (1998) function assumes that the bulk of the gas is entirely molecular, which is obviously not always true, leading to a possible overestimation of the cooling rate (see e.g. Smith et al. 2008, their Fig. 8). In EVOL, the cooling function in this temperature range can be replaced with a new function, obtained from Galli and Palla (1998) calculations of the cooling rate due to collisions between hydrogen atoms and molecules, which depends on the effective abundance of molecules and atoms.

8.9.1 Robo: the ISM model and companion code

ROBO is a numerical code specifically designed to study the evolution of the ISM. It includes several atomic and molecular species linked together by a large network of reactions. ROBO is planned to be used as an ancillary tool for EVOL to calculate the thermal and chemical properties of the gas particles.

To this purpose, we follow the temporal variation of molecules like H_2 , HD and metals like C, O, Si, Fe and their ions. For all the details the reader should refer to Grassi et al. (2010), which we include in the final part of this Thesis.

The model deals with an ideal ISM element of unit volume, containing gas and dust in arbitrary initial proportions, whose initial physical conditions are specified by a set of parameters, which is let evolve for a given time interval. The history leading the element to that particular initial physical state is not of interest here. The ISM element is mechanically isolated from the host environment, i.e. it does not expand or contract under the action of large scale forces, however it can be interested by the passage of shock

waves originated by physical phenomena taking place elsewhere (e.g. supernova explosions). Furthermore, it does not acquire nor lose material, so that the conservation of the total mass applies, even if its chemical composition can change with time. It is immersed in a bath of UV radiation generated either by nearby or internal stellar sources and in a field of cosmic ray radiation. It can generate its own radiation field by internal processes and so it has its own temperature, density and pressure, each other related by an Equation of State (EoS). If observed from outside, it would radiate with a certain spectral energy distribution. For the aims of this study, we do not need to know the whole spectral energy distribution of the radiation field pervading the element, but only the UV component of it. Given these hypotheses and the initial conditions, the ISM element evolves toward another physical state under the action of the internal network of chemical reactions changing the relative abundances of the elemental species and molecules, the internal heating and cooling processes, the UV radiation field, the field of cosmic rays, and the passage of shock waves.

The following species are tracked by ROBO: H, H^+ , H^- , H_2 , D, D^+ , D^- , D_2 , HD, HD^+ , He, He^+ , He^{++} , C, C^+ , O, O^+ , Si, Si^+ , Fe, Fe^+ , and e^- . The list of reactions and their rates that are included in ROBO can be found in Grassi et al. (2010).

ROBO also tracks the evolution of dust, in turn made by several components, of which it follows the formation and destruction.

The evolution of the dust type (both in abundance and dimensions) is followed for silicates and carbonaceous grains. Dust is also destroyed by thermal sputtering and shocks.

Several cooling processes are considered in ROBO. In the high temperatures regime ($T \geq 10^4$ K) the metallicity dependent cooling rates of Sutherland and Dopita (1993) are adopted. For temperatures lower than 10^4 K we consider the following processes: (i) cooling by molecular hydrogen according to the formulation by Galli and Palla (1998), however, supplemented by the results of Glover and Jappsen (2007) who take in account the $H_2 - H$ interaction and the collisions with He, H^+ , H_2 and free electrons; (ii) cooling by metals is modeled including C, Si, O, Fe, and their ions as in Maio et al. (2007), Glover and Jappsen (2007) and Hollenbach and McKee (1989); (iii) cooling by deuterated molecular hydrogen according to the model by Lipovka et al. (2005); (iv) cooling by the CO molecule and (v) finally, cooling by Cen (1992).

The total cooling rate is the sum of all the contributions

$$\Lambda_{\text{tot}} = \Lambda_{\text{SD}} + \Lambda_{\text{H}_2} + \Lambda_{\text{HD}} + \Lambda_{\text{CO}} + \Lambda_{\text{metals}} + \Lambda_{\text{CEN}},$$

where the various terms Λ s are all functions of temperature and density and Λ_{SD} is the cooling rate by Sutherland and Dopita (1993), Λ_{H_2} , Λ_{HD} and Λ_{CO} are the cooling rates of the indicated molecular species, and Λ_{metals} is

the cooling by metals (C, O, Si and Fe). Finally, Λ_{CEN} is the cooling by Cen (1992).

HEATING. For the photo-dissociation of the molecular hydrogen and the UV pumping, the H and He photo-ionization, the H_2 formation in the gas and dust phase and, finally, the ionization from cosmic rays, heating is described as in Glover and Jappsen (2007). For the heating due to the photoelectric effect on dust grains, the model proposed by Bakes and Tielens (1994) and Weingartner and Draine (2001) is used.

DATABASE OF MODELS. Varying the initial value of the parameters describing the ISM (such as the temperature, density, abundances of elemental species etc.), ROBO is used to create a large number of evolutionary models.

8.9.2 The Artificial Neural Networks

The ideal way of proceeding would be to insert ROBO into a code simulating the temporal evolution of large scale structures and to calculate the thermodynamical properties of the ISM component. In all practical cases, this would be unreasonably time consuming. To cope with it without losing accuracy in the physical description of the ISM we construct an ANN able to replace ROBO in all cases.

In brief, ANNs are non-linear tools for statistical data modeling, used to find complex relationships between input and output data or to discover patterns in complex datasets.

Demanding computational problems (e.g. multidimensional fits or classifications) can be easily solved using ANNs whereas other methods would require large computing resources. An ANN is based on a simple conceptual architecture inspired by the biological nervous systems. It looks like a network composed by *neurons* and linked together by *synapses* (numerical weights). This structure is modeled with a simple algorithm, designed to predict an *output state* starting from an original *input state*. To achieve this, the ANN must be *trained*, so that it can “learn” to predict the output state from a given set of initial configurations.

If the convergence to a solution exists, after a number of iterations, the predicted output value gets very close to the original one: *the ANN has learned*.

The use of ROBO/ANN can be selected by the user at compile time.

8.9.3 Star formation

Star formation is a deeply investigated field at least since 1930’s, so it cannot be defined a “young” field. Still, it is one of the most poorly understood issues in astrophysics, and as already stressed it can be considered the turning point in our present problems with understanding galaxy formation. The process of star formation involves an extremely complex interplay between

a number of physical mechanisms: magnetism, plasma physics, turbulence, non-linear growth of gravitational perturbations, energetic feedback, ignitions of nuclear reactions.

Stars are known to form within cold, dense clump of gas, but many details of the process are still mysterious. In particular, there are several questions far from being properly answered. The first one is *why star formation is so inefficient*. If, as long thought, the process leading star formation was the collapse of self-gravitational unstable gas, almost all gravitationally collapsing gas should form stars on very short time-scales, while this is not observed. Secondly, on the opposite side of the problem, what happens to the angular momentum and magnetic flux which are present in typical star-forming gas clouds? If these quantities were conserved during protostellar cloud collapse, centrifugal and magnetic forces would eventually overwhelm gravity and prevent stars from forming, so angular momentum and magnetic flux must be somehow efficiently removed or redistributed during star formation. Finally, what determines the typical properties of stars and stellar systems that form (initial mass function, binary formation, planetary systems)?

A number of well-established empirical features of local stellar systems are usually considered to be universal. Anyway, it may not be so. For example, the Kennicutt (1998) law is surely adequate to describe the local relation between the star formation process and the gas content, but it is not necessarily valid in the high-redshift Universe. The same holds for the Initial Mass Function which shapes the mass range of newly born stars: despite the fact that locally the classic Salpeter (1955) law still works fine, top-heavy IMFs in early Universe or extreme environments seem to be necessary.

Numerical simulations of star forming regions are beginning to cast light on these issues, but a complete understanding of how the story really goes seems still quite far away.

In EVOL star formation is modeled by means of a stochastic method, similar to that introduced by Churches et al. (2001) and Lia et al. (2002). First, only gas particles belonging to convergent flows (such that $\nabla \cdot v < 0$) and denser than a suitable threshold ρ_* are considered eligible to form stars.

We do not impose any restriction on the temperature; this choice is motivated by the fact that thermal instabilities can produce star forming sites even within high temperature gas.

Then, if a particle satisfies these criteria, it is assumed to form stars with a rate $d\rho_*/dt = \epsilon_{SF}\rho_g/t_{ff}$, where $t_{ff} \simeq 0.5/\sqrt{G\rho_{tot}}$ is the free-fall time and ρ_{tot} is the local total mass density (DM plus BM), and ϵ_{SF} the dimensionless efficiency of the star formation process. This means that a gas particle is expected to transform a fraction ϵ_{SF} of its mass into stars over its free-fall time scale. However, a stochastic description based on the star forming mechanism is best suited to avoid the creation of exceedingly large numbers of star particles. Thus, gas particles undergo a Monte Carlo selection to see

whether they will actually form stars - in this case being instantaneously transformed into collisionless star particles - or not (see also Lia et al. 2002; Merlin et al. 2012, in preparation); to do so, a random number $r \in [0, 1]$ is drawn and compared to the probability that the particle is actually forming stars. An obvious choice would be to use the probability $P = \epsilon_{SF}/t_{ff} \times \Delta t$, where Δt is the dynamical time-step. However, the exact evaluation of the probability the real probability must take into account that the dynamical time-steps are generally much shorter than the free-fall times of particles. This implies that within a free-fall time scale a number of star forming events and corresponding random draws are possible depending on Δt , and therefore so does the global probability for a single particle to be turned into stars. If the global probability over t_{ff} is P , the stochastic process of random selection must be corrected considering that the probability summed over the number of draws $n \simeq \Delta t/t_{ff}$ must be equal to the probability of a single draw within the whole free-fall time. A first step is to take as the probability within a single time-step the value $p \simeq P/n$. However, the conjuncted probability of a successful event over n draws is $p \times \sum_{i=1}^n (1 - p)^{i-1} < P$. The correcting multiplicative term is easily obtained as

$$f = \frac{P}{\frac{P}{n} \times \sum_{i=1}^n (1 - \frac{P}{n})^{i-1}}, \quad (8.24)$$

so the final probability during the time-step is $f \times p$. If $r \leq fp$, then the gas particle is instantaneously turned into a stellar particle.

8.9.4 Energy Feedback

Stars (the young and massive ones in particular) inject chemically enriched matter and energy (both kinetic and thermal) in the surrounding medium, via radiative flux, winds, and Supernovae explosions. See e.g. Dyson and Williams (1997) for a complete analysis on this subject.

When a cluster of stars is born, the ultraviolet radiation emitted by massive stars quickly ionizes the surrounding medium, which is still essentially composed of H_2 molecules and/or atomic H. A single O-B star ionizes the surrounding neutral and molecular within a radius of ~ 1 pc, raising its temperature to an almost constant value of $\sim 10^4$. The disruption of the molecules, and the subsequent ionization of the hydrogen atoms, causes a sudden jump in pressure, because the number density of particles increases by a factor of 2 – 4. Then, radiation interacts with the dense electron gas, heating it to a typical constant temperature of some 10^4 K, i.e. a factor of 100-1000 higher than the surroundings, and the pressure follows the same increase. Thus, a shock wave soon develops, because of the pressure difference of a factor 200-4000. It can be shown that the shock front will reach the ionization front in less than 10^4 years, at a typical distance of a few parsecs from the star. In the ideal situation, the ionized gas would expand

until its pressure is greater than the surrounding, unshocked gas pressure. The equilibrium would be given by

$$2n_f k T_i = n_0 k T_n, \quad (8.25)$$

where n_f is the shock front gas density, T_i is the ionized gas temperature, and T_n is the neutral gas temperature. For typical values, the final radius results to be $\sim 50 - 100$ pc, but the time-scale for the equilibrium is much longer ($\sim 10^8$ yrs) than the Main Sequence time life of an O-B star, so it will never be reached, because stellar winds and Supernovae explosions will dramatically change the whole history. Moreover, it results that few percents of the initial UV energy emitted by stars is turned into kinetic energy, because almost everything is soon radiated away.

O-B stars also lose mass (*winds*) at very high velocities, ~ 2000 km/s, at rates of $\sim 10^{-6} M_\odot/\text{yr}$. This would anyway result in an energy input ca. two orders of magnitude lower than the UV flux input, but wind energy is much more efficient kinetically ($\sim 20\%$ of the energy input is retained), so in the end the two effects are comparable. The hot, dense, highly supersonic gaseous mass ejected by stars shocks the surrounding (already ionized by the UV radiation) medium, acting as a piston on the almost quiet warm gas and generating a complex double-shock structure which ends up heating the gas within the external shock front to some 4×10^7 K.

Finally, Supernovae explosions can be described as huge, instantaneous and point-like releases of energy ($\sim 10^{50-51}$ erg) from massive ($M > 8 M_\odot$) stars. In a very *first stage*, which lasts no more than 10^3 yrs, the whole energy budget released by the explosion is absorbed by the gaseous ejecta around the star (about half of the initial stellar mass), which act as supersonic pistons moving at thousands of km/s. This results in a shock structure similar to the case of winds, and generates a so-called “bubble” of hot, expanding material. In a *second, energy-conserving stage*, the bubble expands adiabatically, because cooling is not efficient. After some 4×10^4 years, the gas is finally able to cool and a *third, momentum-conserving (or snowplow) stage* develops. In this stage, a tiny, dense, cool shell of gas forms right behind the shock front. Since now there is no more energy injection in the hot bubble, the gas can cool down arbitrarily and possibly recombine in neutral hydrogen, resulting in new dense, star-forming regions. Numerical models confirm this picture (Hosokawa and Inutsuka 2007).

From this analysis it can be inferred that although energy feedback phenomena from single stars happen at quite small scales, they are thought to affect the ISM at galactic scales, regulating the formation of a whole galaxy (one should keep in mind that the global effects are the results of the energy injections by many Single Stellar Populations). It is well established (see e.g. Pelupessy et al. 2004) that the energy injected cause large-scale (hundreds of parsecs in large galaxies) random bulk motions of the interstellar

gas at close to the sonic speed. Mashchenko et al. (2008) made an hydrodynamical simulation the formation of a dwarf galaxy in cosmological context, concluding that feedback is able to generate strong gravitational potential fluctuations, possibly reducing the discrepancy between the expected and the observed density profile of Dark Matter halo, and affecting the gaseous behaviour on large scales.

The role of feedback is itself controversial, as its effects on star formation are still quite debated. As explained, surely the release of energy in the ISM quenches the ongoing star formation process, raising the temperature of the gas near the new-born stars and causing the expansion of bubbles; but behind the expanding shock waves the gas can be dense enough to cool efficiently, possibly in a thermal instability regime, and new cold clouds are expected to form, eventually giving birth to a new generation of stars. Moreover, if a cold clump is dense enough it may survive the passage of the shock and simply be kicked away without being disrupted (Elmegreen 2007, estimated that the fraction of a molecular cloud disrupted by UV radiation from an OB cluster is only $\sim 10 - 20\%$ of the GMC mass). Thus, star formation can also be triggered by feedback, making the process extremely non-linear.

Ferrara (1998) explored the effects of PopIII feedback on primordial ISM in terms of H_2 formation behind shock waves in a typical high redshift primordial halo, and concluded that these first objects might have a net positive feedback on subsequent galactic formation.

A recent review of the state of the art on the effects of PopIII stars feedback on primordial proto-galaxies can be found in Ciardi (2008).

8.9.5 Energy feedback in EvoL

When a gas particle is turned into a star particle, it can be considered to represent a Single Stellar Population (SSP) made of many real stars, which starts re-fueling the ISM with heavy chemical elements and energy, mainly because of winds from young massive stars, and Supernova (SN) explosions.

We consider two regimes for the energy injection by winds. When the SSP is young, the main source of energy are young massive stars. Their winds have very high speeds. We assume a constant velocity $v_{YSO} = 1500$ km/s, and the kinetic energy of these winds is assumed to be thermalized and released within the surrounding medium with an efficiency of 20% (see e.g. Dyson and Williams 1997). In the late stages of the SSP evolution, slow velocity winds from smaller stars becomes predominant; in these cases we assume an outflow speed $v_{OSO} = 60$ km/s, and the same thermalization efficiency.

Each *real* SN explosion is expected to deposit some 10^{51} kinetic ergs in a very small region, and on a short time scale. However, most of this energy is soon radiated away, and only a few percent is subsequently thermalized. We

use the results by Thornton et al. (1998) to obtain an analytic approximation of the fraction of the initial energy which becomes available at the end of the expanding phase of the SN bubble. However, in the real Universe SN explosions take place in already shock-heated regions, with temperatures raised up to some 10^6 K, because of the action of a photo-ionizing flux from massive young stars, stellar winds, and previous SN explosions. We point out that all this is not taken into consideration in Thornton's study (while it is, for example, in Cho and Kang 2008), so a large range of uncertainty still affects the description of the whole process. A single star particle represent an entire SSP, so a large number of SN explosions are expected to take place within a single star particle over a rather long time scale. The number of SN explosions (attention must be paid to release the energy in discrete bursts consistently with the number of real explosions), as well as the amount of gas released by a SSP of given age and mass during a time-step and its chemical composition are computed with the technique described in Lia et al. (2002), using theoretical SSPs (in EVOL, the Padova tracks are adopted, see Greggio and Renzini 1983), and adopting the initial mass function for the real star in the SSP (star particle) of Kroupa et al. (2001). We refer the reader to the above papers for all the details. It is worth clarifying that we do *not* adopt the Lia's stochastic approach to model SN explosions. On the contrary, each SSP continuously releases gas and energy at each time step.

Finally, the total budget of thermal energy produced (by stellar winds and SN explosions) at each time-step is given continuously to a sort of gas reservoir assigned to star particles, ultimately formed by the gaseous material ejected from the SSP. This reservoir, due to its high temperature, acts a piston on the neighboring particles, effectively injecting kinetic energy into the surrounding interstellar medium. In the next Section we clarify some aspects concerning this method.

A crucial issue in numerical simulations of galaxy formation is the *correct treatment of the gas ejected by SSPs*. Since it is not feasible to continuously form new gas particles from the gas ejected by SSPs, it is necessary to develop a consistent, yet simple method to model this process.

This gas is essentially composed by matter ejected by stars in form of winds and/or SN explosions.

It has long been known that distributing the feedback energy among neighboring particles and then letting them normally evolve among the other cold ones gives poor results and very low feedback efficiencies (Katz 1992). Also, distributing the gas released by the SSP to nearby SPH particles at each time-step has proven to be inefficient. Therefore, as a first step we choose to keep the ejected gas fixed to the parent stellar particles and we subdivide the evolution of the ejected gas into two temporal steps. The gas is kept locked to the parent SSP during the first stages of its evolution, namely from its first appearance in the SSP to the age of $t_{\text{eject}} \simeq 8 \times 10^7$ years, a typical time scale for Type II Supernovae (short lived) to disappear.

When the age t_{eject} is reached, the hybrid stellar/gaseous particle is let free to search among its neighboring hybrid particles, to collect a sufficient amount of gas (ejected by nearby SSPs, which must be beyond their first evolutionary phase too) to form a whole new gas particle. If a mass of gas similar to the mass of a standard SPH particle is collected (in the present models we allow a difference in particle masses of $\sim 50\%$), then a new gaseous particle is spawned, with position, velocity and temperature obtained from the weighed mean over all contributing parent star particles (which are consequently depleted of their gaseous content). From now on, the new particle is let free to evolve normally.

This method has proven to give satisfactory results, at the expense of a slight increase of the total number of particles during the run.

8.9.6 Chemical enrichment

Stellar gaseous ejecta contain heavy elements, which are redistributed within the surrounding gas through a diffusive process, similar to the diffusive approximation for the thermal conduction mechanism. The diffusion of heavy elements in the ISM is included in EvOL, using the same formal law adopted for thermal conduction (which is essentially a matter of electron diffusion). The coefficient for the diffusion is obtained computing a typical diffusion velocity in the same way suggested by Monaghan for the electron diffusion speed, i.e. $v = (\bar{c}_{ij} + 4|\mu_{ij}|)$. All details are described in Merlin et al. (2012, in preparation) and MerlinPhD thesis (2009), to whom the reader should refer.

We also include in a very simple fashion a model for an early population of metal-free stars otherwise known as Population III (Pop III) stars (e.g., Bromm et al. 2002; Yoshida et al. 2003, and references therein for more details). We assume that when a metal-free gas particle is selected to form stars for the first time (see Sec. 8.9.3) it generates a Pop III SSP, mainly composed of massive stars because the underlying initial mass function is currently believed to be heavily skewed towards massive stars (Bromm et al. 2002). Consequently the Pop III SSP is short lived and nearly suddenly injects lots of energy into the interstellar medium by stellar winds and supernova explosions. In our model, a gas particle undergoing this process does *not* turn into a stellar particle, but has simply its temperature raised to the temperature $T \simeq 2 \times 10^7$ K, and acquires some metallicity, approximately $Z \simeq 10^{-4}$.

8.10 Resolution

8.10.1 Open issues Jeans mass and resolution

Owen and Villumsen (1997) investigated the properties of hybrid gravita-

tional/hydrodynamic simulations, examining both the numerics and the general physical properties of gravitationally driven hierarchical collapse in a mixed baryonic/Dark Matter fluid. In their study they found that, while the DM achieves convergence provided that the relevant scales dominating non-linear collapse are resolved, if there is no minimum baryonic collapse mass or if this scale is not resolved, then the baryon results may be wrong by a systematic factor. On the other hand, if the gas has a minimum temperature and the corresponding Jeans mass is resolved, then the baryons also converge. They concluded that this is due to the fact that under hierarchical structure formation, on all scales there is always an earlier generation of smaller scale collapses, causing shocks that irreversibly alter the state of the baryon gas.

In a simulation with finite resolution therefore such earlier generation collapses is always missed, unless a physical scale is introduced below which such structure formation is suppressed in the baryons. Such a physical scale is naturally provided by the local Jeans length.

Note that the computation of the Jeans mass for gaseous particles within a cosmological simulation is a subtle issue. For a single-component fluid (i.e., a self-gravitating gas) Jeans equation applies, but when dealing with a multi-component fluid (i.e. gas *stars* and *Dark Matter*) one must consider that while the pressure support is furnished by the gaseous component only, all kinds of matter contribute to the gravitational potential of the system.

8.10.2 Artificial pressurization

We also include a density-dependent pressure limit to avoid artificial clumping of poorly resolved gas. The method is similar to the one presented by Robertson and Kravtsov (2008). In practice, at each time-step all SPH particles compute the local Jeans mass, which is a function of density and temperature, i.e.

$$m_{Jeans} = \frac{4\pi}{3} \rho_{gas} \left(c_s \sqrt{\frac{\pi}{G\rho_{tot}}} \right)^3, \quad (8.26)$$

where $c_s = \sqrt{\gamma(\gamma - 1)u}$ is the local sound speed, ρ_{gas} is the density of the SPH particle, and ρ_{tot} is the *total* local density (i.e., gas plus DM and/or stars)¹. Then, the internal energy of each particle (limited to computing the hydrodynamical forces) is chosen as the maximum between the real value and an “effective” value, defined as the energy that the gas sphere should

¹This expression for the Jeans mass of a gas sphere in presence of a non collisional component can be obtained from a perturbative analysis of the stability of a pressurized cloud.

have to be stabilized against collapse:

$$u_{eff} = N_{jeans} \times \frac{1}{\gamma(\gamma - 1)} \rho_{tot} \left(\frac{3N_{SPH}m}{4\pi^{\frac{5}{2}}\rho_{gas}} \right)^{\frac{2}{3}}, \quad (8.27)$$

where m is the particle mass, N_{SPH} is the number of particles required to resolve a region (indicatively, this number can be considered equal to the typical number of neighbours, i.e. about 60 in our models, or a multiple of this value), γ is the adiabatic index, and N_{jeans} is a free parameter ≥ 1 . In these models we use $N_{jeans} = 15$ (see Merlin et al. 2012, in preparation, for all details).

Note that u_{eff} is only used to compute the mechanical acceleration due to the internal gas pressure. In the expressions for the variation of internal energy, dissipation, and cooling, the *real* value of u is adopted. In this way, the temperature of gas particles can reach low values, without reaching high densities, thus creating regions of cold material without spurious numerical clumping of particles.

8.11 Parallelization

The parallelization of the PD-TreeSPH, started by C. Dalla Vecchia in his Master Thesis, has been completed in EVoL, using OpenMPI libraries. To properly work within a parallel ambient, particles must first of all be subdivided among the CPUs, so that each processors only have to deal with a limited subset of the total number of bodies. Thus, at each time-step the spatial domain is subdivided using an *Orthogonal Recursive Bisection* (ORB) scheme. In practice, from the beginning of a simulation each particle keeps track of the cumulative time spent to compute its properties, storing it in a “work-load” variable. Then, at each time-step the spatial domain is subdivided trying to give each CPU the same total amount of work-load, re-assigning particles near the borders among the CPUs. To this aim, the domain is first cut along the x -axis, at the coordinate x_{cut} that best satisfies the equal work-load condition among the two groups of CPUs: the ones that have their geometrical centres with $x_{centre} \leq x_{cut}$, and the ones that have $x_{centre} > x_{cut}$. In practice, each CPU exchange particles only if its borders are across x_{cut} . Then, the two groups of CPUs are themselves subdivided into two new groups, this time cutting along the y -axis (at two possibly different coordinates, because they are independent from one another having already exchanged particles along the x -axis). The process is iterated until necessary, cutting recursively along the three dimensions. It should be noted that this scheme only allows the use of a power-of-two number of CPUs, while other mechanisms (e.g. a Peano-Hilbert domain decomposition, see e.g. Springel 2005) allow for a more efficient exploitation of the available resources.

It may sometimes happen that a CPU wants to acquire more particles than permitted by the allocation of its arrays. In this case, data structures are re-allocated using temporary arrays to store extra-data: see the next sub-Section for a description of the procedure.

Apart from the allocation of particles to the CPUs, the other task for the parallel architecture is to spread the information about particles belonging to different processors. This is achieved by means of “ghost” structures, in which such properties are stored while necessary. A first harvest among CPUs is made at each time step *before* computing SPH densities: at this point, position, masses and some other physical features (i.e. temperature of gaseous particles if the multi-phase recipe is adopted) of nearby nodes and particles, belonging to other processors, are needed. Each CPU “talks” with another CPU per time, first importing *all* the data-structures belonging to it in temporary arrays, and then saving within its “ghost-tree” structure only the data relative to those nodes and particles which will be actually used. For example, if a “ghost node” is sufficiently far away from the active CPU geometrical position so that the gravitational opening criterion is satisfied, there will be no need to open it to compute gravitational forces, and no further data relative to the particles and sub-nodes it contains will be stored in the ghost-tree.

A second period of communication among CPUs is necessary before computing accelerations. Now, each particle needs to know the exact value of the smoothing and softening lengths (which have been updated during the density evaluation routine), as well as many other physical values, of neighboring (but belonging to different CPUs) particles. The communication scheme is exactly the same as before, involving “ghost-tree” structures. It was found that, instead of upgrading the ghost-tree built before the density evaluation, a much faster approach is to completely re-build it.

8.11.1 Data structures

All data structures have been made dynamically adaptable to the size of the handled problem. In practice, at the beginning of a simulation run all arrays are allocated so that their dimension is equal to the number of particles per processor, plus some void margin to allow for an increase of the number of particles to be handled by the processor.

Anyway, whenever a set of arrays within a processor becomes full of data and has no more room for new particles, the whole data structure of that processor is re-scaled increasing its dimensions, to allow for new particles to be included. Of course, also the opposite case (i.e. if the number of particles decreases, leaving too many void places) is considered.

To this aim, the data in excess are first stored in temporary T arrays. Then, the whole “old” data structure O is copied onto new temporary N arrays, of the new required dimensions. Finally, T arrays are copied into

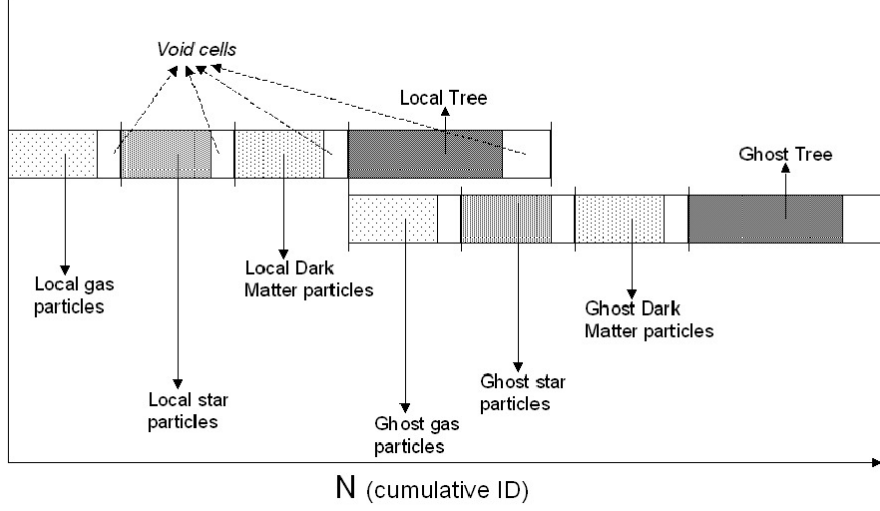


Figure 8.1: Data structures in EVoL. Void cells are included to allow for new particles to be created and/or imported from other CPUs. Ghost structures are reallocated at each time-step.

the N structure; this substitutes the O structure, and is then deleted. This procedure is quite memory-consuming. Anyway, it ideally has to be carried out only a few times per simulation, at least in standard cases. Moreover, it is performed *after* the “ghost” scheme described above has been completed, and ghost-structures have been deallocated, leaving room for other memory-expensive routines.

8.12 Simulations of galaxy formation: setting the initial conditions

In a typical simulation of galaxy formation, the initial conditions have been set as follows. A flat Universe and a concordance Λ CDM cosmology has been assumed. The cosmological parameters are consistent with the WMAP 5-year data (Hinshaw et al. 2009) and are summarized in Table 8.1.

The software package COSMICS (Bertschinger 1995) was used to generate cosmologically perturbed initial grids of particles. COSMICS allows to specify the properties of the constrained density peak to model the density field in the desired way. We force a gaussian spherical over-density with the average linear density² contrast $\delta\rho = 3$, smoothed over a region of radius 3.5 comoving Mpc. COSMICS returns the initial comoving positions and

²The linear density contrast is defined as $\frac{\langle\rho\rangle}{\rho_{bg}} - 1$ (where ρ_{bg} is the average matter density of the Universe)

Table 8.1: *Cosmological parameters adopted in the simulations.*

Symbol	Description	Adopted value
H_0	Hubble constant [km/s/Mpc]	70.1
Ω_Λ	Dark Energy density	0.721
Ω_b	Baryon density	0.0462
n_s	Scalar spectral index	0.960
σ_8	Fluctuation amplitude at $8h^{-1}$ Mpc	0.817

the initial peculiar velocities of the particles at the moment in which the particle with the highest density is exiting the linear regime.

In some cases, taking advantage of an option of the code, each grid was then constrained to have a peak of density in the center of the comoving box, with a given mean over-density within a fixed comoving radius. Then, each particle was split in two: a Dark Matter particle, of mass $(1 - \Omega_{bar}) \times m_{init}$, and a slightly randomly offset gaseous particle, of mass $\Omega_{bar} \times m_{init}$. The initial temperature of gaseous particles was set as in Merlin and Chiosi (2007).

We then single out a sphere centered on the center of the box, change the particles coordinates from comoving to the proper physical values, and add a radial outward velocity to each particle, proportional to its radial position and to the initial redshift of the simulation, thus mimicking the effects of a centered outward directed Hubble flow. A minimal amount of solid-body rotation is also added, with spin parameter $\lambda = 0.02^3$

When working in physical coordinates, the protogalaxy was let evolve in void.

When adopting comoving integration, the outskirts of the comoving box were “degraded” using a code written by myself. Masses and positions of particles were averaged in order to obtain coarser particles, which were used to replace high-res ones outside the central region, thus reducing the total number of bodies. The process can be iterated as many times as one desires. Experiments have been made using a single jump from the high-res to the low-res zone: the resulting mass discontinuity (up to a factor of ~ 20) caused very wrong determinations of SPH densities on the boundary layer, causing unavoidable numerical artifacts. Adopting a smoother sequence of mass jumps (using refinement mass factors of ~ 4) substantially reduced the

³The spin parameter λ is

$$\lambda = \frac{J|E|^{\frac{1}{2}}}{GM^{\frac{5}{2}}} \quad (8.28)$$

where J is the angular momentum, E is the initial binding energy, and M the total mass of the system. Typical values of λ range from 0.02 and 0.08 (White 1984), which corresponds to angular velocities of the order of fractions of a complete rotation over time-scales as long as about ten free-fall time-scales (Carraro et al. 1998).

artificial discontinuity. Anyway, this issue confirms the fact that the SPH can prove quite inefficient in the determination of densities if particles of (very) different masses are used. Of course, coarse gas particles give a somehow poor description of physical processes; Steinmetz and White (1997) have also shown that the interaction among particles of different masses may lead to unphysical two-body heating, inhibiting the cooling of gaseous particles. This issue deserves careful analysis in future work.

Chapter 9

Profiling EvoL: actions and measurements

This chapter describes the actions and measurements of suitable parameters we have made to assess the profiling of EvoL. These are split in different groups and are obtained using different computers. The list of simulations and computers are given in the Appendix. In order to get meaningful measurements of code performance, the initial conditions for each simulation are created always using the same procedure. The performance is evaluated for realistic situations, in particular the number of particles is about the same as that used for real models.

9.1 The initial conditions

To compare the behavior of the code in different situations, we first generate different cosmological boxes containing particles of DM and BM with specified masses, positions, and velocities, in the way described in the previous Chapter.

We produce models with (auto-)similar initial conditions but different numbers of particles. By doing so, we introduce only the changes that are strictly necessary to obtain the desired differences in the initial conditions from model to model. The size of the cosmological box is, for all models, $l=9.2$ comoving Mpc, populated with a grid of $2 \times 18^3, 2 \times 24^3, 2 \times 30^3, 2 \times 46^3, 2 \times 58^3, 2 \times 74^3$ particles¹ which are perturbed by their otherwise regular positions consistently with cosmological random gaussian fluctuations, and finally imposing a constrained density peak, to force the formation of a virialized structure at some early time close to the center of the box.

In one case we use already evolved initial conditions; this corresponds to the case of high density medium mass (thereinafter referred to as HDMM)

¹The number 2 has been adopted because of the presence of an equal number of particles of DM and BM.

model by Merlin et al. (2012a).

9.2 Overview

The measurements are carried out in different ways, using the techniques presented in the previous sections. The tests are made both directly by measuring the total amount of CPU time consumed to evolve a particular physical configuration for a suitable interval of cosmic time, and using specifically designed profiling tools such as **gprof**, and **SCALASCA** or Trace Analyzer & collector software. The runs are carried out on different machines. The results concern:

- Simulations of the cpu/core ratio, to see if it changes by the balance of profiling (**gprof** and **SCALASCA**);
- Measurements of the total time required to compute simulations from identical initial conditions but increasing progressively the number of processors (strong scaling) with **gprof** and **SCALASCA**;
- Simulations carried out by increasing the size of the problem to ascertain whether the problem is “self-similar” at different resolutions. Using a constant number of processors (weak scalability), one can calculate/estimate how long a run should be;
- Simulations with progressively increasing number of processors to highlight the importance of communications in problems of growing size;
- The optimal number of particles per processor (different numbers for different architectures);
- A series of simulations of various sizes to highlight cache problems. Remember that cache problems are masked when the whole problem is in the cache itself so particular care must be taken to pin it down.

Where technically possible, different tools have been used on the same machine to evaluate any differences between the results. The profiling tools altering the code can, in principle, give different results. Thus, measurements made with different software, can be considered, if in agreement with each other, more significant than those obtained with a single tool. Apart from some minor differences, the results are mutually consistent.

In general most of the measurements used in this thesis were performed with **gprof** and **SCALASCA** on SP6 (CINECA supercomputer), while only a small number were performed with the Trace Analyzer on MONSTER, the Cluster of the Astronomy Department of Padova. Finally, the measurements on the cache-grind were done on two small computers, BUD and HAL9002, at the Padova Department of Astronomy. The tests to verify the performance

after the optimization process have been made on BUD and HAL9002 to save computational time at CINECA.

9.3 Impact of profiling on the computational time

First of all, we need to perform some simulations with and without profiling, to assess the impact of profiling on the total computational time. They are used to estimate how many simulations can be calculated and how many processors must be used in order to get satisfactory results within the allocated CPU time. At the same time, these simulations can be used to estimate the scalability of the code, the computational time, and number of necessary processors at increasing size of the problem. To this aim many simulations are launched varying the profiling options, namely, without profiler, with **gprof**, and with **SCALASCA**. Table 9.1 shows the results of two simulations normalized for convenience to the fastest case (no profiler).

Table 9.1: Impact on the computational times of the different tools under examination. The times are normalized to the run without profiling.

PROFILER	NAME	Ncore	NSTEP/CPUTIME
None	HDMM	16	1
SCALASCA	HDMM	16	3.84
gprof	HDMM- gprof	16	1.35
None	HDMM	8	1
SCALASCA	HDMM	8	3.64
gprof	HDMM- gprof	8	1.16

The impact of the profiling on the computational time is not negligible, it amounts to a maximum factor of about 4 for **SCALASCA**, and 1.4 for **gprof**. I recall that in order to correctly measure the timings and to get the profiling, one has to estimate in advance the duration of the simulation to be sure that it is regularly terminated and not abruptly interrupted for expiration of computational allocated time.

9.4 CORE vs. CPU

The knowledge of the platforms as mentioned above is essential to have codes that exploit the performance features of the hardware in use. The CINECA SP6 can work in Simultaneous Multi-Threading (SMT). A series of simulations were run in this mode to see if we got the expected benefits. Three simulations (HDMM) were run in CORE (standard) and CPU (SMT)

mode for which we got the number of “nstep per hour” in the two modes. The results are listed in Table 9.2.

Table 9.2: Computational times in CORE and CPU modes. In the CORE mode the number of used processors is equal to the number of tasks, while in the CPU mode it is half this number. The number of time-steps per hour is only slightly higher in the CORE mode, even with a doubled hardware (data from HDMM simulation)

N_task	CORE	CPU	CORE/CPU
32	66.45	59.25	1.12
16	36.90	32.38	1.14
8	18.50	16.25	1.14

The results are very good: using the SMT mode, the total computational time resulted similar to that obtained using about twice as much hardware resources. In all tests we noted an additional increase in speed of approximately 1.8 times, i.e. using N-core physical “CPU mode” one gets performances very similar to those obtained with $1.8 \times N$ physical cores.

Profiling in SMT mode. Another extremely important point to check is whether the profiling procedure affects profiling itself in some way. To this aim one has to check on the SMT mode that the weights of individual routines are not significantly altered by profiling.

In Table 9.3 examples of percentages of time spent by selected routines with `gprof` flat profile in CORE and CPU mode are given for comparison .

Table 9.3: Profiling with `gprof` in CORE and CPU modes, using the same number of tasks. The profilings are very similar, so the Cpu modality does not substantially change the computational time ripartition between the routines in EVOL.

Name	%time _{CORE}	%time _{CPU}
.accelerate	29.1	29.6
...mcount	14.2	15.3
.getneigh_accel_all	11.5	12.3
..atan	6.8	6.0
.soft_phi	5.4	5.2
.soft_grav	5.3	4.8
.getneigh_dens_all	4.4	4.5
.getneigh_accel_dm	4.4	4.3
.density	3.1	3.0

The profile obtained in the SMT mode is almost identical to the one

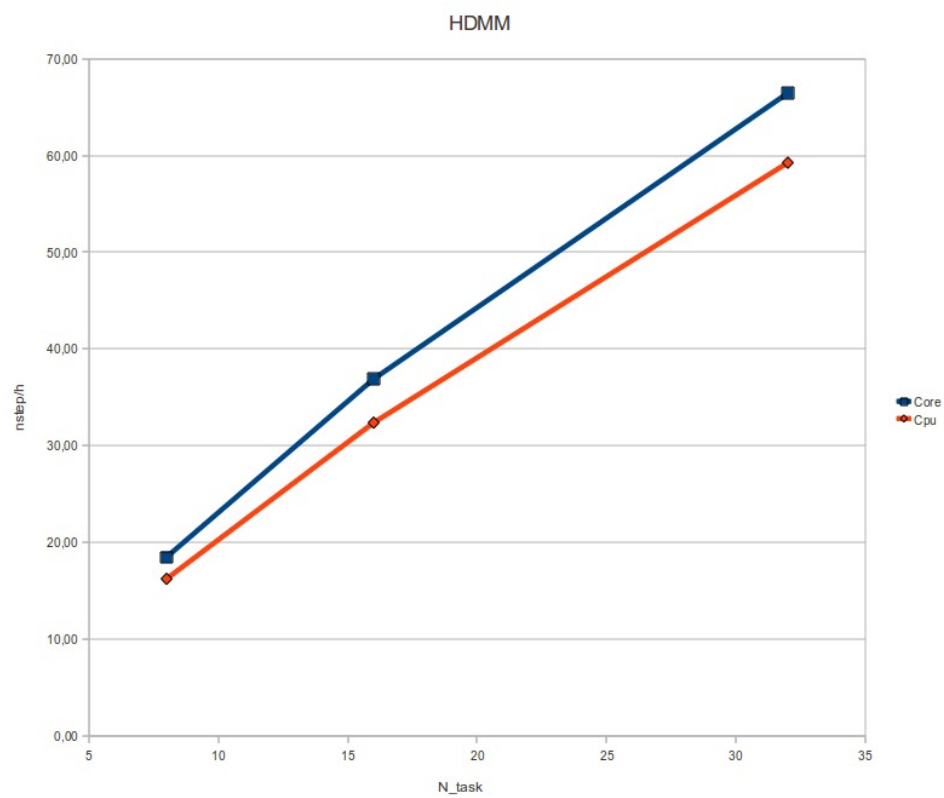


Figure 9.1: Same as in Tab. 9.2, in graphical form.

obtained with the standard configuration on the same problem. As one can note the differences are less than 1 %, i.e. fully adequate to our purposes. We also must verify that the measurements with SCALASCA are not significantly influenced by the SMT mode.

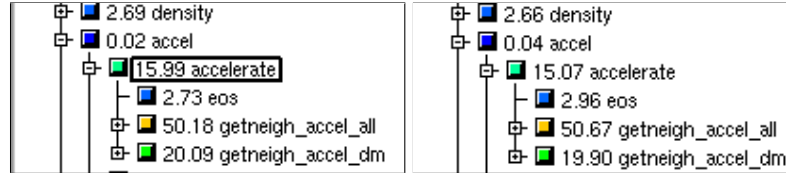


Figure 9.2: Scalasca profiling in CORE and CPU modes. As for the gprof data 9.3, the results show that the CPU mode does not change the computational time ripartition among the routines.

The data produced with SCALASCA are shown in Fig. 9.2. The results obtained with gprof and SCALASCA allowed us to safely use the SMT mode throughout this study.

9.5 Weak and strong scalability

To properly plan the subsequent simulations it is necessary to know the behavior of the code at varying the number of particles and the number of processors. To this aim, two sets of simulations were carried out with and without profiling.

Strong scalability. We have chosen a simulation of intermediate size (2×46^3 particles) and launched it using an increasing number of processors without profiling. We then calculated the number of particles per second per CPU. The results are normalized because we are only interested in time ratios. The speed-up shows us how many times the computation time decreases as the number of processors increases. The data is given in Table 9.4 and shown in Figure 9.3.

Table 9.4: Scalability: number of particles per second and speed-up, in the “46” strong scalability test.

N.cpu.	Part/sec	SpeedUp
32	4349.63	11.52
16	3208.58	8.50
8	2518.44	6.67
4	1092.72	2.89
2	772.51	2.05
1	377.64	1.00

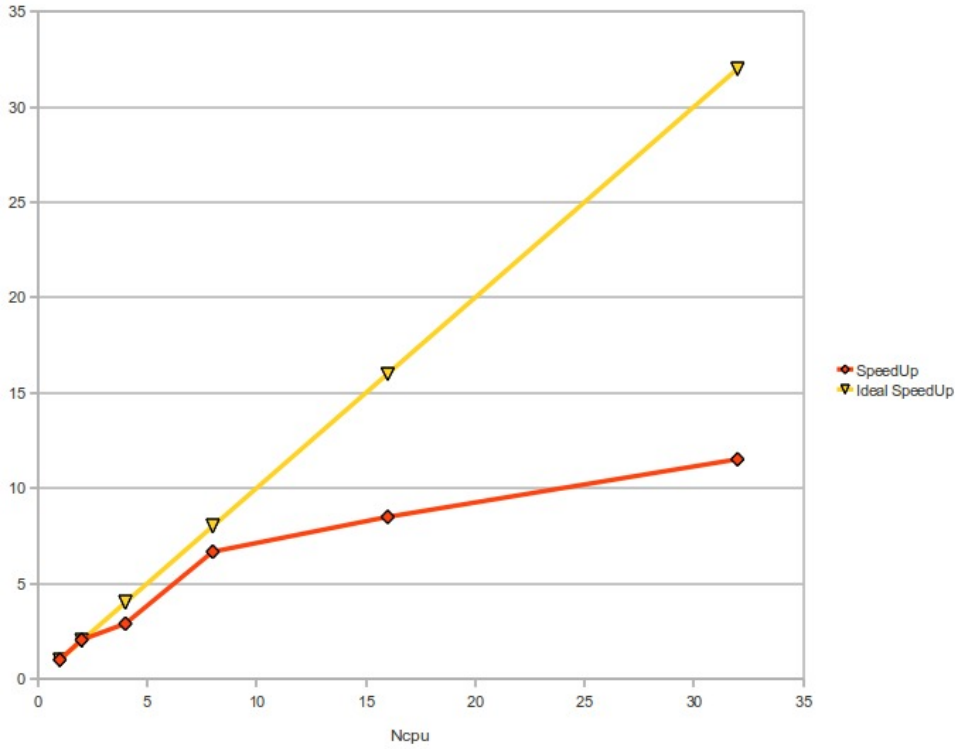


Figure 9.3: Same data of 9.4 compared with the ideal speed-up.

Weak scalability. In this case simulations of increasing size were launched on a single processor. The simulations are listed in Table 9.5 where the data of interest have been normalized. For our purposes the important thing to know is how to increase the computing time at increasing the number of particles in the simulation.

9.5.1 Planning the measurements

Thanks to the data we obtained, we can plan the simulations to perform with SCALASCA and **gprof**², i.e. the number of simulations to be carried out, the number particles per simulation, the number of processors in relation to the total time allocated to us by CINECA. Furthermore, we also schedule a number of simulations with increasing number of particles and processors. Some simulations are also scheduled on the clusters and PC's in the house, using Trace Analyzer for the former and cache-grind for the latter. The details are presented in the Appendix.

²The code in use is the version after pre-processing optimization

Table 9.5: Particles per second on a single processor, progressively increasing the dimension of the problem (weak scalability). In the right column, data are normalized to the largest number.

N.part.	Part/sec	Part/sec/Partsecmax
2×74^3	119,20	0,17
2×58^3	195,83	0,27
2×46^3	377,64	0,53
2×30^3	578,23	0,81
2×24^3	667,23	0,93
2×18^3	713,65	1,00

9.6 Computation time and communication

Let us examine in detail only a few cases, since all the others are similar. We try to understand where the code spends most of the time, separately for calculations and communications. Two typical situations for the galaxy models to simulate are:

- Comoving coordinates, with periodic conditions of DM, gas and stars (when formed)
- Physical coordinates, without periodic conditions of a single halo of DM, gas and many stars (advanced stage of evolution).

The simulation named HDMM has physical coordinates, the one named “46” has comoving periodic coordinates with 2×46^3 particles. The results, obtained with SCALASCA, are given in Table 9.6.

Table 9.6: Computation and communication times of simulations in physical (HDMM) and comoving (“46”) coordinates. Two cases with different numbers of processors are shown.

N.proc.	46 exe	46 MPI	HDMM exe	HDMM MPI
2	96.7%	3.3%	98.3%	1.7%
128	75.4%	24.6%	88%	12%

The same information is also obtained using MONSTER with Trace Analyzer and 18^3 particles, see the data in Table 9.7. The different hardware increases the relative weight of the network communications. Remember, however, that the number of processors used in these tests is deliberately excessive in relation to the number of particles to follow. A few hundred particles per processor is too small a number. The typical configurations to

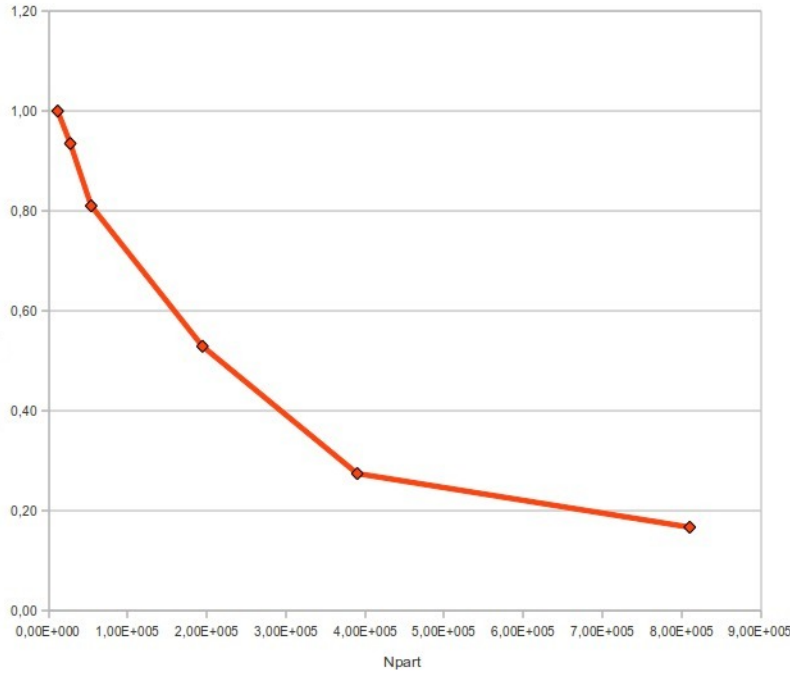


Figure 9.4: Same data of 9.5, in graphical form.

use on different machines should have $\sim 10^4$ particles per processor. With the small number we have used, we can get only upper limits. Also we can see the transition between communications between processors belonging to the same node and nodes belonging to different processors³.

Table 9.7: Fractional ripartition between computation and communication for the 2×18^3 particles simulation on MONSTER (data are obtained with Trace Analyzer).

N.proc.	18 exe	18 MPI
2	98.4%	1.6%
8	95.5%	4.5%
16	79.4%	20.6%

In a typical run for a standard model of galaxy evolution during a number of time-steps, the global weight of the MPI communications time with respect to the overall CPU time has proven to be small, between 10% and 25%. The computational time dominates. This implies that optimization

³The computational nodes of MONSTER are composed of 8 processors.

must first cure the computational time, and only subsequently the communication time.

Execution Time. SCALASCA is very useful to identify the routines that waste time. Selecting only the execution time, SCALASCA allows one to understand which percentage of time is spent in each different routine (9.5

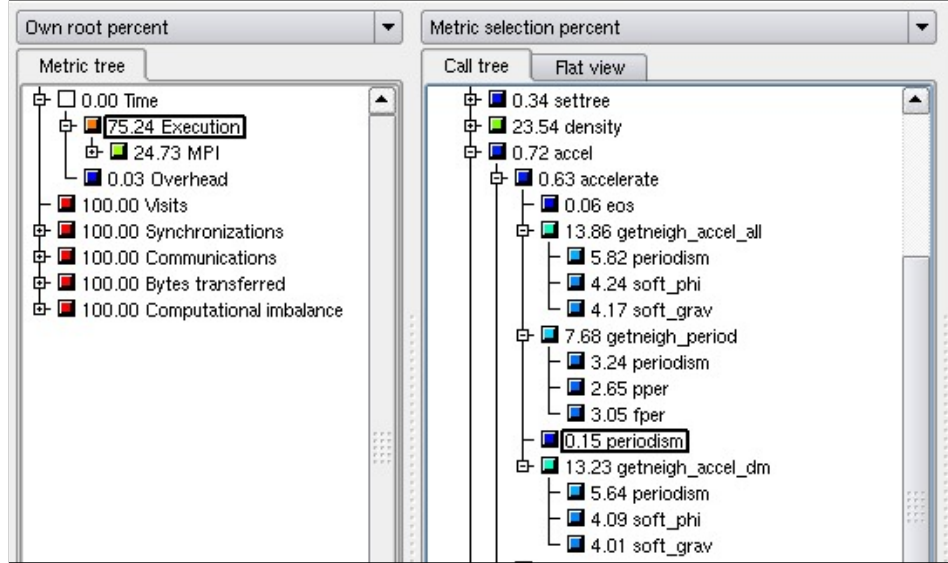


Figure 9.5: SCALASCA call-tree for the “46” run on 128 processors.

In Figures 9.5 and 9.6 one can see the great importance of the routines for calculating the acceleration and density. In turn, these routines call other routines and these latter also contribute differently to the total amount time. For example, 73.23% of the time spent in **Accel** is mostly spent distributed among the various calls. Among the most important routines, selected according to the call tree of SCALASCA, we note **periodism**, **soft_phi**, **soft_grav**, **pper** and **fper**. The measurements made with **gprof** on SP6, MONSTER and BUD yield much similar results, and do not provide additional information. Their results are not reported here for the sake of brevity.

Speedup and efficiency With the SCALASCA data we can discuss speedup and efficiency for one of the cosmological simulations “46”⁴. The other cases are similar. They are listed in Table 9.8. In the case of the speed-up analysis we can separate the contributions to the total time by calculations and MPI communications. The results are indeed more significant than expected.

⁴the simulation with 2×46^3 particles

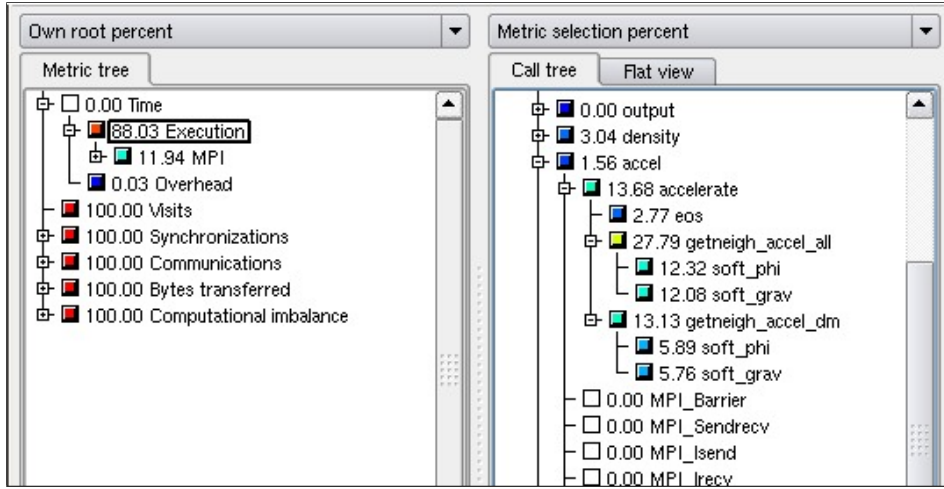


Figure 9.6: SCALASCA call-tree for the HDMM run on 128 processors.

Table 9.8: Speedup and efficiency obtained with Scalasca for the “46” run on 128 processors.

N_proc	Speedup	Efficiency
1	1.00	1.00
2	1.49	0.74
4	2.50	0.62
8	7.07	0.88
16	9.24	0.58
32	13.05	0.41
64	25.17	0.39
128	38.01	0.30

The Figures 9.7 and 9.8 show the speed-up and efficiency for EvOL using up to 128 processors.

There is a decrease in efficiency with increasing number of processors. One can see from the graph that with 64 processors we use only about 40% of the available power. With 8 processors, there is a peak of efficiency, probably due to hardware features of SP6. The speedup with 128 processors is 38, far from the ideal (128): this result indicates that there are not only MPI communications limiting the scalability of EvOL. The most likely cause is the presence of an important serial fraction of the code.

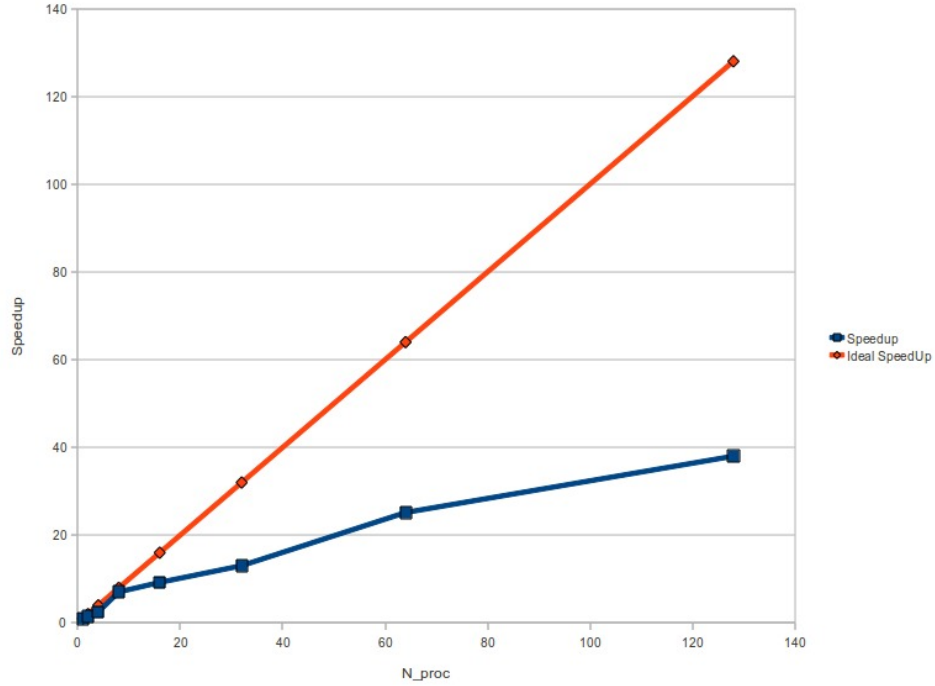


Figure 9.7: Speed-up and ideal speed-up up to 128 processors with data from Scalasca. Data are the same as in Tab. 9.8, left column.

9.6.1 Serial fraction

The data at our disposal are sufficient to calculate the serial fraction. In fact, only two measurements would be sufficient for this calculation. Due to the dynamic nature of the code, we expect the “true” serial fraction to depend on many factors. While the program runs through different routines, it exchanges data, activates or excludes the presence of various physical processes that lead to the execution of different routines in different processors, writes files to disk, etc. Calculating the serial fraction in this way would provide an unreliable result. We have used several measurements, first with a simple graphical method, and then with a more accurate procedure, getting what can be call a “mean serial fraction”. Nonetheless, even this can be a useful estimate of the serial fraction of the code. This estimate allows us to understand the scalability limits of EVOL and what we can expect from using an increasing number of processors. The calculation was done using two sets of measurements obtained from the simulations calculated on MONSTER and SP6.

Graphical method We take two simulations calculated on a few processors and two on a larger number of CPUs among those calculated with

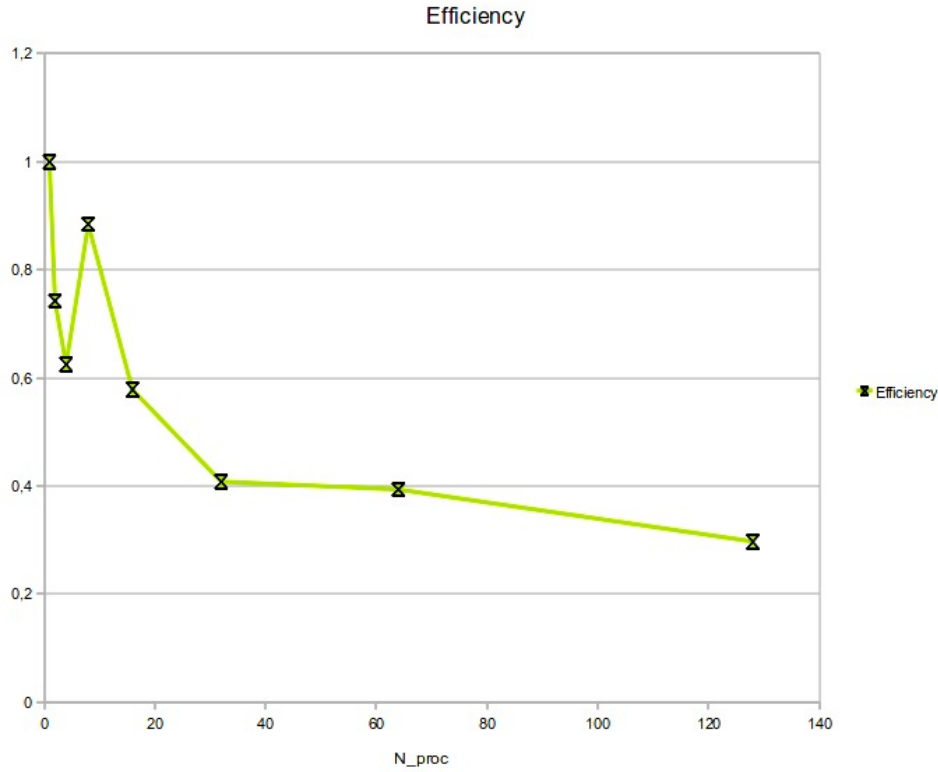


Figure 9.8: Efficiency up to 128 processors with data from Scalasca. Data are the same as in Tab. 9.8, right column.

Trace Analyzer. The total time is equal to the sum of computational time on each processor, the time on single processor (which gives the trace-analyzer net of MPI) is given by a serial time (the same for all) plus a parallel part, which decreases at increasing number of processors. So

$$T_{tot} = \sum_{n=1}^N \left(S + \frac{P}{N} \right) = NS + P \quad (9.1)$$

These are straight lines (known computational time) that should intersect at the same point but as mentioned this does not occur in reality. The computation times (in seconds) are the known terms of the following equations:

$$\begin{cases} P + 32S = 8305.508184 \\ P + 16S = 5898.579342 \\ P + 4S = 5205.902668 \\ P + 2S = 4445.101888 \end{cases} \quad (9.2)$$

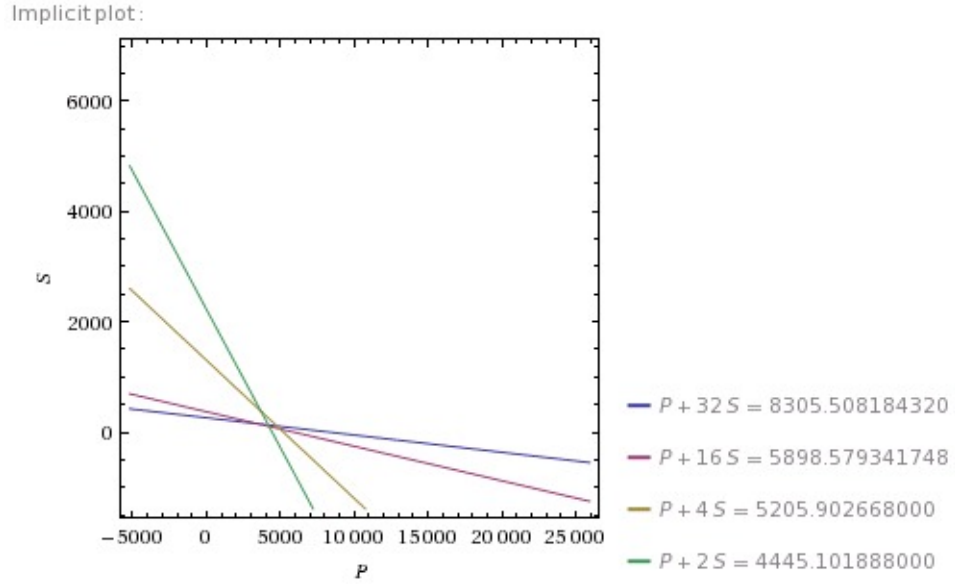


Figure 9.9: Approximate crossing of the lines for the computation of the serial fraction with the simple graphic method described in the text. Data are from Trace Analyzer on MONSTER.

The straight lines intersect over in a wide range of values.

$$4200 \leq P \leq 4700 \quad (9.3)$$

$$60 \leq S \leq 130 \quad (9.4)$$

$$(9.5)$$

I simply take the average values $P \approx 4450$, $S \approx 95$

$$F_{ser} = \frac{S}{P+S} \approx 0.0209021 \quad (9.6)$$

$$F_{par} = \frac{P}{P+S} \approx 0.979098 \quad (9.7)$$

The conclusion is that the code was well parallelized, leaving a serial part amounting to only a 2 %. This small percentage is however important in determining the speed-up limit.

The same method is applied to a simulation performed on SP6 (simulation with 2×46^3 particles). The times are obtained with SCALASCA.

$$\left\{ \begin{array}{l} P + 64S = 153617.65 \\ P + 32S = 148152.88 \\ P + 16S = 104589.00 \\ P + 8S = 68350.38 \\ P + 4S = 96749.62 \\ P + 2S = 81354.65 \end{array} \right. \quad (9.8)$$

Once more the straight lines roughly intersect over a range of values,

$$55000 \leq P \leq 90000 \quad (9.9)$$

$$-5000 \leq S \leq 8000; \quad (9.10)$$

$$(9.11)$$

The mean values are $P \approx 72500$, $S \approx 1500$.

$$F_{ser} = \frac{S}{P+S} \approx 0.02027 \quad (9.12)$$

$$F_{par} = \frac{P}{P+S} \approx 0.97973 \quad (9.13)$$

The new results are in good agreement with the previous estimates and therefore we conclude that the serial fraction is correctly estimated.

Rigorous method. In addition to this crude graphical method, the method of least squares can be used to determine S and P. The method use a matrix formalism: Let $f(x)$ be a function with respect to the parameters

$$f(x) = p_1 f_1(x) + p_2 f_2(x) + \dots + p_k f_k(x) \quad (9.14)$$

where p_i are the k parameters, $k \ll n$ and n are the number of data points. One can rearrange the data in a oversized linear system $Ap \approx y$ where

$$A = \begin{bmatrix} f_1(x_1) & \dots & f_k(x_1) \\ \vdots & & \vdots \\ f_1(x_n) & \dots & f_k(x_n) \end{bmatrix}, p = \begin{bmatrix} p_1 \\ \vdots \\ p_k \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}. \quad (9.15)$$

Let (x_i, y_i) with $i = 1, 2, \dots, n$ be the points representing the input data. One wants to find a function f approximating the sequence of data points. This function can be determined by minimizing the (Euclidean) distance between the two sequences y_i and $f(x_i)$, i.e. the quantity

$$S = \sum_{i=1}^n (y_i - f(x_i))^2, \quad (9.16)$$

(hence the name “least squares”).

In practical cases, $f(x)$ is usually parametric: thus the problem reduces to determining the parameters that minimize the distance between the points on the curve (in the known pattern). To obtain a single optimized curve and not a bundle, one needs a number of experimental points greater than the number of parameters that determine the curve (the problem is said “overdetermined”). In general, experimental data obtained from a distribution is expected to be regulated by certain analytical relationships, then it is useful to parameterize a curve and determine the parameters so as to minimize S . The problem of minimizing S leads, therefore, to minimize the norm of the residual $\|r\| = \|Ap - y\|$

$$\|r\|^2 = \|Ap - y\|^2 = \sum_{i=1}^2 ([Ap]_n - y_n)^2 = \sum_{i=1}^n (f(x_i) - y_i)^2 = S \quad (9.17)$$

where with $[Ap]_i$ one means the i -th component of the vector product of A and p . One can minimize $\|r\|$ deriving $\|r\|^2$ with respect to each p_m and setting the derivative equal to 0:

$$\frac{d\|r\|^2}{dp_m} = \sum_{i=1}^n 2 \left(\sum_{j=1}^k a_{ij} p_j - y_i \right) a_{im} = 0. \quad (9.18)$$

These equations are equivalent to the system:

$$(Ap - y)^T A = 0 \quad (9.19)$$

Then the vector p that minimizes S is the solution of the equation:

$$A^T Ap = A^T y \quad (9.20)$$

If the rank of A is complete⁵ then $A^T A$ is invertible and therefore:

$$p = (A^T A)^{-1} A^T y. \quad (9.21)$$

Finally, to evaluate the “goodness of fit” usually one considers the coefficient of determination R^2 , measuring the goodness of fit of the linear regression estimated from the experimental data. Let us define the equation

$$M_0 = I - \frac{1}{N} \mathbf{1}\mathbf{1}'; \quad (9.22)$$

where $\mathbf{1}$ is a column vector made up of one ($nx1$), n is the number of equations and I is the identity matrix of order n . R^2 is defined as:

$$R^2 = \frac{\hat{p}^T A^T M_0 A \hat{p}}{y' M_0 y} = \frac{\hat{y}' M_0 \hat{y}}{y' M_0 y} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} \quad (9.23)$$

⁵The system is impossible when incomplete; the rank of the matrix $\neq$ the rank of the full matrix

Rewriting our problem with the symbolism of this method:

$$A = \begin{bmatrix} 1 & 32 \\ 1 & 16 \\ 1 & 8 \\ 1 & 4 \\ 1 & 2 \end{bmatrix}, p = \begin{bmatrix} P \\ S \end{bmatrix}, y = \begin{bmatrix} 8305.508184 \\ 5898.579342 \\ 4397.377416 \\ 5205.902668 \\ 4445.101888 \end{bmatrix}. \quad (9.24)$$

The result is:

$$p = (A^T A)^{-1} A^T y = \begin{bmatrix} 4356.225542 \\ 84.0686382 \end{bmatrix}, \quad (9.25)$$

Which yields:

$$F_{ser} = \frac{S}{P+S} \approx 0.02027 \quad (9.26)$$

$$F_{par} = \frac{P}{P+S} \approx 0.98107 \quad (9.27)$$

$$R^2 = 0.90781. \quad (9.28)$$

The goodness of fit is ensured by a coefficient close to one.

The same analysis made with the SCALASCA data yields similar results

$$F_{ser} = \frac{S}{P+S} \approx 0.01556 \quad (9.29)$$

$$F_{par} = \frac{P}{P+S} \approx 0.98444 \quad (9.30)$$

$$R^2 = 0.76781. \quad (9.31)$$

Here the fit is less good and this enables us to revise some of the assumptions we initially made. Basically it is a confirmation of the suspected change in the serial component. The argument is as follows: the computational time for each CPU is partly serial and partly parallel. The parallel part scales with the number of processors. We have considered constant the serial part and this allows to derive the 9.1. If the serial part is zero, then the overall computation time (CPU time) would remain constant as N processors perform a fraction 1/N of the work. A constant serial fraction implies a total time increasing linearly⁶. This fact suggests that deviations from perfect linearity are due to changes in the serial computation time (and hence the serial fraction). The causes of this behavior are easy to understand: the code evolves a dynamic system, a box such as the cosmological, and each processor will evolve to a different part of the physical properties. The computation time for different particles will be different because many quantities

⁶MPI communication time and the imperfect balance are automatically separated by SCALASCA

change: for example, the number of neighbors for the calculation of the density, the number of different cells to open the Tree for the calculation of gravity, different as the feedback must be considered and then changes the code that each processor must perform to develop the individual particles. So the value of the serial fraction to calculate as we saw above is only an estimate in the typical conditions of use.

Speed-Up limit of EvoL This value is simply given by the reciprocal of the serial fraction. So in our case it is $47.5/49.34$ (with the graphical method), and $52.82/64.26$ (with the rigorous method). These values should be considered indicative; however, we can say that we have a speed-up limit of about 64 (because the code uses powers of 2 processors at a time).

This information is useful to get an idea about whether a given simulation can be computed in less time by using multiple CPUs. The problem arises from the simple question: how much time do we need to run a simulation with a certain number of particles and a given number of processors? This is a key point to addressed in order to plan the work properly. This question in turn can be rephrased as follow: how do I have to scale the code at varying the number of particles? To answer the question a series of simulations has been run at increasing the number of particles while keeping constant the number of processors.

9.6.2 Increasing problem size

All the analysis presented so far refers to problems of fixed size. But that is not the way parallel processors are used in practice. Usually, both the size of the problem size and the numbers of processors are increased so that the time of a run remains more or less constant. According to Gustafson, the serial fraction of the time spent executing a serial code decrease as the problem size grows, because increasing the size of the problem, the time (roughly constant) required to execute a serial code becomes less and less important,

$$T_{tot} = \sum_{i=1}^N \left[S + k(n) \frac{P}{N} \right] = NS + k(n)P \quad (9.32)$$

where n is the number of particles. The (increasing) function $k(n)$ indicates the increases of the time in the parallel mode as the number of particles increases. Again the serial part is supposed to remain constant, a condition that is verified a fortiori given the Gustafson's arguments.

Our purpose here is to derive the function $k(n)$ so that we can estimate the time of a run and answer the question. The functional form of $k(n)$ is completely unknown but not difficult to discover. Our code is a classic TreeSPH code, whose part dealing with gravity has a computational complexity $O(N \log N)$ (where N is the number of particles), plus the hy-

hydrodynamic (SPH) part whose complexity is not known. However, we can assume that the total complexity is of order $O(N)$, considering the fact that doubling the SPH particles will double the work for the search of neighbors, which is the part where our algorithms spend most of the computation time. Starting from 9.32 we can write the following system of equations:

$$T_i(N, n) = NS_i(N, n) + k(n)P_i(N, n), i = 1 \dots N \quad (9.33)$$

which in our case becomes the following:

$$\left\{ \begin{array}{l} T(32, 18^3) = 32S(32, 18^3) + k(18^3)P(32, 18^3) \\ T(32, 24^3) = 32S(32, 24^3) + k(24^3)P(32, 24^3) \\ T(32, 30^3) = 32S(32, 30^3) + k(30^3)P(32, 30^3) \\ T(32, 46^3) = 32S(32, 46^3) + k(46^3)P(32, 46^3) \\ T(32, 58^3) = 32S(32, 58^3) + k(58^3)P(32, 58^3) \\ T(32, 74^3) = 32S(32, 74^3) + k(74^3)P(32, 74^3) \end{array} \right. \quad (9.34)$$

Dividing the first equation by the others, the equations can be recast as follows:

$$\frac{k(n)}{k(18^3)} = \frac{\frac{T(32, n)}{T(32, 18^3)} - 32 \frac{S(32, n)}{T(32, 18^3)}}{1 - 32 \frac{S(32, n)}{T(32, 18^3)}} \quad (9.35)$$

Now suppose that even if the serial fraction F_{ser} does not depend on the size of the problem, the number of processors is fixed (to 32). Therefore:

$$S(32, n) = S(32, 18^3) = F_{ser}T(32, 18^3) \quad (9.36)$$

and so

$$F_{ser} = \frac{S(32, n)}{T(32, 18^3)} \quad (9.37)$$

Then, by measuring time in units of $T(32, 18^3)$ and $k(n)$ in $k(18^3)$, we get

$$k_{18}(n) = \frac{T_{18}(32, n) - 32F_{ser}}{1 - 32F_{ser}} \quad (9.38)$$

We can now assume that the functional form of $k(n)$ is

$$k(n_{DM}, n_G, n_S) = (n_{DM} + n_G + n_S) \log(n_{DM} + n_G + n_S) \alpha + n_G \beta \quad (9.39)$$

where n_{DM} , n_G , and n_S are the numbers of particles of dark matter, gas and star, respectively, while the α and β are the parameters to find (if the number of equations is greater than 4, at least). In our case $n_{DM} = n_G$ and $n_S = 0$ because the tree constructed for the calculation of gravity is the same for all particles, and the search of neighbors in the SPH calculations

is performed only for the gas particles. The serial fraction is already known from the above analysis and the matrix method can be adopted. We need a set of measurements made holding constant the number of processors and varying the number of particles. The measurements were made using the computer software and SCALASCA in the CINECA SP6. The data are as follows:

Table 9.9: The increasing function $k(n)$ indicates the growth of the computational time in the parallel mode as the number of particles increases. Data are from Scalasca on 32 CPUs.

N_DM	N_gas	Execution	k(n)
18^3	18^3	2254.88	1.00
24^3	24^3	5964.42	5.82
30^3	30^3	14145.14	16.45
46^3	46^3	64299.05	81.63
58^3	58^3	155237.53	199.82
74^3	74^3	375998.74	486.72

For the calculations we have used the average values of the serial fraction both from the graphical method and the rigorous one. The values of the parameters α and β for the two sets of measurements are given in Table 9.10.

Table 9.10: α and β parameters. See text for details.

Ser. Frac.	α	β
Graphical	$1.22 \cdot 10^{-4}$	$-2.12 \cdot 10^{-3}$
Rigorous	$9.19 \cdot 10^{-5}$	$-1.59 \cdot 10^{-3}$

Given the function $k(n)$ we can find the fit for the computation time, and predict the time of calculation of new simulations. All this is valid on a specific machine. The Figure 9.10 shows the result of the fit ?? of computing time.

The knowledge of the function $k(n)$ together with the knowledge of the serial fraction (which gives us the law of strong scalability) enable us to better estimate how long does it takes to build a CPU simulation.

The times calculated are net of MPI communication and any imbalance of the code. It should also be noted that the above considerations do not allow us to calculate (in the sense of providing for) the computation time of a simulation, because we cannot know the length of a time-step (which is related to how the simulation evolves as a whole). The considerations and equations are valid and can be usefully applied when, known the computa-

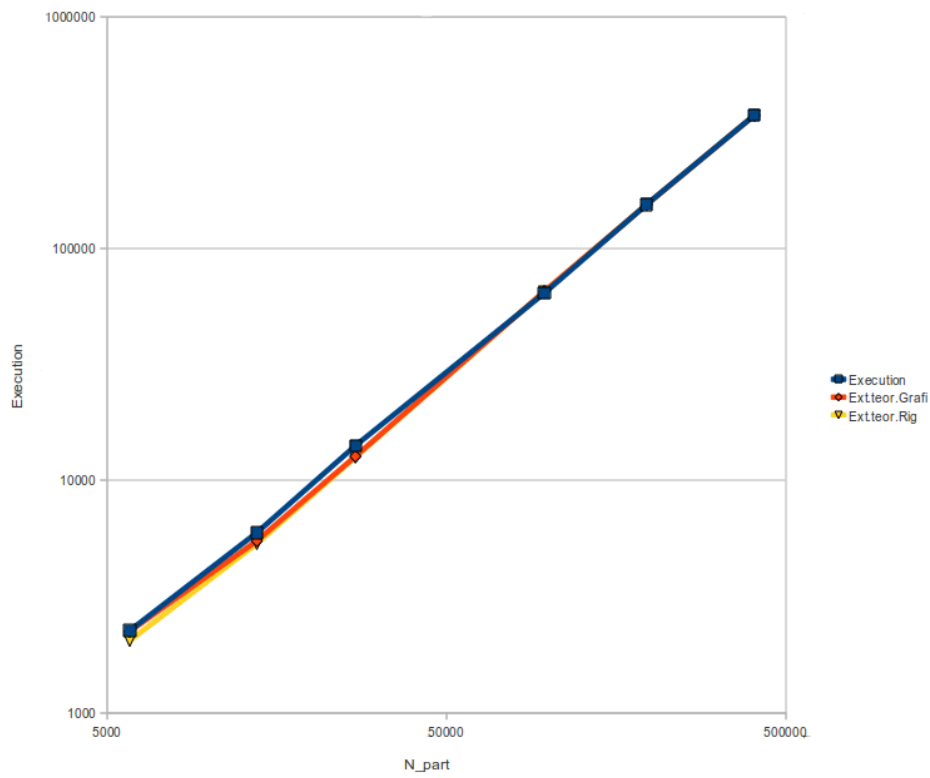


Figure 9.10: Computational time in seconds, using as serial fraction the one obtained with the graphic method and the one obtained with the rigorous method. The differences are negligible for our purposes.

tion time of a simulation, we want to know how long it would take to do it again at higher resolution (larger number of particles) even when more CPUs are available. The duration of the time steps greatly varies as the evolution proceeds, the density increases, or as soon as stars are formed. As a general rule, to follow correctly the evolution of the physical system the time step must continuously change (in most cases it must become very short).

Time steps To correctly estimate the overall time of execution, I need to know as the length of the individual steps as the simulation proceeds. The knowledge of the time steps is essential to estimate the CPU time required to complete a set of simulations. To this end, in several simulations the time steps were monitored and stored. The result is that although the time step greatly varies during the early stages of a simulation, once a sufficiently long time has elapsed it becomes nearly constant (and higher than the minimum required) thus allowing us to estimate the total computation time. For the sake of illustration, in the Figure 9.11 we present a typical simulation.

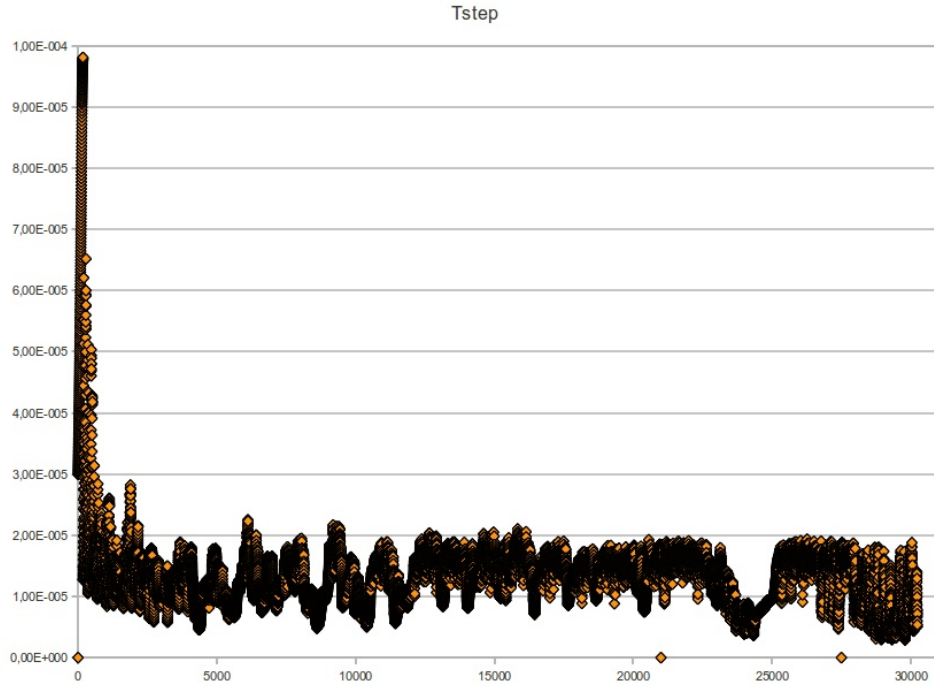


Figure 9.11: Time steps duration (in units of the Hubble time) during a typical run.

The time step, after an initial phase, remains fairly constant throughout the entire duration of the simulation. This fact allows us to estimate the duration of the entire run by measuring only a (small) appropriate fraction

of it.

9.7 Study of the MPI communications

The other component of a parallel code, along with the computation, are the MPI communications that, while necessary, can reduce the ideal performance, so that any analysis of the performance of a parallel code cannot ignore the time spent in communications.

9.7.1 Factors Affecting MPI Performance

The factors which can affect the performance of a MPI application are numerous, complex and interrelated. Because of this, generalizing about an application's performance is usually very difficult. Most of the important factors are briefly described below, and a number of them covered in more detail later. Others even of great importance are not discussed, as they go far beyond the scope of this thesis.

- Platform / Architecture Related:
 - CPU - clock speed, number of CPUs
 - Memory subsystem - memory and cache configuration, memory-cache-cpu bandwidth, memory copy bandwidth
 - Network adapters - type, latency and bandwidth characteristics
 - Operating system characteristics - many
- Network Related:
 - Hardware - network adapters, switch and intermediate hardware
 - Protocols used
 - Network tuning options
 - Network contention / saturation
- Application Related:
 - Algorithm efficiency and scalability
 - Communication to computation ratios (granularity)
 - Load balance
 - Memory usage patterns
 - I/O
 - Message size used
 - Types of MPI routines used - blocking, non-blocking, point-to-point, collective communications
 - ...many more

- MPI Implementation Related:

- Message buffering

- Message passing protocols - eager, rendezvous, other

- Sender-Receiver synchronization, interrupt

- Routine internals - efficiency of algorithm used to implement a given routine

Only some of these points are of direct interest to us and directly related to the way a parallel code is written, in particular: load imbalance, algorithm efficiency and scalability, communication to computation ratios (granularity), type of MPI routines used (used - blocking, non-blocking, point-to-point, collective communications) and message size used. I remember that the total time of a communication amounts to a latency time, the same for every act of communication, and a real-time communication that depends on the amount of data exchanged. Three cases are examined:

- A cosmological simulation, with DM and gas particles thrown on a growing number of processors;
- A cosmological simulation, with DM, gas and stars particles in an advanced stage of evolution launched on a growing number of processors;
- A galaxy simulation in physical coordinates.

The simulations were run only for a small number of time steps, due to the limited amount of CPU time available. However, we tried to have a sufficient number of time steps to make meaningful measures (10-500). The data are as follows:

Table 9.11: Percentage of MPI communication for three simulation at varying of the number of adopted CPUs.

N proc.	Sim. 18	Sim. 46	HDMM
2	0.97%	3.31%	1.76%
4	4.75%	2.69%	2.70%
8	3.64%	2.06%	2.79%
16	9.99%	19.50%	4.35%
32	10.32%	18.32%	6.40%
64	19.36%	22.61%	9.31%
128	35.73%	24.73%	11.94%

Figure 9.12 shows the same data in graphical form to highlight a possible trend.

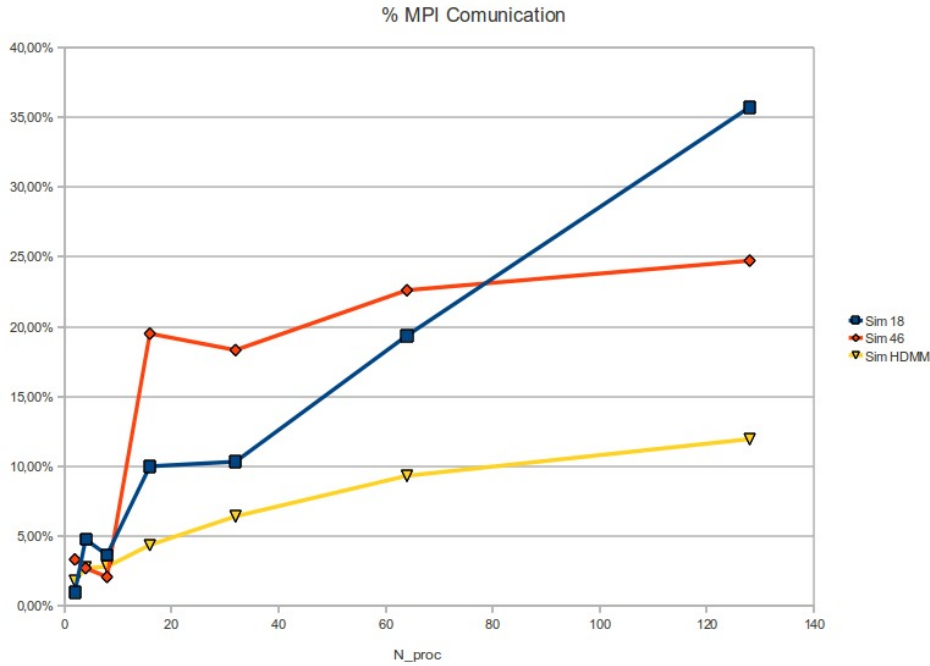


Figure 9.12: Percentage of MPI communication for three simulation at varying of the number of adopted CPUs.

The communications have a fairly regular upward trend. It is worth recalling that we are performing tests at a constant problem size. Therefore, the MPI time can increase due to an increase of data communicated between the processors or because of the latency. One may notice the excessive weight of the communications in the case of 18^3 , caused by the use of too high a number of processors for a small simulations. However, we are not interested to the values of time but only to their absolute trends. The ratio computation to communication time varies with the type of processor, the network subsystem in use used, and the number of users running simultaneously on the same machine. Therefore the details are irrelevant here. It is more useful to see which component of the communications dominates and how it scales with the main parameters of the simulation.

We now compare the data obtained using Trace Analyzer on MONSTER, see Figure 9.13.

In this case, the communications may dominate the overall time more easily. In the graph for clarity I have not reported the case with 32 proc. The low bandwidth, but especially the high latency of the network, are the factors explaining this behavior. The same simulation launched on SP6 launched, even with a higher number of processors, behaves differently thanks to its more efficient network infrastructure. To understand what drives the MPI

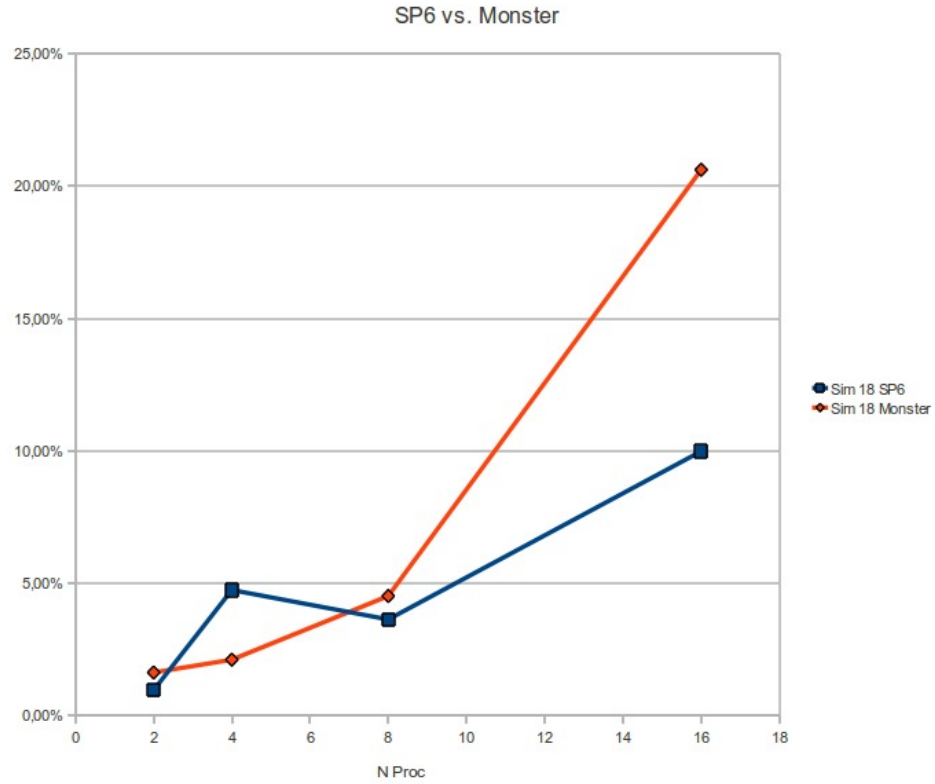


Figure 9.13: Different weight in MPI communications in the same simulation on different machines at varying the number of adopted CPUs. The difference is in the performances of the high performance net subsystem on SP6.

communication time, we can look at the amount of bytes transmitted during the simulations. The software SCALASCA provides this information (split in routines, processors etc.). The results can be seen in Figure 9.14⁷ of the three usual simulations (18^3 , 46^3 , $HDMM$). One can clearly note the increase of bytes sent by the simulations, run under identical conditions, except for the number of processors used.

It is worth noting that while the number of processors, physical communications, transmitted by a single processor, are expected to decrease at decreasing the size of the physical of the sub system under investigation, the total amount of data transmitted increases because, although each processor is working on a sub-domain and so should communicate less data, each

⁷Similar data can be extracted with the Trace Analyzer on the set of simulations run on MONSTER

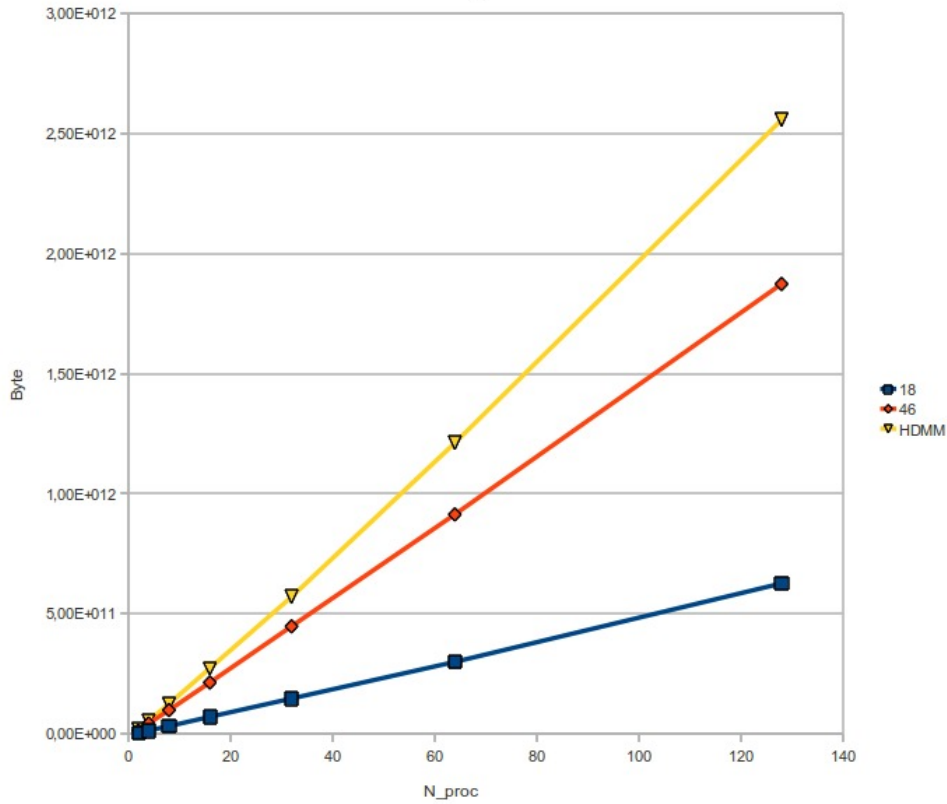


Figure 9.14: Total number of bytes exchanged between processors at varying the number of CPUs for three simulations.

sub-domain must, for the code characteristics, construct and communicate all physical variables considered and the whole tree-code, without knowing which part of it really need the processor with which it is sharing data. This eventually explain the observed increase in the total amount of transmitted with the number of CPUs. This simple approach is acceptable when the run is made a few small processors: in terms of performance it takes longer to cut the tree, transforming it into a essential tree than transmitting the whole body of data. The situation changes when a large number of processors is at work. Unnecessary transmission of data is no longer acceptable and need to cured.

We can also get from SCALASCA more details about the real nature of these communications. SCALASCA breaks the MPI communication among different components. The Figure 9.15 shows the different components in which SCALASCA splits the MPI communication time. The displayed times (simulation 46^3) are the total MPI time, the total Synchronization time, the total communication time (in turn divided into its component P2P and

collective), the initialization time, and finally the time spent to execute the same MPI routines (the cost of the call).

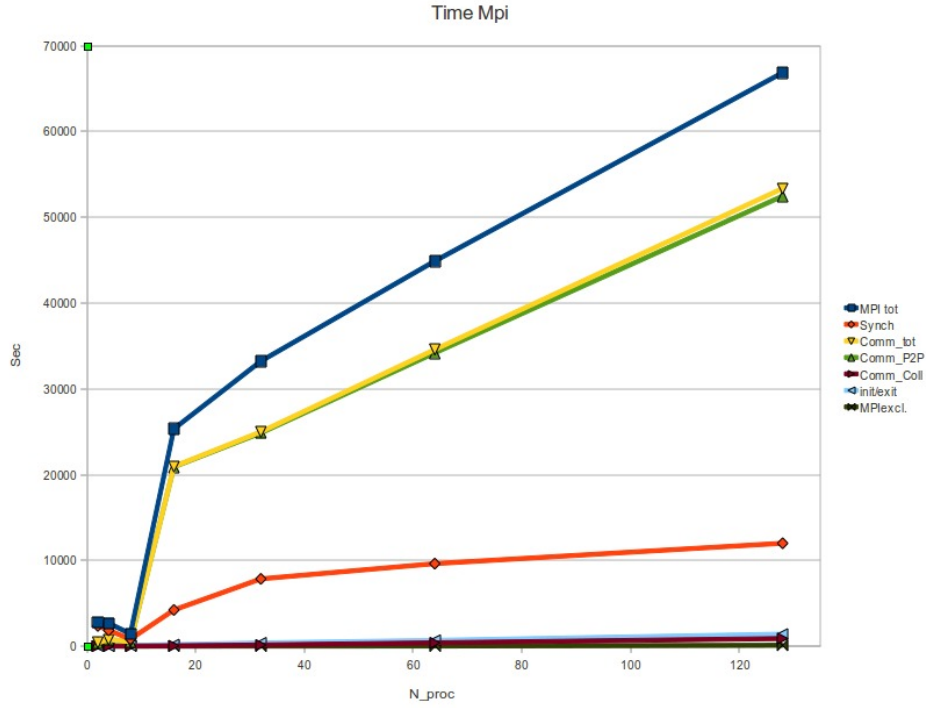


Figure 9.15: Ripartition of time spent MPI in communications: total (blue), synchronizations (orange), total communications divided into point to point (green) and collective (purple), the time for initialize the MPI execution environment (cyan) and the temporal cost of the calls to the MPI routines (brown and negligible). The dominant components are the P2P communications and the synchronizations.

The two principal components are point-to-point (P2P) communications and synchronization. But it is not possible to neglect the collective communications with respect to the P2P communications.

P2P is the communication of data between processors⁸, while the collective operations include the reduction or global communications (all_reduce, All_gather). P2P communications grow linearly for the reasons explained, while the collective operations grow quadratically and then, even though negligible on a few processors, become important to many⁹.

We can see from Figure 9.16 as the ratio P2P/Collective decreases, and therefore the collective communications become comparable with P2P.

⁸Usually the communications are between a processor and another

⁹One or two orders of magnitude at least

Table 9.12: Total number of trasmitted bytes among the processors for three simulations(with 2×18^3 , 2×46^3 particles and the HDMM simulation) divided into P2P and collective communications.

Nproc	18 P2P	18 Coll	46 P2P	46 Coll	HDMM P2P	HDMM Coll
2	$4.64 \cdot 10^9$	$7.98 \cdot 10^5$	$1.40 \cdot 10^{10}$	$1.35 \cdot 10^5$	$1.79 \cdot 10^{10}$	$4.57 \cdot 10^5$
4	$1.41 \cdot 10^{10}$	$3.79 \cdot 10^6$	$4.30 \cdot 10^{10}$	$6.93 \cdot 10^5$	$5.40 \cdot 10^{10}$	$2.14 \cdot 10^6$
8	$3.25 \cdot 10^{10}$	$1.57 \cdot 10^7$	$9.88 \cdot 10^{10}$	$2.96 \cdot 10^6$	$1.25 \cdot 10^{11}$	$9.02 \cdot 10^6$
16	$7.11 \cdot 10^{10}$	$6.34 \cdot 10^7$	$2.14 \cdot 10^{11}$	$1.10 \cdot 10^7$	$2.73 \cdot 10^{11}$	$3.57 \cdot 10^7$
32	$1.47 \cdot 10^{11}$	$2.47 \cdot 10^8$	$4.48 \cdot 10^{11}$	$4.31 \cdot 10^7$	$5.71 \cdot 10^{11}$	$1.41 \cdot 10^8$
64	$2.99 \cdot 10^{11}$	$9.69 \cdot 10^8$	$9.14 \cdot 10^{11}$	$1.80 \cdot 10^8$	$1.21 \cdot 10^{12}$	$5.54 \cdot 10^8$
128	$6.23 \cdot 10^{11}$	$3.85 \cdot 10^9$	$1.87 \cdot 10^{12}$	$6.67 \cdot 10^8$	$2.56 \cdot 10^{12}$	$2.18 \cdot 10^9$

SCALASCA includes in the collective communications the "WaitAll" that are places where the computational load imbalance manifests itself. The "wait-all" may have less weight in a more balanced parallel execution. The rest of the collective communications are difficult to reduce because essential to carry out calculations unless a drastic, profound revision of the communication strategies is applied.

In the Figures 9.17 and 9.18 we can see the relative importance of the dominant components of the MPI communication at varying the number of particles and processors.

9.7.2 Load imbalance

As already mentioned, in general the message passing time is not the main problem, which is instead usually due to processes waiting for other processes. Good parallel algorithms try to minimize a processor's idle time with proper load balancing and efficient coordination of processor computation and communication. The load imbalance becomes increasingly important as the number of particles grow. This is because small "errors" in the division of the domain lead to marked differences due to the high number of particles, see Figure 9.19.

A more difficult task is to interpret the behavior of the "load imbalance" with the number of processors, shown in Figure 9.20 for the case of the "46" simulation.

Here, the load imbalance is the first small and decreasing, then after a big break, again 8 / 16 processors, starts to decrease regularly. This result undoubtedly indicates the difficulty of the routine with the domain decomposition (ORB) to create perfectly uniform even simple computational domains. In the considered case the simulation started at a very high redshift from a substantially uniform background, so splitting the domain among the particles was difficult and uncertain.

On MONSTER, the measurements were made with Trace-Analyzer and

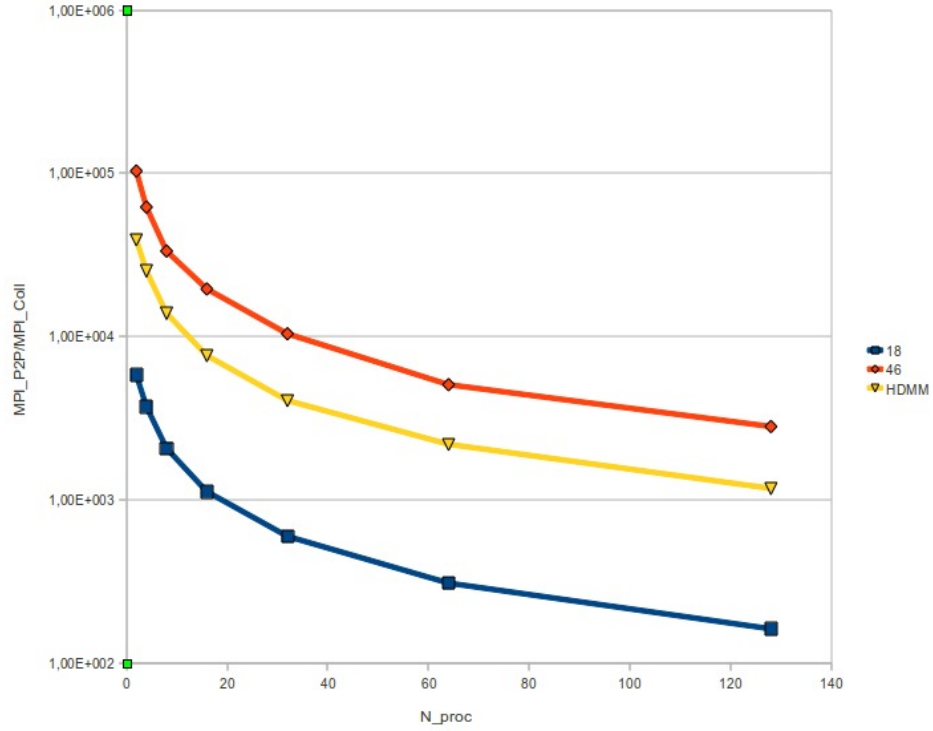


Figure 9.16: Ratio between P2P and collective communications. the Y axis is logarithmic. Increasing the number of processors, in the small simulations (cyan) the collective communications are not negligible, and using a large number of CPUs (256 or more) this contribution can become important.

Collector. The data obtained with the Trace-Analyzer is rather useful for the study of load balance and communication MPI. The times spent in computing and communications in MPI are divided and easily accessible tanks to an efficient graphical interface, which allows us to get in a few seconds the main features of the communications. The data is split into time spent in the user code (the calculations) and time spent in communications. Both entries are provided for individual processors. So it is very easy to detect imbalance problems at the expense of a single processor. In addition, the MPI communication can be divided according to type and again, each type can be divided per processors. The data confirm the measurements made with SCALASCA. These measurements add details (for now not used in the optimization process) on the behavior of MPI communications, and highlight the limits of the network system of MONSTER (less powerful than similar clusters of CINECA). Figure 9.21 shows some data taken from the simulation with 18^3 particles, launched on 16 processors.

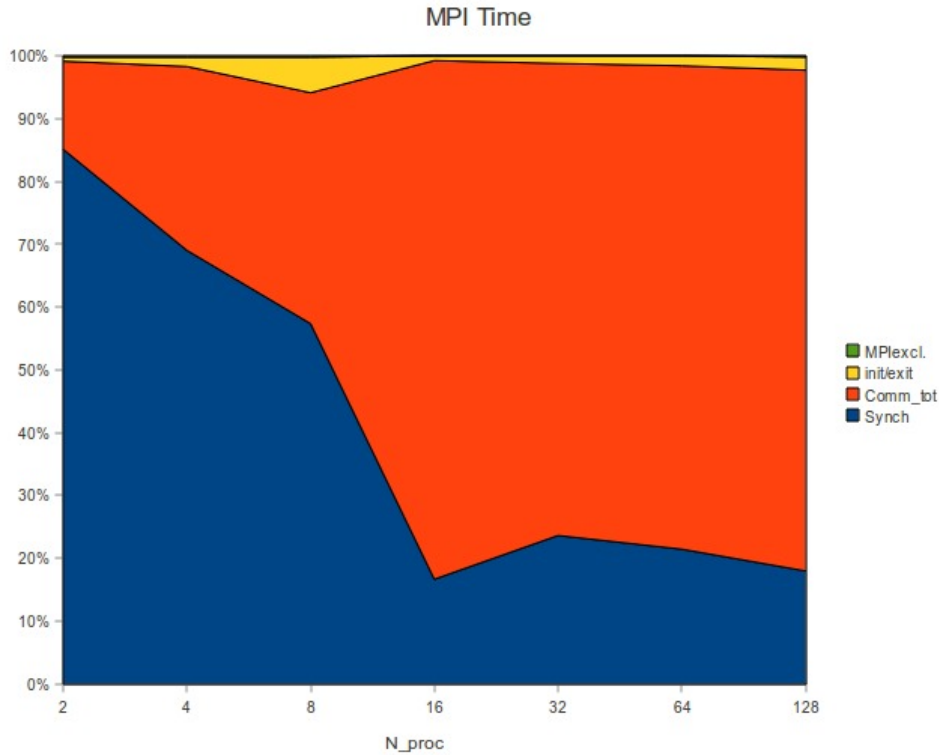


Figure 9.17: Variation of the fractional weight of the various components of MPI communications with the number of CPUs. Orange: total communications; blue: synchronizations.

Easily one can estimate the load imbalance divided between calculation and communication as shown in Figures 9.22 and 9.23

It can be very useful for displaying the type of communication, see Figure 9.24, where one can see the importance of barriers in the communication process.

The time spent at these barriers gives us an idea of the time lost by the code due to not perfect work sharing among processors (load imbalance). A domain decomposition should take the variability of the system during the evolution into account, and, therefore, must be dynamic. Our code redefines the decomposition based on the computation time of the previous time step, and tries to balance the load among all the processors. Obviously this cannot secure perfect balancing, because the calculation time at the current time step is not known. One may somehow cure this by periodically synchronizing the processors with the introduction of barriers. In brief, the domain decomposition strategy is important as the serial optimization. In addition to this, we can create custom CHART, with various combinations of data.

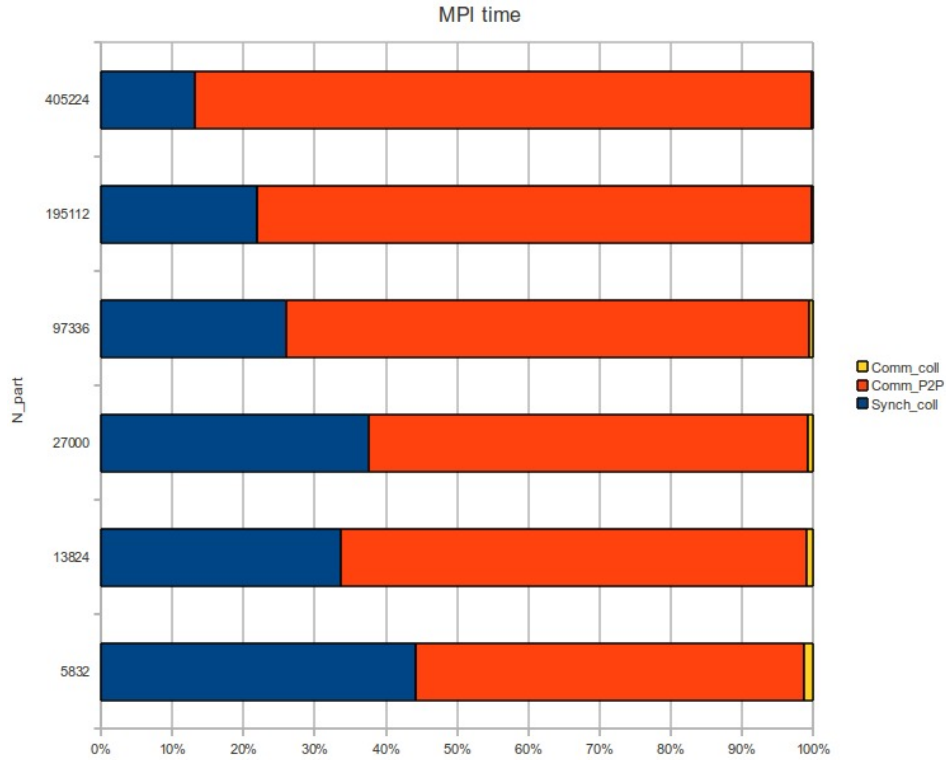


Figure 9.18: Variation of the fractional weight of the various components of MPI communications with the number of particles. Orange: P2P; Yellow: collective; blue: synchronizations.

With trace-analyzer we can explore, for example, average and total times of communication and calculation, the total number of bytes transmitted and received or band-width between pairs of processors. All this is achieved very easily and provides a set of additional information on the detailed behavior of the code.

9.7.3 MPI communication time

The time spent in MPI communication consists of two parts: communication time itself, and latency. The latency is a fixed amount of time that depends on the hardware of the machine and the MPI implementations. Because, every act of communication implies this fixed time, too few communication may not be so advantageous at the end because in this way the communication time is dominated by latency. In the following, we want to verify, under normal conditions of use, how close our code is to this critical point. To work properly we must have that the average time of a single communi-

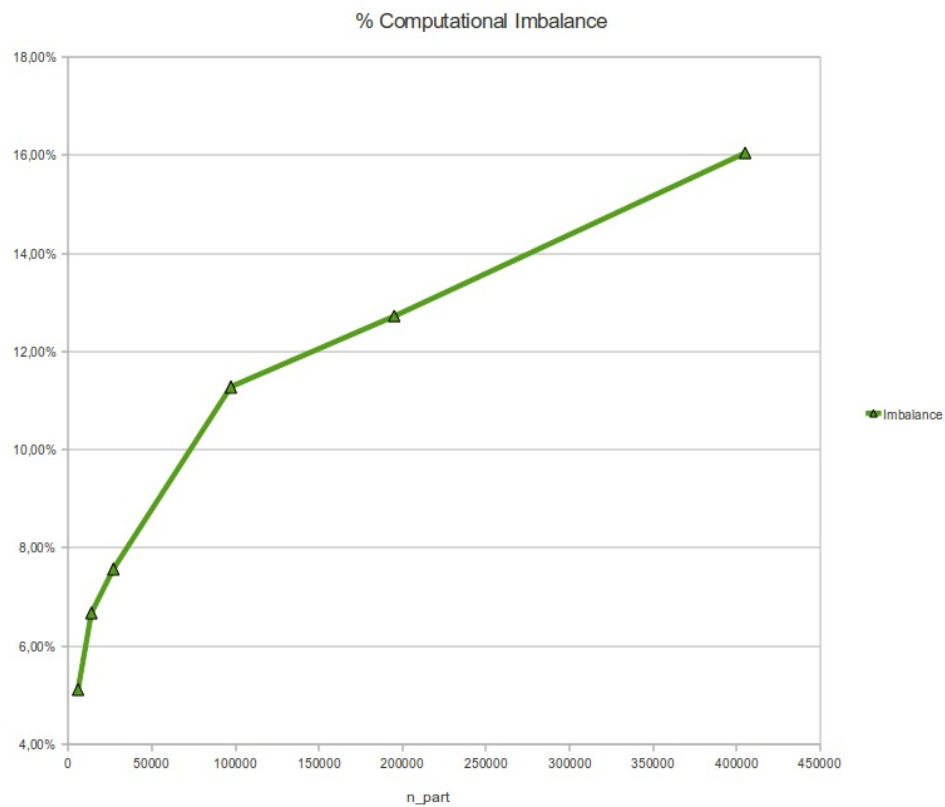


Figure 9.19: Load imbalance as fraction of the total computational time at varying the number of used particles on a constant number of CPUs (32).

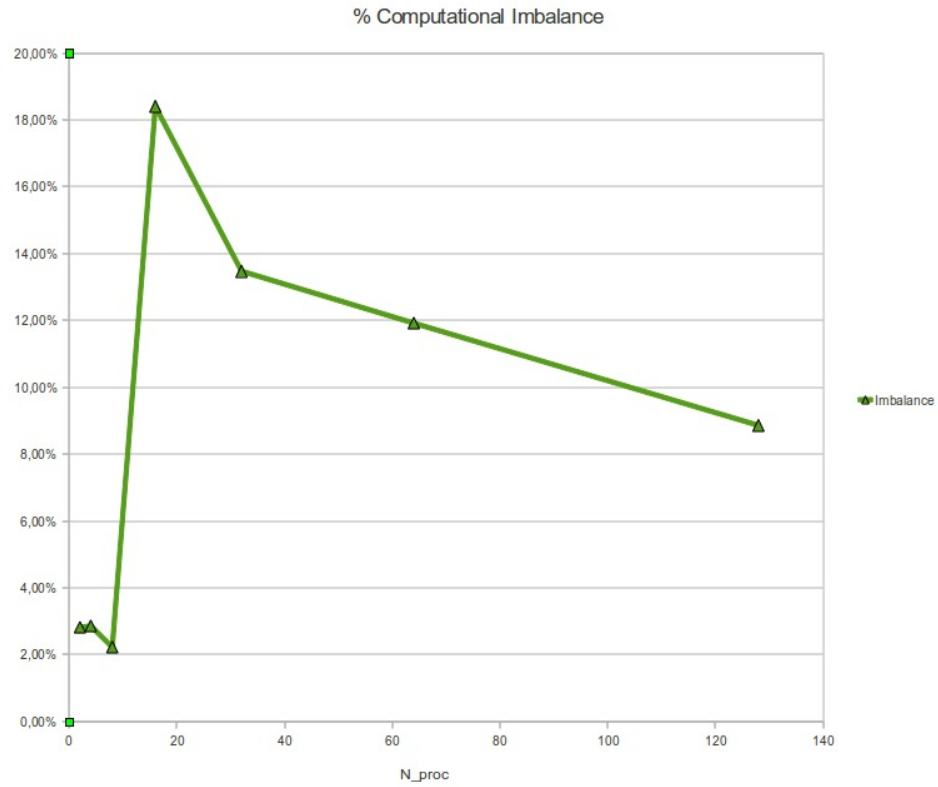


Figure 9.20: Load imbalance as fraction of the total computational time at varying the number of used CPUs. The simulation is the 18^3 .

Flat Profile Load Balance Call Tree Call Graph						
Group All_Processes						
Name	△	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes						
User_Code		5.9e+3 s		7.11e+3 s	16	369 s
Group MPI		1.22e+3 s		1.22e+3 s	24709636	49.2e-6 s

Figure 9.21: Ratio computation/communications as displayed with Trace Analyzer, for the 18^3 simulation on MONSTER with 16 CPUs.

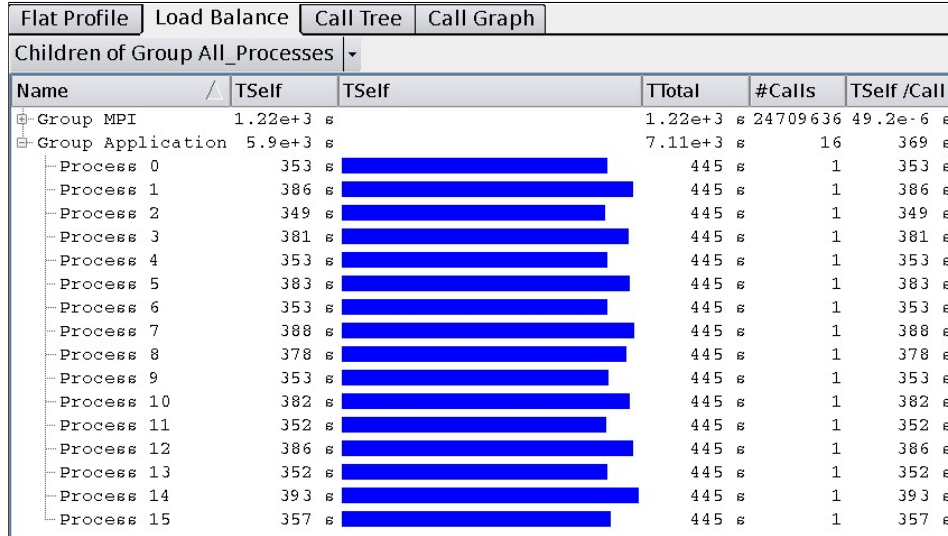


Figure 9.22: Load imbalance in the computation. Same simulation as in 9.21

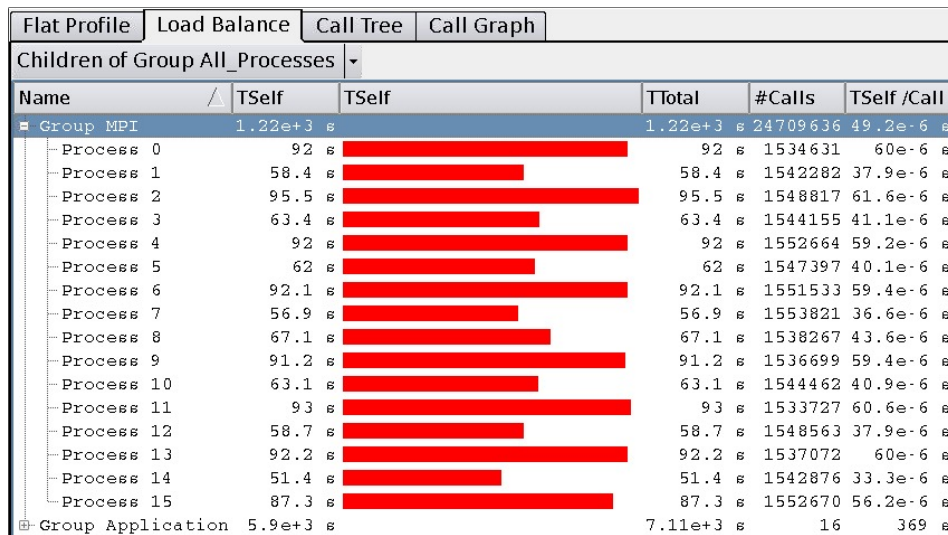


Figure 9.23: Load imbalance in the communications. Same simulation as in 9.21

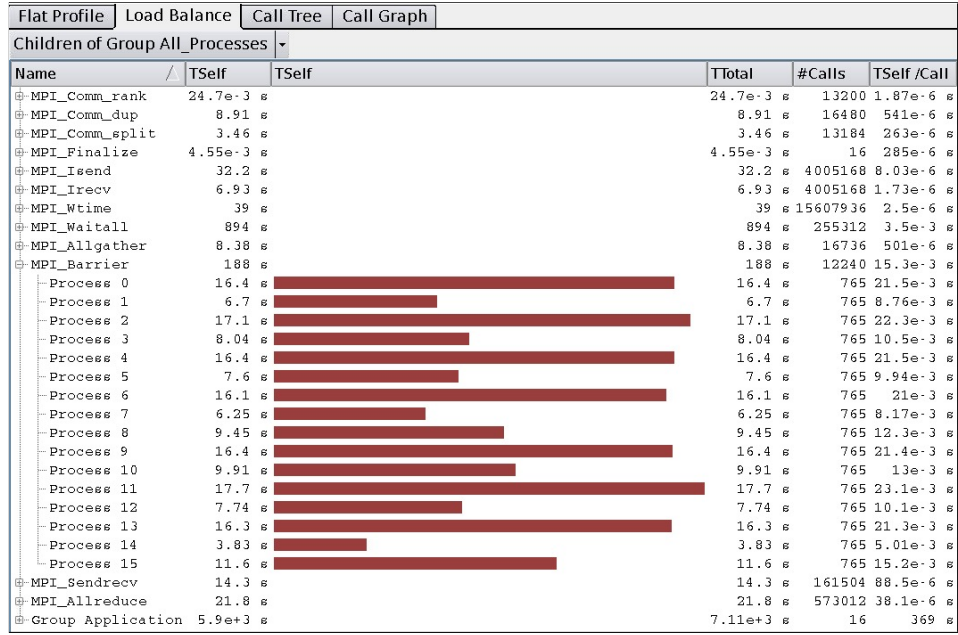


Figure 9.24: As in 9.23 but with communications divided by type. Total times spent in Barriers are shown.

cation is much longer than the latency. To this aim, we get from SCALASCA and trace-analyzer the data of interest to estimate the average total time of a communication and make an experiment to estimate the latency of the computers in use. Finding the average communication time is trivial. SCALASCA and trace-analyzer give the total number of communications sent and the number of bytes transmitted, the ratios of the two is the quantity we need. Since we an estimate of the time made under the worst working conditions, using a rather small simulation (18^3) launched on 16 processors of SP6 and MONSTER (for very different times) we get the data we need. The MPI communication in our evolutionary code takes place in various routines, but the most important one is ORB, in which the code transmits the full array (ie for all particles) of variables representing the physical quantities. The size of *individual* communications decreases as the number of processors. Increasing the number of processors and keeping constant the size of the problem, the amount of data exchanged in a single message decreases linearly and so the communication time, so that the latency (in principle) becomes more and more important. The maximum effect occurs when a a small simulation is launched on a high number of processors. In our simulations, the exchange between processors throughout the tree-code is tracked and with these data we calculate the values listed in Table 9.13.

Table 9.13: Time spent in communications, number of communications, and estimation of the average duration of a single communication.

Machine	Time	Count	Single MPI time
SP6	3141.97	$1.34 \cdot 10^7$	$2.35 \cdot 10^{-4}$
MONSTER	93.2	$1.74 \cdot 10^3$	$5.36 \cdot 10^{-3}$

9.7.4 Latency

We must now estimate the latency, i.e. by definition the time to send a message of size zero. The latency time is due to a number of factors, namely hardware, software, state of the network, average use, congestion or other. We need to derive the order of magnitude of the latency and compare it with the average time of message transmission, to verify that the transmission time in the test conditions is not dominated by latency. To this aim, we made a series of measurements, creating communications of known size and measuring the transmission time with “MPI_Wtime ()”. The data were obtained using SP6 and MONSTER. The results are reported in Table 9.14. For MONSTER, we also show the same results in graphical form to highlight the trend of the transmission times at growing amounts of transmitted bytes, see Figure 9.25.

Table 9.14: Comparison of the MPI transmission times (in μsec) at varying of the number of bytes transmitted, on SP6 and on MONSTER.

Bytes	SP6	MONSTER
1	4.08	125.35
2	4.70	124.89
4	5.32	124.93
8	17.17	125.33
16	5.01	125.39
32	5.01	125.93
64	4.05	129.92
128	12.87	139.67
256	5.96	156.02
512	5.96	181.22
1024	7.15	226.28
2048	8.11	275.20
4096	10.01	393.35
8192	14.06	624.90
16384	24.08	1063.11
32768	42.92	1997.48

We must measure time needed to exchange messages between processors residing in different nodes, otherwise we would measure the RAM latency of the shared node. MPI in the same node does not use the network, but communicates through the shared memory. This can be seen clearly from the Figure 9.25 obtained with the trace-analyzer chart, comparing the time on a single node and two nodes (the measurement is made on MONSTER, whose nodes have 8 processors).

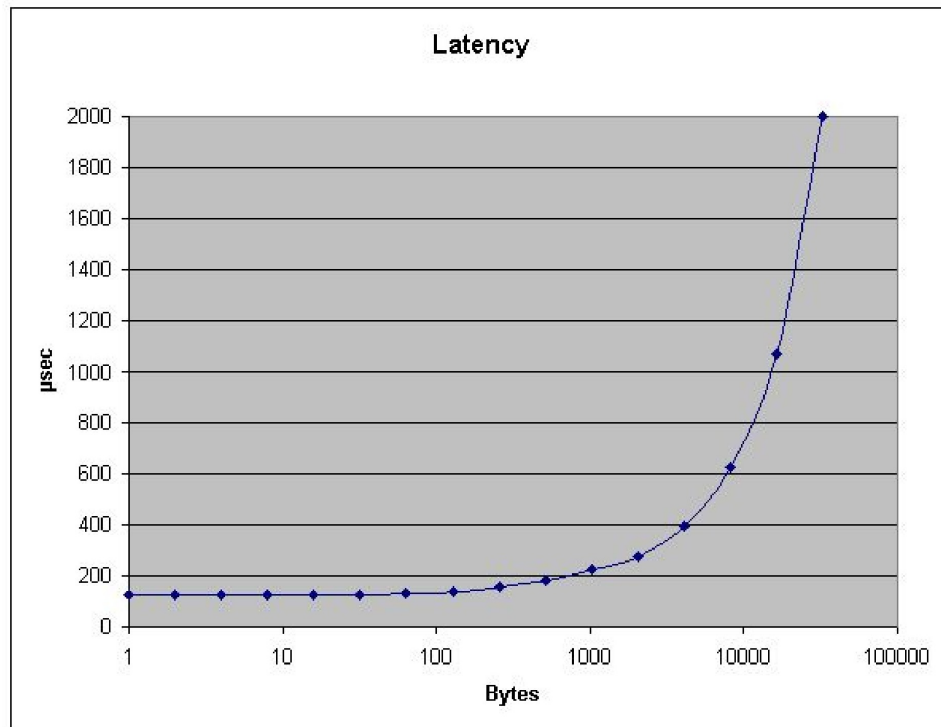


Figure 9.25: MONSTER data from Tab. 9.14. When the dimensions of the transmitted data are too small the total time of communication is essentially due to the latency.

The measurements are made when MONSTER has no other active processes so that they are not disturbed by possible interfering jobs. SP6 is part of a large network with many users at the same time; this explain why the output is more irregular. From the graph we note that at the beginning the communication time does not vary increasing the size of messages in MONSTER (messages of a few bytes are sent). This means that latency is dominating. At decreasing the number of transmitted bytes the transmission time decreases (latency + transmission time itself) to the point that only latency survives. So the minimum time is a good estimate of the latency: the latency of MONSTER is about 125μ sec, while that of SP6 is about $5 - 10\mu$

sec.

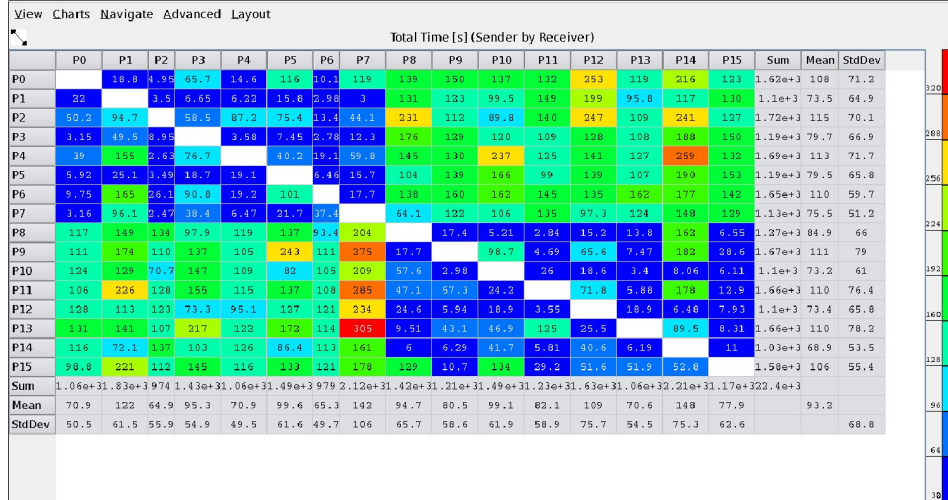


Figure 9.26: Trace Analyzer can easily highlight the differences in the communication times among processors. Here the structure of the MONSTER cluster is made evident (8 processors per node). The communication times between processors belonging to the same node are much lower (blue) then those relative to processors belonging to different nodes.

In contrast, if the transmission time of each message is much longer than latency, this latter can be neglected. Now knowing the latency and the average length of a message during the standard use of our code, we can soon know if we are in the area where latency is important or, even worse, dominates the communication time. We summarize in Table 9.15 the results we have obtained:

Table 9.15: Comparison among the average times for a single communication and the latency, for MONSTER and SP6.

Machine	Single MPI time	Communication Latency
SP6	$2.35 \cdot 10^{-4}$	$5 \cdot 10 \cdot 10^{-6}$
MONSTER	$5.36 \cdot 10^{-3}$	$1.25 \cdot 10^{-4}$

Under normal conditions (number of particles per processor not too small) latency can be neglected. There is another useful result: since our experiments use $2 \times 46^3 \approx 10^4$ particles on 16 processors, we can assume this is as the **minimum number of particles** per processor to be used to avoid latency problems.

Finally, let me simply mention that small local computers like BUD and

HAL9002, on which codes have been freely install and tested, have been used with no particular optimization strategy.

9.8 Problems of cache misses

Now we know a lot about the behavior of EVOL: we know in which routines it spends most of the time, what is its efficiency in the use of processors, how it scales with their number, how does the particle number impact on the communications, etc. However, its performance can still be optimized further. The question to clarify is: How to decide whether a routine is wasting time? In general, if we discover that routines, among those consuming the majority of time, also have many cache misses, these are the first candidates for optimization to look at, whereas the others (non-cache missed) require a deep revision of the underlying algorithm. Memory hierarchy is a problem thus far largely neglected, because there is much dependence on the hardware in use and the impossibility of make measurements on each platform. Even with the limitations that this analysis shows, knowing where EVOL makes a bad use of the cache in a particular architecture, could be considered a result that could be extended to other platforms.

The software used is Cachegrind, which is part of the package Valgrind. Even in this case I used to consult the output file in a graphical front end, namely kcachegrind. Only the serial version of the code has been tested on the "domestic" desktops. The results are still significant and highlight the routines that fail in the use of the memory hierarchy. I am confident that the behavior of different architectures is similar. Improving the use of the cache depends on how the data is stored and accessed, then the data structure, the length and the order in which arrays are handled as well as use of variables temporarily avoid constant recalculation of formulas can improve both the spatial and temporal locality.

In the Figure 9.27 is an example the output kcachegrind. One can see the routines more involved in the problem: looking at the routine `near.F90`, for example, one can note the shared cache-miss in `near` and selecting a calling subroutines (e.g. `period`) one can get the detail of the source.

Using this software one easily understand, even before touching a single line of code, if there are better ways of using the memory. The changes to make in these cases are usually very specific, but simple and quick to apply.

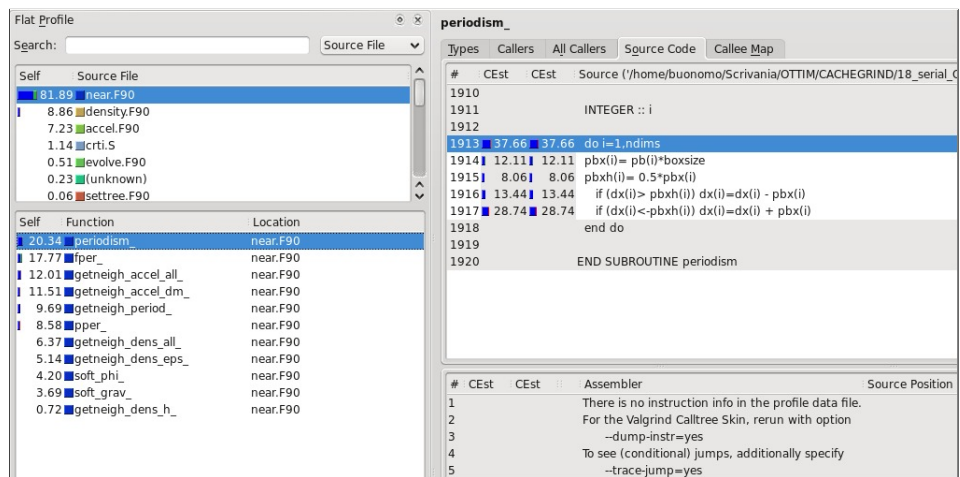


Figure 9.27: Cachegrind on the 18^3 particles simulation on BUD (on a single processor). The routine `periodism` is highlighted.

Chapter 10

Optimizing EvOL

In this chapter I describe the optimizations I introduced in EvOL. They can be grouped as follows:

- Preprocessor and Makefile technique
- Routine Periodism (several optimizations)
- Routines `fper` and `pper`
- Routines `soft_phi` and `soft_grav`

10.1 Compiler Flags

The first, obvious attempt to improve the performance of the code is to make proper use of the possibilities offered by the compiler. Usually a compiler of middle/high class (-O2/-O3) provides good performance. There are also additional options that can be activated in the long list of options usually disabled with standard optimization levels (see the appendix). Many tests have been made looking for the best options. In particular, we tested the inter-procedural optimization (IPA) and unroll loops when not enabled by default. The improvements are always very modest given that the code was already compiled properly on all machines in use. I tried also different paths, always exploiting the possibilities offered by self-optimizing of the compiler. There is the possibility of automatic optimization whose name changes for different compilers. The procedure is always the same. The process takes place in two phases: In the first phase it collects the necessary data, in the second one, the compiler chooses the options that turn out most useful for reducing the overall execution time. The procedure is fully automatic, but the final results are not dissimilar from those achieved with activating very simple flags. To my experience I consider this kind of guided optimization absolutely useless.

10.2 Preprocessor and Makefile technique

The compiler cannot be very useful. As already discussed, even writing the source code can also inhibit many of the optimizations that the compiler would be able to do. In addition, the compiler cannot remove all the causes of a slowdown, which are due to a poor design of the code so that restructuring the whole code is often mandatory. A typical cause of significant loss of time is the improper or excessive use of `if-then-else`.

Our evolutionary code can operate in several modes that are selected by the user. The choices are made via a text file (`data.ini`) that is read by EVOL. Looking at the way in which this technique is implemented we noticed that it was very inefficient. In fact the individual options were tested many times during the run: the code was full of `if-then-else` statements. Typical options of old `data.ini` were:

```
...
T    ! do_redshift
T    ! ind_tstep
F    ! fixed
F    ! nocs
F    ! nograv
T    ! usquad
F    ! MWpotential
T    ! comov
T    ! period
T    ! SPHcalc
F    ! heqeeps
T    ! ghterms
F    ! do_entr
F    ! turb
T    ! do_cool
T    ! do_cond
T    ! do_star
...
```

For instance, considering the periodism, a variable was set to `TRUE` and then throughout the simulation, the code had many lines like this:

```
IF (period) THEN
CALL periodism
ENDIF
```

Of course all this was very time consuming. The variable `period` does not change its value (`TRUE` or `FALSE`) for the duration of the simulation and then the continuous control checks are not necessary. On the other hand,

the flow of execution of the program should be different depending on the value of that variable. We must then use a technique that allows to make the CALL periodism if appropriate, without any control. Fortunately there is a simple way to achieve this. We must, specialize, before running the code, to do or not do CALL periodism if appropriate. In essence, we must generate a different code (specialized) for each case desired. In this case, the pre-processor comes to rescue, may generate a conditional compilation, editing the source (deleting or replacing parts) depending on the options passed by the user.

After verifying that the use of the pre-compiler could be done without major complications or incompatibility of different platforms (to maintain the absolute portability of the code), the required changes in the code have been made. Basically the `if` statement in the source code is replaced by new statements that at the pre-compilation stage define appropriate macros. For example, in the case above, the source becomes:

```
#ifdef _PERBC_
CALL periodism
#endif
```

If the macro `_PERBC_` is defined, then the line will be present in the pre-compiled source `CALL periodism`, it is compiled, and later, during the run it will make the call request, otherwise the row is simply deleted. The macros can be passed in the source or during pre-compilation, as an option. The number of options in our code is high, and the two-steps (after pre-compilation and compilation) are made by hand, adding all the necessary options became expensive and cause of errors. In addition, the specific mode of passage of these parameters varies slightly from compiler to compiler.

All options in the parameter file have been translated into `data.ini` pre-processor directives and go through the Makefile. A series of new options has been added. In particular, the possibility of compiling the code for testing single-precision and fast simulations. In single precision, the code becomes almost 2 times faster. However, it is not wise to use this mode for scientific production. In addition, a specific environment variable controlling the Makefile is able to identify the machine on which the compilation is underway and uses, the syntax and the compiler in the correct way. An example of Makefile is as follows:

```
...
OPT +=-D_FIXEPS_
OPT +=-D_COMOVING_
OPT +=-D_PERBC_
OPT +=-D_CALCSPH_
#OPT +=-D_HEQEPS_
```

```

OPT +=-D_REALSIZE_8_#_REALSIZE_4_
...
#
# MONSTER
#
ifeq ($(HOSTNAME), monster.astro.unipd.it)
FC = ifort
MPFC =mpiifort
FLAGS =
OLEVEL =-O3 -ip -ipo -unroll -pc32
#DEBUG = -O0 -g -traceback -fp-stack-check -check bounds -fpe0
LIBS =
TEMPFILE = -save-temps
KNOWN_SYSTEM=yes
endif
...
EXEC = EvoL.exe
OBJS = modules.o \
accel.o ahead.o alloc.o ann.o cooling.o create.o density.o evolve.o \
init.o interpol.o main.o near.o orb.o output.o ran1.o reallocate.o settree.o sp
all: mpi3sph

mpi3sph: checksystem $(OBJS)
$(MPFC) $(OLEVEL) -o $(EXEC) $(DEBUG) $(LIBS) $(OBJS)

clean:
rm -f $(OBJS) $(EXEC) *~ *.vo *.mod *.lst *.s *.i90 *.f95 *.f90 F*.f90 core

tar:
tar czvf EvoL_III.tar.gz *.F90 *.ini Makefile sub_*

zip:
zip EvoL_III.zip *.F90 *.ini Makefile sub_*

.SUFFIXES: .o .F90

.F90.o:
$(MPFC) $(OPT) $(FLAGS) $(OLEVEL) $(DEBUG) $(TEMPFILE) -c $<

checksystem:
ifeq ($(KNOWN_SYSTEM), yes)
@echo ""
@echo "Compiling EvoL for $(HOSTNAME)"
@echo ""

```

```

    else
@echo " make: ERROR: value of SYSTEM = $(HOSTNAME) not recognised..."
quit
    endif

```

There was no need of sophisticated profiling analyses to understand that this change in the code structure is much more advantageous on any kind of computer.

Table 10.1: Improvements of EvoL with respect to the originary code.

Version	Orig. ver. -O0	Preproc.-O3	Orig. ver. -O3	Preproc.-O3
EvoL	22350.47	3926.83	8240.22	1412.45

In general the overall execution time is about a factor of 6 faster than before (see table 10.1. The effective speed-up however slightly depends on the computer and simulation.

10.3 Periodism

Let us consider a small simulation, which can be calculated on a short time on local machines, and start to optimize some selected routines. For this type of optimization we do not need to communicate, but simply to profile a serial code. In the Figure 10.2 we see the code profiling with `gprof` obtained on BUD for a test case with 2×18^3 particles leaving the code running on 1 processor for 300 steps.

Profiling provides the percentage of total time spent by individual routines, including also the total time spent in routines called by them. The times “self” rather indicate the time spent exclusively in the procedure. The two different orderings are shown in Fig. 10.3. One can soon recognize the routines best suited to optimization. The code has been compiled -O0, i.e. removing all the optimizations made by the compiler. This allows us to clearly see the net effect of changes.

The routine `periodism` is written in this way:

```

do i=1,ndims
if (dx(i)>0.5*pb(i)*boxsize) dx(i)=dx(i) - pb(i)*boxsize
if (dx(i)<-0.5*pb(i)*boxsize) dx(i)=dx(i) + pb(i)*boxsize
end do

```

It consists of a few lines, but they are calculated many times in many parts of the code (see the Figure 10.3 showing the routines that call `periodism`).

Function/Method	Count	Total (s)	% ▲	Self (s)
MAIN_	1	3693.950	99.700	0.000
accelerate_	3510864	2577.590	69.600	377.420
accel_	301	2577.860	69.600	0.270
density_	301	1080.440	29.200	394.430
getneigh_accel_all_	1755432	746.640	20.200	364.640
getneigh_period_	3510864	742.670	20.000	217.320
getneigh_accel_dm_	1755432	697.950	18.800	330.210
periodism_	14625841134	630.260	17.000	630.260
getneigh_dens_all_	1785314	360.950	9.700	272.720
getneigh_dens_eps_	1774608	291.430	7.900	228.620
fper_	2232730145	271.030	7.300	271.030
soft_phi_	7626589592	215.050	5.800	215.050
soft_grav_	7626589592	207.320	5.600	207.320

Figure 10.1: gprof on a 18^3 particles, single CPU simulation on BUD. Routines are ordered per fraction of total inclusive time spent (i.e., time spent to call the routine plus time spent in all the depending subroutines).

Function/Method	Count	Total (s)	%	Self (s) ▲
periodism_	14625841134	630.260	17.000	630.260
density_	301	1080.440	29.200	394.430
accelerate_	3510864	2577.590	69.600	377.420
getneigh_accel_all_	1755432	746.640	20.200	364.640
getneigh_accel_dm_	1755432	697.950	18.800	330.210
getneigh_dens_all_	1785314	360.950	9.700	272.720
fper_	2232730145	271.030	7.300	271.030
getneigh_dens_eps_	1774608	291.430	7.900	228.620
getneigh_period_	3510864	742.670	20.000	217.320
soft_phi_	7626589592	215.050	5.800	215.050
soft_grav_	7626589592	207.320	5.600	207.320

Figure 10.2: gprof on a 18^3 particles, single CPU simulation on Bud. Routines are ordered per fraction of exclusive (SELF) time spent.

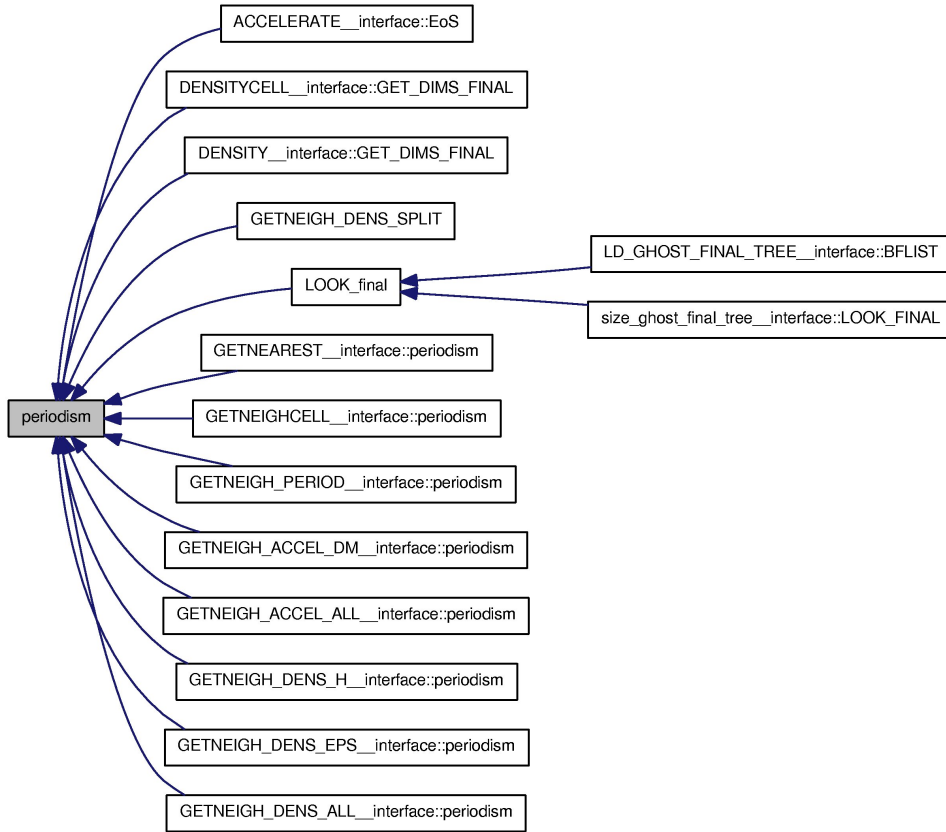


Figure 10.3: All the routines which call the routine periodism.

Therefore, even minor changes (improvements) will be very important and hopefully yield good results. In the following, we will examine in detail these few lines of code to give an example of the optimizing strategy.

Temporary variables. We start defining a temporary variable (`pbxh`) and write the instructions for `periodism` in the following way

```

do i=1,ndims
  pbx(i)= pb(i)*boxsize
  pbxh(i)= 0.5*pbx(i)
  if (dx(i)> pbxh(i)) dx(i)=dx(i) - pbx(i)
  if (dx(i)<-pbxh(i)) dx(i)=dx(i) + pbx(i)
end do

```

As particles are spatially close, the same holds for the variables and therefore we expect some effect (improvement) from the the use of the cache (location). After these simple changes, the execution time is reduced by as much as 6%.

Do not recalculate the useless. We also noticed that the quantities `pbx` and `pbxh` were always re-computed even if there was no real need to do so. Therefore, we calculated the two lines once for all before calling the routine that becomes,

```
do i=1,ndims
    if (dx(i)> pbxh(i)) dx(i)=dx(i) - pbx(i)
    if (dx(i)<-pbxh(i)) dx(i)=dx(i) + pbx(i)
end do
```

With this change there is a further improvement of 8%. The data with the total execution time of the code and the single periodic module, are shown in the Table 10.2 and Figure 10.4

Table 10.2: Improvement of periodism with respect to preprocessor version

Ver.	Prepr.-O0	Per I -O0	Per II -O0	Prepr.-O3	Per I -O3	Per II -O3
EvoL	3926.83	3693.95	3401.05	1412.45	1331.24	1258.04
Periodism	1052.13	630.26	329.44	331.75	191.76	108.16
Evol -%	-	-5.93	-7.93	-	-5.75	-5.50
Periodism -%	-	-40.10	-47.73	-	-42.20	-43.60

Periodism some more versions. Can we do better? Certainly we can. Let us see how!

Considering that, only one of two `if` is true for each particle, it is useless to check both. Therefore, the instructions are changed to (version III):

```
...
if (dx(i)> pbxh(i)) THEN
    dx(i)=dx(i) - pbx(i)
else if (dx(i)<-pbxh(i))THEN
    dx(i)=dx(i) + pbx(i)
...

```

There are still six tests, three on each side of the box. Therefore they can be further reduced writing (version IV):

```
...
if (ABS(dx(1))>pbxh(1))THEN
    dx(1)=dx(1)-sign(pbx(1),dx(1))
endif

if (ABS(dx(2))>pbxh(2))THEN
    dx(2)=dx(2)-sign(pbx(2),dx(2))
endif

```

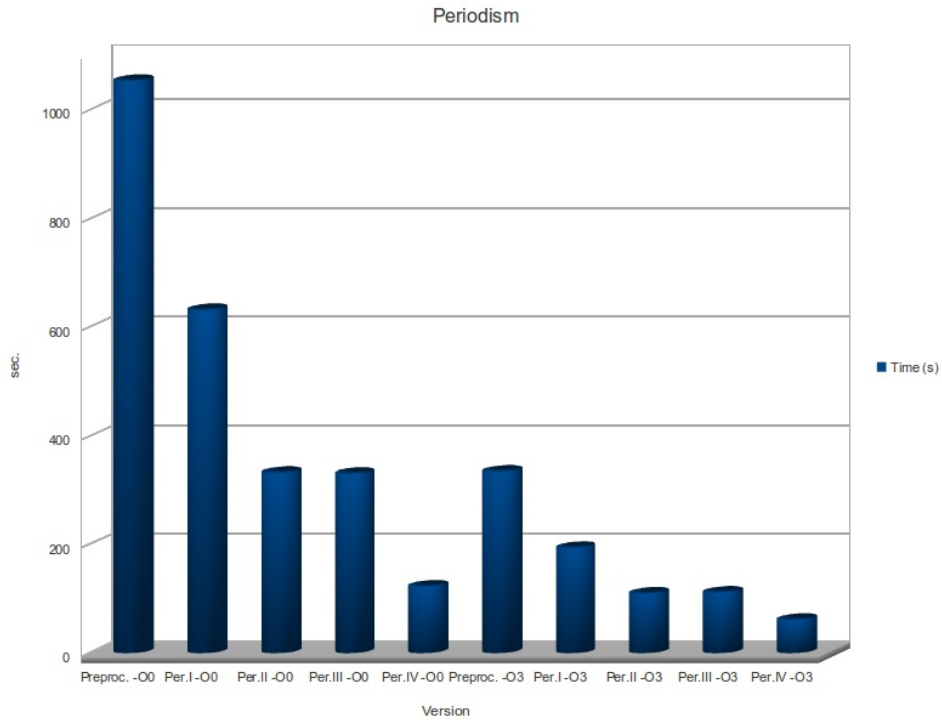


Figure 10.4: Final improvements of the routine `periodism` with respect to the preprocessed version.

```

if (ABS(dx(3))>pbxh(3))THEN
    dx(3)=dx(3)-sign(pbx(3),dx(3))
endif
...

```

eliminating the cycle of `ndims` which is fixed at the beginning of the simulation, and reducing the tests to three. The price to pay are the two functions `abs` and `sign` that were not present before. One should check that the total budget is still positive for us. The test shows that this is the case both at compilation `-O0` and `-O3`, see the entries of Table 10.3.

The final reduction of the total execution time for the entire code, and of the single routine in question, are shown in the Table 10.4. In the Figure 10.4 improvements of the different versions of `periodism` with respect to the preprocessed version.

Table 10.3: Further improvements of the routine periodism with respect to the previous version.

Version	Per.II -O0	Per.III -O0	Per.IV -O0	Per.II -O3	Per.III -O3	Per.IV -O3
EvoL	3401.05	3411.45	3201.37	1258.04	1249.27	1158.24
Period.	329.44	327.72	120.91	108.16	110.07	59.99
Evol -%	-	+0.31%	-6.16%	-	-0.70%	-7.93 %
Per -%	-	-0.71%	-10.55%	-	+1.77%	-44.54%

Table 10.4: Final improvements of the routine periodism with respect to the preprocessed version.

Version	Per. III -O0	Per. IV -O0	Per. III -O3	Per. IV -O3
EvoL %	-13.12%	-18.47%	-11.55 %	-18.00%
Periodism %	-68.85%	-88.51%	-66.82%	-81.92%

10.4 soft_phi and soft_grav

These are the functions which apply the softening to all particles, heavily weighting on the overall computing time of the code: their total weight is about 15%. The functions are called many times in close pairs and there is also some redundancy in the quantities calculated. To optimize the use of these routines we have applied the following corrections:

- we removed many calculations and the ones repeated several times, saving in the number of operations;
- we fostered a positive use of the memory hierarchy, because the quantities loaded into the cache from the ram memory are reused almost immediately, and are more likely they still reside in the cache when they are reused.

The two functions have been replaced by a single routine computing the same quantities. Profiling the code it is soon evident that, as expected, the strategy works perfectly and that the computing time decreases even compared to the previous version Period IV -O3 by about 5%. Table 10.5 shows the time with the old functions at work and the time with the new routine `soft_phi_grav`. The improvement amounts to about 25%.

10.5 Functions fper and pper

These two functions terminate the calculation of periodism, giving the correction to the force and potential for the presence of infinite replicas of the box. Together, the functions take about 15% of the total execution time. The

Table 10.5: Improvements of the `soft_phi` and `soft_grav` routines with respect to the Period IV version.

Ver.	Period IV -O3	SoftPG -O3
EvoL	1158.24	1098.26
<code>soft_phi</code> + <code>soft_grav</code>	178.55	133.84
EvoL %	-	-5.18%
<code>soft_phi</code> + <code>soft_grav</code> %	-	-25.04%

two functions, although short, make use of unnecessary `if` constructions that are notoriously heavy. Rethinking the algorithm slightly and removing the `if`, we rewrite the important parts of the routines making use of the intrinsic function `ABS`.

These changes lead to the improvement we are looking for. Moreover, the two functions share a part of code and execute twice the same calculations, so even in this case it is wise to merge them into a single routine. The times obtained with `gprof` show that the improvement brought about by the above changes a further 5% of the code as a whole and 57% in the two functions considered. The data are displayed in Table 10.6.

Table 10.6: Improvements of the routines `fper` and `pper` with respect to the SoftPG version.

Ver.	SoftPG -O3	FPper -O3
EvoL	1098.26	1039.54
<code>fper</code> + <code>pper</code>	181.56	104.96
EvoL %	-	-5.35%
<code>fper</code> + <code>pper</code> %	-	-57.81%

10.6 Back to the compiler

Finally, after getting these significant reductions in time, it is useful to let the compiler take care of all additional optimizations that can be activated by the compilation procedure. Although the optimization level `-O3` is already a high one it does not automatically enable all possible additional optimizations. The choice must be made by the user to push the code to full seed.

It is worth recalling however that the high-level optimizations, changing the semantics of the code and the order of operations may introduce approximations that sometimes lead to unacceptable errors in the results or crashing of the code the code. In addition, when using options little tested

on our code, we must use them carefully and then check the results. In our case the tests have confirmed the full accuracy of the results. These latter indeed are the same before these additional optimizations.

The flags activated to let the GNU compiler add the final optimizations are:

```
-O3
-funroll-loops
-fipa-cp
-fipa-matrix-reorg
-fipa-type-escape
-fsingle-precision-constant
```

Looking at the entries of Table 10.7, although there are clear evidences of additional improvement there are signs of a light deterioration. The total computational time, however, significantly decreases.

Table 10.7: After the final optimization procedure has been added by the compiler.

Version	FPper -O3	IPA -O3
EvoL	1039.54	862.33
soft_phi+soft_grav	133.12	110.48
fper+pper	104.96	102.05
EvoL %	-	-17.05%
soft_phi+soft_grav	-	-17.01%
fper+pper %	-	-2.78%

10.7 Total Optimization obtained in summary

If we take as reference the original version of the code without optimization at all, the overall improvement (in the sense of reducing the total measurement time) is about a factor 8-9, depending on the type and configuration of the simulation code. Figure 10.5 compares the execution time of the original with the new version of the code. The times at intermediate optimization levels are also displayed for comparison.

For the sake of better insight, in Figure 10.6 we show only the optimizations obtained with the code already in the preprocessor version (with the pre-compiler directive `#IFDEF`).

Cache use after optimization. After applying the optimization procedure, it is worth checking whether the shorter execution time is due at least in part, to a better use of cache memory hierarchy. Figure 10.7 and

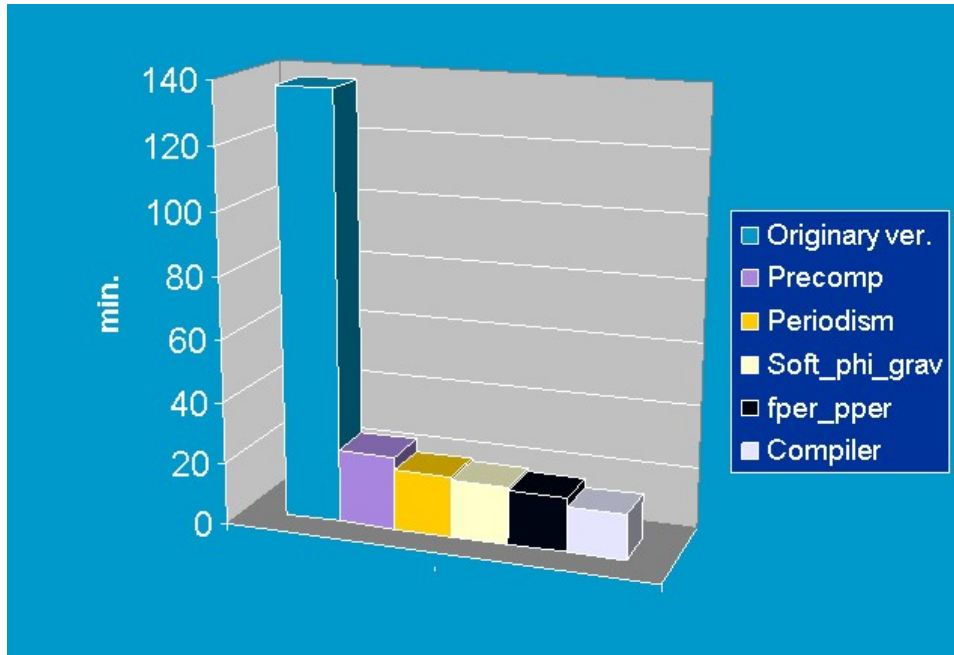


Figure 10.5: Total optimization obtained.

Table 10.8: Percentage of reduction of cache-miss after optimization

Function	% Reduction of cache-miss
EvoL	-19%
periodism	-62%
soft_phi+soft_grav	-33%
fper+pper	-22%

Table 10.8 show the estimates made by cachegrind of the number of cycles lost due to cache-misses.

The number of cache misses is still high (leaving open the possibility of further improvements), but there is anyway a good reduction that partly explains the reduction of the total execution time.

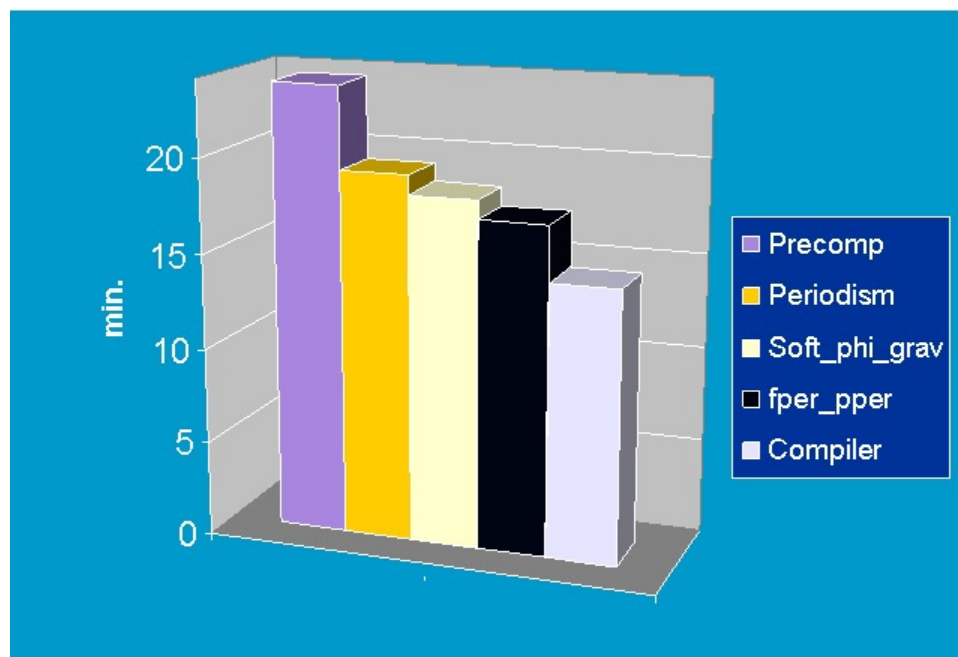


Figure 10.6: Same as in Fig. 10.5 excluding the first column

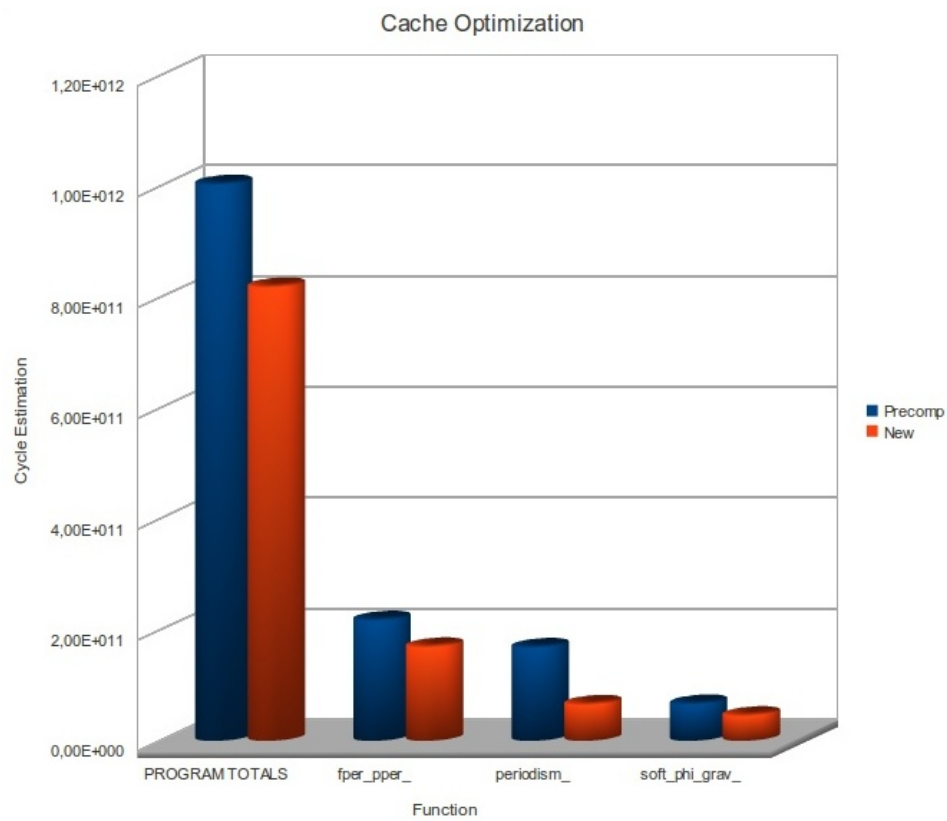


Figure 10.7: Cache use after optimization

Chapter 11

Modelling galaxies with EvoL

As already mentioned, EvoL is not only the target of dedicated studies on computational performance, but it is also the tool we adopted to attack the problem of galaxy formation and evolution. Therefore with the aid of EvoL and a few ancillary codes like ROBO and MANN we have produced a number of paper devoted to simulating the formation and evolution of early type galaxies taking into account also the thermodynamical behaviour of their interstellar medium. In this chapter we list the complete referencing to the studies in question and present the full text of four of them particularly relevant to the aims of this thesis (in boldface).

(1) **Merlin, E., Chiosi, C., Piovan, L., Grassi, T., Buonomo, U., La Barbera, F., "Formation and Evolution of Early-Type Galaxies. III Star formation history as a function of mass and over-density", 2012, Astron. Astrophys., submitted**

(2) **Grassi, T., Krstic, P., Merlin, E., Buonomo, U., Piovan, L., Chiosi, C., "ROBO: a model, a code for studying the interstellar medium", "arXiv"1012.1142, 2011, Astron. Astrophys., 533, A123**

(3) Piovan, L., Chiosi, C., Merlin, E., Grassi, T., Tantalo, R., Buonomo, U., Cassarà , L. P., "Formation, Evolution of the Dust in Galaxies. III. The Disk of the Milky Way", "arXiv",1107.4567, 2012, Astron. Astrophys., submitted

(4) Piovan, L., Chiosi, C., Merlin, E., Grassi, T., Tantalo, R., Buonomo, U., Cassarà , L. P., "Formation, Evolution of the Dust in Galaxies. II. The Solar Neighbourhood ", "arXiv",1107.4561, 2012, Astron. Astrophys., submitted

(5) Piovan, L., Chiosi, C., Merlin, E., Grassi, T., Tantalo, R., Buonomo, U., Cassarà , L. P., "Formation, Evolution of the Dust in Galaxies. I. The Condensation Efficiencies", "arXiv",1107.4541, 2012, Astron. Astrophys., submitted

(6) Grassi, T., Merlin, E., Piovan, L., Buonomo, U., Chiosi, C., "MaNN: Multiple Artificial Neural Networks for modelling the Interstellar Medium", "arXiv", 1103.0509, 2012, *Astron. Astrophys.*, submitted

(7) Merlin, E., Buonomo, U., Grassi, T., Piovan, L., Chiosi, C., "EvoL: the new Padova Tree-SPH parallel code for cosmological simulations. I. Basic code: gravity, hydrodynamics", "arXiv", 0912.1786, 2010, *Astron. Astrophys.*, 513, A36

(8) Merlin, E., Chiosi, C., Grassi, T., Buonomo, U., Chinellato, S., "Simulating the formation, evolution of galaxies with EvoL, the Padova N -body Tree-SPH Code ", *Nuovo Cimento C Geophysics Space Physics C*, 2009, V. 32, 2, p. 020000-146

11.1 Scientific Rationales of the selected papers

Formation and Evolution of Early-Type Galaxies. III Star formation history as a function of mass and over-density. To date, a consistent and comprehensive theory of galaxy formation is still missing. Growing observational evidences claim for a population of massive, red galaxies at very high redshifts, apparently at odds with the theoretical concordance cosmology scenario, in which larger systems form later via merger of smaller subunits. How did the big, early red galaxies which we observe at high redshifts form? Did they assemble their stellar content in a strong, early burst of star formation activity? Is this scenario consistent with the theoretical cosmological background? We investigate the influence of the initial proto-galaxies over-densities and masses on their evolution, to understand whether the internal properties of the proto-galactic halos are sufficient to account for the varied properties of the galactic populations. By means of fully hydrodynamical N-body simulations performed with the code EVOL we produce twelve self-similar models of early-type galaxies following their evolution from the epoch of their detachment from the linear regime, i.e. $z \geq 20$ to $z \leq 1$ (we also produce some more ancillary models for further analysis). The simulations include radiative cooling, star formation, stellar energy feedback, a reionizing photoheating background, and chemical enrichment of the ISM. We do not consider the possible presence of Active Nuclei. We find a stunning correlation between the initial properties of the proto-halos and their star formation histories. Massive ($M_{tot} \simeq 10^{13} M_{\odot}$) halos experience a single, intense burst of star formation (with rates $\geq 10^3 M_{\odot}/yr$) at early epochs, consistently with observations, with a less pronounced dependence on the initial over-density; intermediate mass ($M_{tot} \simeq 10^{11} M_{\odot}$) halos histories strongly depend on their initial over-density, whereas small ($M_{tot} \simeq 10^9 M_{\odot}$) halos always have fragmented histories, resulting in multiple stellar populations, due to a "galactic breathing" phenomenon. The

galaxy models have morphological, structural and photometric properties comparable to real galaxies, often closely matching the observed data; there are some incongruities which we interpret as a consequence of some numerical choices. The position of the models on diagnostic planes is generally consistent with observed galaxies. We conclude that internal properties are essentially sufficient to explain many of the observed features of early type galaxies, particularly the complicated and different star formation histories shown by halos of very different mass. In this picture, nature seems to play a pivotal role, whereas the action of nurture is of secondary importance.

ROBO: a model and a code for studying the interstellar medium.

The aim is to provide an accurate description of the physical evolution of the ISM and to set the ground for an ancillary tool to be inserted in NB-TSPH simulations of large-scale structures in the cosmological context or of the formation and evolution of individual galaxies. The ISM model consists of gas and dust. The gas chemical composition is regulated by a network of reactions that includes a large number of species (hydrogen and deuterium-based molecules, helium, and metals). New reaction rates for the charge transfer in H^+ and H_2 collisions are presented. The dust contains the standard mixture of carbonaceous grains (graphite grains and PAHs) and silicates. In our model dust are formed and destroyed by several processes. The model accurately treats the cooling process, based on several physical mechanisms, and cooling functions recently reported in the literature. The model is applied to a wide range of the input parameters and the results for important quantities describing the physical state of the gas and dust are presented. The results are organized in a database suited to the artificial neural networks (ANNs). Once trained, the ANNs yield the same results obtained by ROBO with great accuracy. We plan to develop ANNs suitably tailored for applications to NB-TSPH simulations of cosmological structures and/or galaxies.

MaNN: Multiple Artificial Neural Networks for modelling the Interstellar Medium. Modelling the complex physics of the Interstellar Medium in the context of large-scale numerical simulations is a challenging task. A number of methods have been proposed to embed a description of the ISM into different codes. We propose a new way to achieve this task: Artificial Neural Networks. The ANN has been trained on a pre-compiled model database, and its predictions have been compared to the expected theoretical ones, finding good agreement both in static and in dynamical tests run using the Padova Tree-SPH code *EvoL*. A neural network can reproduce the details of the interstellar gas evolution, requiring limited computational resources. We suggest that such an algorithm can replace a real-time calculation of mass elements chemical evolution in hydrodynamical codes.

EvoL: The new Padova T-SPH parallel code for cosmological simulations - I. Basic code: gravity and hydrodynamics. We present *EvoL*, the new release of the Padova N-body code for cosmological simulations of galaxy formation and evolution. In this paper, the basic Tree + SPH

code is presented and analysed, together with an overview on the software architectures. EVOL is a flexible parallel Fortran95 code, specifically designed for simulations of cosmological structure formation on cluster, galactic and sub-galactic scales. EVOL is a fully Lagrangian self-adaptive code, based on the classical Oct-tree by (Barnes and Hut 1986) and on the Smoothed Particle Hydrodynamics algorithm (SPH, (Lucy 1977)). It includes special features such as adaptive softening lengths with correcting extra-terms, and modern formulations of SPH and artificial viscosity. It is designed to be run in parallel on multiple CPUs to optimize the performance and save computational time. We describe the code in detail, and present the results of a number of standard hydrodynamical tests.

Chapter 12

What to do next

The ideas and procedures presented in this thesis are the first step towards the ideal performance of the code *EvoL*, the main tool for modelling the physical processes leading to the formation of galaxies, following their evolution in time, providing the correct description of their structure, and describing in detail their stellar and gaseous contents.

First of all, from the numerical point of view some aspects still need to be analysed and improved. They are shortly summarized below:

- Fully optimize the code for high performance in the serial mode. This will have additional improvements also in the parallel mode.
- Identify and reduce (if possible) the serial fraction of the code.
- Reduce the load imbalance:

Adopt the Peano-Hilbert domain decomposition instead of the Orthogonal Recursive Bisection (ORB).

Divide of the computational domain mimicking the division of computational nodes in the machine: a communication between processors avoids mixing occurring in the same node (the node using the ram and not the underlying network) with those between processors belonging to different nodes.

Use OpenMP within a node to improve the balance of the loop using the dynamical scheduling: once a particular thread finishes its allocated iteration, it returns to get another one from the iterations that are left. Care must be paid to the time spent in the phases of fork and join¹.

¹All OpenMP programs begin as a single process. The master thread then creates a team of parallel threads (FORK phase); when the team threads complete the statements in the parallel region, they synchronize and terminate, leaving only the master thread (JOIN phase)

- Reorganize the data structures to reduce cache misses. We can arrange the neighbor searches for one particle in such a way that is immediately followed by neighbor searches for a nearby particle. The data (nearly identical neighbor lists) are probably already in the cache. Therefore, the calculations will proceed much faster due to the decreased load times ((Warren and Salmon 1995; Springel 2005; Stadel 2001).
- Neighboring particles in space should be stored physically close in the system's memory, so that the correlated spatial positions and memory locations ensure that data are ordinarily found in the same set of physical pages of memory².
- Improve the time-stepping algorithm, following the suggestions by Quinn et al. (1997) for cosmological comoving simulations.
- Compare *EvoL* with similar codes in literature such as *GADGET*, *FLASH*, etc., once the optimization level reached by *EvoL* is considered fully satisfactory.

From the point view of the physical content the most physical ingredients to include are:

- The presence of magnetic fields in the SPH equations, to study of the effects of these on the structure formation.
- The heating by cosmic rays and/or AGN as additional and important sources of energy in the total energy budget of a galaxy.
- The diffusion of photons to properly model the ionization of neutral regions, especially at high redshifts.
- The possible and likely variation of IMF in the chemical enrichment, and energy feed-back.
- A better treatment of the interaction among particles of different mass.

²The Virtual Memory, seen from a process, can be larger than physical memory. This space is segmented into fixed size pages. Generally, only a few pages are loaded into physical memory and load a different page in physical memory takes time.

Chapter 13

General discussion and conclusions

This thesis describes the various steps through which we have developed a numerical code for modelling the formation and evolution of galaxies made of dark and baryonic matter in cosmological context, with particular emphasis on single early type systems.

The numerical simulations of galaxy formation and evolution, in which many physical processes are taken into account going from the formation of cosmological perturbation, to their growth with time, their collapse to proto-galaxies, gas heating and cooling by many physical agents, star formation and evolution, stellar and galactic winds, chemical enrichment, and companion spectro-photometric evolution, and finally the overall dynamical and morphological structure of a galaxy, are a tremendous task that require at least highly sophisticated numerical codes and long computational times even on the most powerful computers.

On one hand, all these aspects of the problem have been the targets of an articulated research project carried on for about a decade by the Padova team of theoretical Astrophysics; on the other hand they have spurred the set up of the high performance numerical code *EvOL*, entirely built in the house, to model with the NB-TSPH formalism and technique the galactic systems in question. The task was not easy because developing such a code requires expertises in the area of computer sciences that often goes beyond the classical cultural equipments of an astrophysicist.

The aim of the thesis articulates along three lines of thought: first, understanding the physics of galaxy formation and evolution; second, to develop a high performance numerical code; third, to acquire the required expertise in the field of computational sciences, in view of future applications.

The first goal was the ground of the other two, and gave origin to a number of studies and models of galaxies that turned out to be in good agreement with observational data, on which we have reported in course of

the thesis and will not be touched upon here. We limit ourselves to the series of papers concerning the hydrodynamical tests to check the physical models of galaxies, the new treatment of the interstellar medium with ROBO and companion MaNN, and the production of galaxy models to understand the relation between the initial mass/density and the history of star formation. A particular mention is deserved the study on the relationship between the star mass and effective radius of galaxies for which an elegant explanation is found. The thesis indeed is more focused on the other goals.

In the first part we summarized the fundamentals of informatics necessary to understand in some detail the functioning of a modern computer. We briefly reviewed the structure of a computer, its memory banks and communication network. This is the prerequisite to understand why a numerical code may give unsatisfactory performances. Then we present the key ideas on optimization techniques that are commonly adopted to get high performance codes. Often, ignoring these basic information may lead to codes that despite their formal adequacy, still do not exploit all the resources offered by modern computers. We described the methods commonly used to cure the problems. In this context we defined the quantities customarily adopted to rank the performances of a code and to understand its limitation and drawbacks.

We also introduced the fundamentals of code debugging, and the correct use of the compilation flags of a computer, to get rid of pitfalls and drawbacks. We illustrated the techniques and softwares commonly used to get information on the functioning of a code, and to monitor the critical points responsible for the performance degrading. We then passed to describe simple software for scientific visualization that are very useful to get a first idea on how a system evolves or to detect where, why, and how a code may fail. We proceeded then to describe in detail the main characteristics of EvOL, and the results of the profile analysis applied to it. On the basis of this analysis, we may pin down the main causes limiting the performances of EvOL; these are:

- The periodism in cosmological simulations;
- Search of neighbours to calculate the density;
- Gravitational and hydrodynamical softening;
- Computation of acceleration of individual particles.

The cause of informatics origin generating these problems are:

- Bad use of the memory hierarchy;
- Insufficient implementation of the code (empty pipeline and/or redundancy of calculations)

- Use of inefficient algorithms.

As far as the MPI communications are concerned we recall:

- Superfluous communications of data structures (excess of communications);
- Excessive use of barriers that often halt the processors;
- Bad balance of the computational load among different processors;
- Limited scalability due to the presence of a serial part in the code.

Starting from this analysis we deeply optimized the code drastically reducing its execution time. Even if the result can be considered as highly successful (computational times reduced by one order of magnitude or so), the work is still preliminary and much remains to be done. The available data indicate that an ample range for improvements exists recasting some sections of the code. For instance a better use of the cache could be possible by reconsidering the order with which particles are stored and bring in turn a good reduction of the computational time to calculate distances and search the neighbours. Furthermore, reducing the serial fraction of the code (if possible) would give a better exploitation of the available processors in parallel mode. The increase in computational speed could be used to implement new physics such as the magnetic fields, complete treatment of dust as an important component of the interstellar medium, and finally the inclusion of AGNs and their effect on the star formation history.

Acknowledgments

I would like to thank my supervisor prof. C. Chiosi, for the precious lessons provided to me in these years and for the many hours devoted to my thesis, and also E. Merlin, T. Grassi, L. Piovan, for the useful discussions and their concrete help.

I am grateful to all my family, for the great help and support they have given me. In particular I thank Sara for her patience, for her encouragement and for having always stayed close.

Bibliography

- J.-M. Alimi, A. Serna, C. Pastor, and G. Bernabeu. Smooth particle hydrodynamics: importance of correction terms in adaptive resolution algorithms. *Journal of Computational Physics*, 192:157–174, November 2003. doi: 10.1016/S0021-9991(03)00351-6.
- Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. doi: <http://doi.acm.org/10.1145/1465482.1465560>. URL <http://doi.acm.org/10.1145/1465482.1465560>.
- E. Athanassoula, E. Fady, J. C. Lambert, and A. Bosma. Optimal softening for force calculations in collisionless N-body simulations. *MNRAS*, 314: 475–488, May 2000. doi: 10.1046/j.1365-8711.2000.03316.x.
- E. L. O. Bakes and A. G. G. M. Tielens. The photoelectric heating mechanism for very small graphitic grains and polycyclic aromatic hydrocarbons. *ApJ*, 427:822–838, June 1994. doi: 10.1086/174188.
- J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986. doi: 10.1038/324446a0.
- M. R. Bate and A. Burkert. Resolution requirements for smoothed particle hydrodynamics calculations with self-gravity. *MNRAS*, 288:1060–1072, July 1997.
- U. Becciani, C. Gheller, V. Antonuccio, D. Ferro, and M. Melotti. AstroMD, a tool for Stereographic Visualization and data analysis for astrophysical data. *Memorie della Societa Astronomica Italiana Supplementi*, 1:80, 2003.
- U. Becciani, M. Comparato, and C. Gheller. VisIVO: a VO enabled tool for Scientific Visualization and Data Analysis . *Memorie della Societa Astronomica Italiana Supplementi*, 9:427, 2006.
- U. Becciani, A. Costa, V. Antonuccio-Delogu, G. Caniglia, M. Comparato, C. Gheller, Z. Jin, M. Krokos, and P. Massimino. VisIVO: Integrated Tools

- and Services for Large-Scale Astrophysical Visualization. In *Astrophysics Source Code Library, record ascl:1011.020*, page 11020, November 2010.
- W. Benz. Smooth Particle Hydrodynamics - a Review. In J. R. Buchler, editor, *Numerical Modelling of Nonlinear Stellar Pulsations Problems and Prospects*, page 269, 1990.
- E. Bertschinger. COSMICS: Cosmological Initial Conditions and Microwave Anisotropy Codes. *ArXiv Astrophysics e-prints*, June 1995.
- V. Bromm, P. S. Coppi, and R. B. Larson. The Formation of the First Stars. I. The Primordial Star-forming Cloud. *ApJ*, 564:23–51, January 2002. doi: 10.1086/323947.
- G. Carraro, C. Lia, and C. Chiosi. Galaxy formation and evolution - I. The Padua tree-sph code (pd-sph). *MNRAS*, 297:1021–1040, July 1998. doi: 10.1046/j.1365-8711.1998.2970041021.x.
- R. Cen. A hydrodynamic approach to cosmology - Methodology. *ApJS*, 78: 341–364, February 1992. doi: 10.1086/191630.
- C. Chiosi, A. Bressan, L. Portinari, and R. Tantalo. A new scenario of galaxy evolution under a universal Initial Mass Function. *AAP*, 339:355–381, November 1998.
- H. Cho and H. Kang. Feedback from multiple supernova explosions inside a wind-blown bubble. *NewA*, 13:163–177, April 2008. doi: 10.1016/j.newast.2007.07.006.
- D. K. Churches, A. H. Nelson, and M. G. Edmunds. Numerical simulations of the formation and chemical evolution of galaxies. *MNRAS*, 327:610–622, October 2001. doi: 10.1046/j.1365-8711.2001.04743.x.
- B. Ciardi. Feedback From the First Stars and Galaxies and its Influence on Structure Formation. In B. W. O’Shea and A. Heger, editors, *First Stars III*, volume 990 of *American Institute of Physics Conference Series*, pages 353–363, March 2008. doi: 10.1063/1.2905580.
- R. Dave, J. Dubinski, and L. Hernquist. Parallel TreeSPH. *NewA*, 2:277–297, August 1997. doi: 10.1016/S1384-1076(97)00019-5.
- W. Dehnen. Towards optimal softening in three-dimensional N-body codes - I. Minimizing the force error. *MNRAS*, 324:273–291, June 2001. doi: 10.1046/j.1365-8711.2001.04237.x.
- J. Dongarra. *The Sourcebook of Parallel Computing*. The Morgan Kaufmann Series in Computer Architecture and Design, 2003.

- J. E. Dyson and D. A. Williams. *The physics of the interstellar medium*. The physics of the interstellar medium. Edition: 2nd ed. Publisher: Bristol: Institute of Physics Publishing, 1997. Edited by J. E. Dyson and D. A. Williams. Series: The graduate series in astronomy. ISBN: 0750303069, 1997.
- B. G. Elmegreen. On the Rapid Collapse and Evolution of Molecular Clouds. *ApJ*, 668:1064–1082, October 2007. doi: 10.1086/521327.
- P. P. Ewald. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Annalen der Physik*, 369:253–287, 1921. doi: 10.1002/andp.19213690304.
- A. Ferrara. The Positive Feedback of Population III Objects on Galaxy Formation. *ApJ*, 499:L17, May 1998. doi: 10.1086/311344.
- D. Galli and F. Palla. The chemistry of the early Universe. *AAP*, 335:403–420, July 1998.
- R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *MNRAS*, 181:375–389, November 1977.
- S. C. O. Glover and A.-K. Jappsen. Star Formation at Very Low Metallicity. I. Chemistry and Cooling at Low Densities. *ApJ*, 666:1–19, September 2007. doi: 10.1086/519445.
- T. Grassi, P. Krstic, E. Merlin, U. Buonomo, L. Piovan, and C. Chiosi. ROBO: a Model and a Code for the Study of the Interstellar Medium. *ArXiv e-prints*, December 2010.
- L. Greggio and A. Renzini. The binary model for type I supernovae - Theoretical rates. *AAP*, 118:217–222, February 1983.
- W. Gropp, E. Lusk, and A. Skjellum. *Using MPI : portable parallel programming with the message-passing interface*. Scientific and engineering computation: Cambridge, MA ; London : MIT, 1994.
- John L. Gustafson. Reevaluating amdahl’s law. *Communications of the ACM*, 31:532–533, 1988.
- J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 4th edition 2007 edition, 2007.
- L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *ApJS*, 70:419–446, June 1989. doi: 10.1086/191344.

- L. Hernquist, F. R. Bouchet, and Y. Suto. Application of the Ewald method to cosmological N-body simulations. *ApJS*, 75:231–240, February 1991. doi: 10.1086/191530.
- G. Hinshaw, J. L. Weiland, R. S. Hill, N. Odegard, D. Larson, C. L. Bennett, J. Dunkley, B. Gold, M. R. Greason, N. Jarosik, E. Komatsu, M. R. Nolte, L. Page, D. N. Spergel, E. Wollack, M. Halpern, A. Kogut, M. Limon, S. S. Meyer, G. S. Tucker, and E. L. Wright. Five-Year Wilkinson Microwave Anisotropy Probe Observations: Data Processing, Sky Maps, and Basic Results. *ApJS*, 180:225–245, February 2009. doi: 10.1088/0067-0049/180/2/225.
- D. Hollenbach. Heating and cooling of molecular clouds and their surfaces. *Astrophysical Letters and Communications*, 26:191–205, 1988.
- D. Hollenbach and C. F. McKee. Molecule formation and infrared emission in fast interstellar shocks. I Physical processes. *ApJS*, 41:555–592, November 1979. doi: 10.1086/190631.
- D. Hollenbach and C. F. McKee. Molecule formation and infrared emission in fast interstellar shocks. III - Results for J shocks in molecular clouds. *ApJ*, 342:306–336, July 1989. doi: 10.1086/167595.
- T. Hosokawa and S.-i. Inutsuka. Dynamical Formation of Dark Molecular Hydrogen Clouds around Diffuse H II Regions. *ApJ*, 664:363–376, July 2007. doi: 10.1086/518396.
- IBM Optimization and Tuning Guide 1996. *Optimization and Tuning Guide for Fortran, C, and C++*. IBM, 1996.
- IBM Optimization and Programming Guide 2010. *Optimization and Programming Guide*. IBM, 2010.
- S. Ikeuchi and J. P. Ostriker. Evolution of the intergalactic medium - What happened during the epoch $Z = 3-10$? *ApJ*, 301:522–543, February 1986. doi: 10.1086/163921.
- Intel Optimization Reference Manual 2011. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel® Corporation, June 2011.
- Intel® Trace Analyzer and Collector Documentation. Trace analyzer and collector documentation, 2011. <http://software.intel.com/en-us/articles/intel-trace-analyzer-and-collector-documentation/>.
- N. Katz. Dissipational galaxy formation. II - Effects of star formation. *ApJ*, 391:502–517, June 1992. doi: 10.1086/171366.
- R. C. Kennicutt, Jr. The Global Schmidt Law in Star-forming Galaxies. *ApJ*, 498:541, May 1998. doi: 10.1086/305588.

- P. Kroupa, S. Aarseth, and J. Hurley. The formation of a bound star cluster: from the Orion nebula cluster to the Pleiades. *MNRAS*, 321:699–712, March 2001. doi: 10.1046/j.1365-8711.2001.04050.x.
- C. Lia, L. Portinari, and G. Carraro. Star formation and chemical evolution in smoothed particle hydrodynamics simulations: a statistical approach. *MNRAS*, 330:821–836, March 2002. doi: 10.1046/j.1365-8711.2002.05118.x.
- A. Lipovka, R. Núñez-López, and V. Avila-Reese. The cooling function of HD molecule revisited. *MNRAS*, 361:850–854, August 2005. doi: 10.1111/j.1365-2966.2005.09226.x.
- L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *AJ*, 82:1013–1024, December 1977. doi: 10.1086/112164.
- U. Maio, K. Dolag, B. Ciardi, and L. Tornatore. Metal and molecule cooling in simulations of structure formation. *MNRAS*, 379:963–973, August 2007. doi: 10.1111/j.1365-2966.2007.12016.x.
- S. Mashchenko, J. Wadsley, and H. M. P. Couchman. Stellar Feedback in Dwarf Galaxy Formation. *Science*, 319:174–, January 2008. doi: 10.1126/science.1148666.
- E. Merlin and C. Chiosi. Simulating the formation and evolution of galaxies: multi-phase description of the interstellar medium, star formation, and energy feedback. *AAP*, 473:733–745, October 2007. doi: 10.1051/0004-6361:20077465.
- E. Merlin, U. Buonomo, T. Grassi, L. Piovan, and C. Chiosi. EvoL: the new Padova Tree-SPH parallel code for cosmological simulations. I. Basic code: gravity and hydrodynamics. *AAP*, 513:A36, April 2010. doi: 10.1051/0004-6361/200913514.
- E. Merlin, C. Chiosi, L. Piovan, T. Grassi, U. Buonomo, and F. La Barbera. Formation and Evolution of Early-Type Galaxies. III Star formation history as a function of mass and over-density. in preparation, 2012.
- D. Merritt. Optimal Smoothing for N-Body Codes. *AJ*, 111:2462, June 1996. doi: 10.1086/117980.
- J. J. Monaghan. Smoothed particle hydrodynamics. *ARAA*, 30:543–574, 1992. doi: 10.1146/annurev.aa.30.090192.002551.
- J. J. Monaghan. SPH compressible turbulence. *MNRAS*, 335:843–852, September 2002. doi: 10.1046/j.1365-8711.2002.05678.x.
- J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *AAP*, 149:135–143, August 1985.

- R. P. Nelson and J. C. B. Papaloizou. Variable Smoothing Lengths and Energy Conservation in Smoothed Particle Hydrodynamics. *MNRAS*, 270:1, September 1994.
- Open MPI Documentation. Open mpi documentation, 2011. <http://www.open-mpi.org/doc/>.
- OpenMP Tutorials. Openmp tutorials, 2011. <http://openmp.org/wp/resources/#Tutorials>.
- B. W. O’Shea, G. Bryan, J. Bordner, M. L. Norman, T. Abel, R. Harkness, and A. Kritsuk. Introducing Enzo, an AMR Cosmology Application. *ArXiv Astrophysics e-prints*, March 2004.
- B. W. O’Shea, K. Nagamine, V. Springel, L. Hernquist, and M. L. Norman. Comparing AMR and SPH Cosmological Simulations. I. Dark Matter and Adiabatic Simulations. *ApJS*, 160:1–27, September 2005. doi: 10.1086/432645.
- J. M. Owen and J. V. Villumsen. Baryons, Dark Matter, and the Jeans Mass in Simulations of Cosmological Structure Formation. *ApJ*, 481:1, May 1997. doi: 10.1086/304018.
- P. J. E. Peebles. *The large-scale structure of the universe*. Research supported by the National Science Foundation. Princeton, N.J., Princeton University Press, 1980. 435 p., 1980.
- F. I. Pelupessy, P. P. van der Werf, and V. Icke. Periodic bursts of star formation in irregular galaxies. *AAP*, 422:55–64, July 2004. doi: 10.1051/0004-6361:20047071.
- D. J. Price. splash: An Interactive Visualisation Tool for Smoothed Particle Hydrodynamics Simulations. *PASA*, 24:159–173, October 2007. doi: 10.1071/AS07022.
- D. J. Price. Modelling discontinuities and Kelvin Helmholtz instabilities in SPH. *Journal of Computational Physics*, 227:10040–10057, December 2008. doi: 10.1016/j.jcp.2008.08.011.
- D. J. Price and J. J. Monaghan. An energy-conserving formalism for adaptive gravitational force softening in smoothed particle hydrodynamics and N-body codes. *MNRAS*, 374:1347–1358, February 2007. doi: 10.1111/j.1365-2966.2006.11241.x.
- T. Quinn, N. Katz, J. Stadel, and G. Lake. Time stepping N-body simulations. *ArXiv Astrophysics e-prints*, *astro-ph/9710043*, October 1997.

- B. E. Robertson and A. V. Kravtsov. Molecular Hydrogen and Global Star Formation Relations in Galaxies. *ApJ*, 680:1083–1111, June 2008. doi: 10.1086/587796.
- A. B. Romeo. Modelling gravity in N-body simulations of disc galaxies. Optimal types of softening for given dynamical requirements. *AAP*, 335: 922–928, July 1998.
- S. Rosswog. Astrophysical smooth particle hydrodynamics. *NewAR*, 53: 78–104, April 2009. doi: 10.1016/j.newar.2009.08.007.
- D. Ryu, J. P. Ostriker, H. Kang, and R. Cen. A cosmological hydrodynamic code based on the total variation diminishing scheme. *ApJ*, 414:1–19, September 1993. doi: 10.1086/173051.
- E. E. Salpeter. The Luminosity Function and Stellar Evolution. *ApJ*, 121: 161, January 1955. doi: 10.1086/145971.
- Scalasca User Guide. Scalasca 1.4 user guide, 2011. <http://www2.fz-juelich.de/jsc/datapool/scalasca/UserGuide.pdf>.
- A. Serna, J.-M. Alimi, and J.-P. Chieze. Adaptive Smooth Particle Hydrodynamics and Particle-Particle Coupled Codes: Energy and Entropy Conservation. *ApJ*, 461:884, April 1996. doi: 10.1086/177109.
- B. Smith, S. Sigurdsson, and T. Abel. Metal cooling in simulations of cosmic structure formation. *MNRAS*, 385:1443–1454, April 2008. doi: 10.1111/j.1365-2966.2008.12922.x.
- V. Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364: 1105–1134, December 2005. doi: 10.1111/j.1365-2966.2005.09655.x.
- V. Springel and L. Hernquist. Cosmological smoothed particle hydrodynamics simulations: the entropy equation. *MNRAS*, 333:649–664, July 2002. doi: 10.1046/j.1365-8711.2002.05445.x.
- V. Springel, N. Yoshida, and S. D. M. White. GADGET: a code for collisionless and gasdynamical cosmological simulations. *NewA*, 6:79–117, April 2001. doi: 10.1016/S1384-1076(01)00042-2.
- J. G. Stadel. *Cosmological N-body simulations and their analysis*. PhD thesis, UNIVERSITY OF WASHINGTON, 2001.
- M. Steinmetz and S. D. M. White. Two-body heating in numerical galaxy formation experiments. *MNRAS*, 288:545–550, July 1997.
- J. M. Stone and M. L. Norman. ZEUS-2D: A radiation magnetohydrodynamics code for astrophysical flows in two space dimensions. I - The

- hydrodynamic algorithms and tests. *ApJS*, 80:753–790, June 1992. doi: 10.1086/191680.
- R. S. Sutherland and M. A. Dopita. Cooling functions for low-density astrophysical plasmas. *ApJS*, 88:253–327, September 1993. doi: 10.1086/191823.
- M. Tegmark, J. Silk, M. J. Rees, A. Blanchard, T. Abel, and F. Palla. How Small Were the First Cosmological Objects? *ApJ*, 474:1, January 1997. doi: 10.1086/303434.
- K. Thornton, M. Gaudlitz, H.-T. Janka, and M. Steinmetz. Energy Input and Mass Redistribution by Supernovae in the Interstellar Medium. *ApJ*, 500:95, June 1998. doi: 10.1086/305704.
- Valgrind User Manual. Valgrind user manual, 2011. <http://valgrind.org/docs/manual/manual.html>.
- J. W. Wadsley, J. Stadel, and T. Quinn. Gasoline: a flexible, parallel implementation of TreeSPH. *NewA*, 9:137–158, February 2004. doi: 10.1016/j.newast.2003.08.004.
- M. S. Warren and J. K. Salmon. A parallel, portable and versatile treecode. In D. H. Bailey, P. E. Bjorstad, J. R. Gilbert, M. V. Mascagni, R. S. Schreiber, H. D. Simon, V. J. Torczon, & L. T. Watson, editor, *Parallel Processing for Scientific Computing*, page 319, 1995.
- J. C. Weingartner and B. T. Draine. Dust Grain-Size Distributions and Extinction in the Milky Way, Large Magellanic Cloud, and Small Magellanic Cloud. *ApJ*, 548:296–309, February 2001. doi: 10.1086/318651.
- M. Wetzstein, A. F. Nelson, T. Naab, and A. Burkert. Vine—A Numerical Code for Simulating Astrophysical Systems Using Particles. I. Description of the Physics and the Numerical Methods. *ApJS*, 184:298–325, October 2009. doi: 10.1088/0067-0049/184/2/298.
- P. R. Williams, D. K. Churches, and A. H. Nelson. On Constraining the Smoothed Particle Hydrodynamics Kernel Radius in Galaxy Simulations. *ApJ*, 607:1–19, May 2004. doi: 10.1086/383256.
- N. Yoshida, T. Abel, L. Hernquist, and N. Sugiyama. Simulations of Early Structure Formation: Primordial Gas Clouds. *ApJ*, 592:645–663, August 2003. doi: 10.1086/375810.

Appendix A

A.1 Flags compiler

In this section we will briefly describe some IBM XLF compiler options. The options and the names are architecture-specific and language, but the structure, in general, is the same also for other compilers and other architectures. Fundamental rule is to always read the manuals for the compiler in use.

- `-qhot`: enables complex transformations (high-order transformation) of as loop blocking, unrolling;
- `-qarch= pwr5, pwr6`: select the architecture on which will run the resulting executable (in this case on Power5 and Power6);
- `-qtune= pwr3, pwr4`: select the chip on which will have to turning the resulting executable;
- `-qcache= (auto,...)`: defines the characteristics of cache (size, latencies) the machine on which will run the resulting executable;
- `-qipa= (.....)`: enables inter-procedural analysis (IPA), which allows to study and optimize at the level of different files and procedures;
- `-O2,-O`: local optimization, it is usually a compromise between the speed of compilation, optimization and size of the executable (default level);
- `-O3`: memory-intensive optimizations can alter the semantics of the program, and thus the results;
- `-O4`: series of aggressive optimizations (`-qhot`, `-qipa`, `-qarch=auto`, `-qtune=auto`, `-qcache=auto`);
- `-O5`: as the previous case but with much deeper interprocedural analysis (`-qipa=level2`), and is usually a very time consuming.

- -qpdf1/2: this will automatically optimize the code with the feedback of profiling. With-qpdf1 produces a code that keeps track of the instruction stream. With qpdf2-profiling using the information recorded for subsequent optimization, targeted on the specific case executed.
- -WF, Dmacro defines a macro for the preprocessor
- -q32/-q64 Compiling 32-bit/64-bit MAX 2^{32} bytes (2 GB) or 2^{64} bytes addressable.
- -C check array bounds
- -g O0 trace error
- -qflttrap=enable:invalid:nanq:overflow:underflow:zerodivide trap floating point exceptions
- -qsigtrap=xl_trcedump abort + core + death row

As an example, the following list of flags are customarily used to compile EvoLunder different compilers:

INTEL FORTRAN

DEBUG:

-O0 -g -traceback -check bounds -fpe0 -warn unused

PRODUCTION:

-O3 -ip -ipo -unroll -pc32

GNU FORTRAN

DEBUG:

-O0 -g -fbacktrace -fbounds-check -ffpe-trap=invalid,zero,overflow,underflow

PRODUCTION:

-O3 -funroll-loops -fipa-cp -fipa-matrix-reorg -fipa-type-escape -fipa-pta

IBM FORTRAN

DEBUG:

-C -O0 -qsource -qnosave -qarch=pwr6 -qtune=pwr6 -qcache=auto
-qfullpath -qflttrap=enable:invalid:nanq:overflow:underflow:zerodivide
-qsigtrap=xl_trce

PRODUCTION:

-O3 -q64 -qstrict -qhot -qarch=pwr6 -qtune=pwr6 -qcache=auto
-qipa=partition=large:inline=auto:level=2

A.2 Machines and tools used

A.2.1 Machine

- The IBM P575 Power 6 (codename at CINECA "SP6")
 Processor Type: IBM Power6, 4.7 GHz
 Operating System: AIX 6
 Compilers: IBM XL Fortran (xlf90)
- Monster (cluster dipastro)
 Processor Type: Intel(R) Xeon(R) CPU E5430 @ 2.66GHz
 Operating System: CentOS
 Compilers: Intel Fortran (ifort)
- BUD (desktop dipastro)
 Processor Type: Intel(R) Core(TM)2 Quad CPU Q9450 @ 2.66GHz
 Operating System: Ubuntu
 Compilers: GNU Fortran (gFortran)
- HAL9002 (My laptop)
 Processor Type: Intel(R) Core(TM)2 Duo CPU T9550 @ 2.66GHz
 Operating System: Ubuntu
 Compilers: GNU Fortran (gFortran)

A.2.2 Tools

- SP6: SCALASCA, Gprof/Xprofiler;
- Monster: Intel Trace Analyzer and Collector;
- BUD and HAL9002: Gprof/kprof, Cachegrind/kcachegrind.

A.3 List of simulations

Many simulations have computed for this Thesis. Most of them were carried out at CINECA on the basis of applications for CPU time. Others have benefitted of C ISCRA CPU time left over by another research project. A large fraction have been carried out on local machines such as MONSTER, BUD and HAL9002. In general we have considered and mentioned only the simulations that are relevant for the purposes of the this thesis. Finally we recall that all the simulations named HDHM have 64883 particles of dark matter, 57280 of gas, and 35083 of stars. In the Tables below we list the simulations in use together with some details for the sake of an easy identification.

Table A.1: Simulations on MONSTER with the old code.

Group	Name	Profiler	Npart	Ntask	Nstep
ITAC_OLD	18_32_proc	Traceanalyzer	18^3	32	200
ITAC_OLD	18_16_proc	Traceanalyzer	18^3	16	200
ITAC_OLD	18_8_proc	Traceanalyzer	18^3	8	200
ITAC_OLD	18_4_proc	Traceanalyzer	18^3	4	200
ITAC_OLD	18_2_proc	Traceanalyzer	18^3	2	200

Table A.2: Simulations on MONSTER with the new code.

Group	Name	Profiler	Npart	Ntask	Nstep
ITAC_NEW	18_32_proc	Traceanalyzer	18^3	32	200
ITAC_NEW	18_16_proc	Traceanalyzer	18^3	16	200
ITAC_NEW	18_8_proc	Traceanalyzer	18^3	8	200
ITAC_NEW	18_4_proc	Traceanalyzer	18^3	4	200
ITAC_NEW	18_2_proc	Traceanalyzer	18^3	2	200

Table A.3: BUD/HAL9002: only the most relevant ones; 18^3 particles on 1 processor.

Group	Name	Profiler	Nstep
OPTIM	18_serial_O3	CACHEGRIND	30
OPTIM	NEW_CODEII	CACHEGRIND	30
OPTIM	NEW_CODEII	GPROF	30
OPTIM	TEST_PERIODISM_O3	CACHEGRIND	1000
PROFILING	TEST_PERIODISM	GPROF	1000
PROFILING	TEST_fper_pper	GPROF	1000
PROFILING	TEST_softphi_grav	GPROF	1000

Table A.4: Simulations at CINECA ISCRA

Group	PROFILER	Npart	Ntask	Task_Affinity	Nstep
SCAL/1_SIM	none	46^3	32	core	7990
SCAL/1_SIM	none	46^3	16	core	5890
SCAL/1_SIM	none	46^3	8	core	4620
SCAL/1_SIM	none	46^3	4	core	1950
SCAL/1_SIM	none	46^3	2	core	1380
SCAL/1_SIM	none	46^3	1	core	710
SCAL/1_PROC	none	74^3	1	core	50
SCAL/1_PROC	none	58^3	1	core	190
SCAL/1_PROC	none	46^3	1	core	710
SCAL/1_PROC	none	30^3	1	core	3860
SCAL/1_PROC	none	24^3	1	core	8630
SCAL/1_PROC	none	18^3	1	core	21740
262144	Gprof	64^3	32	core	200
262144	Gprof	64^3	32	cpu	200
262144	SCALASCA	64^3	32	core	200
262144	SCALASCA	64^3	32	cpu	200
405224	Gprof	74^3	32	cpu	500
405224	SCALASCA	74^3	32	cpu	500
HDMM	Gprof	-	16	core	500
HDMM	Gprof	-	16	cpu	500
HDMM	SCALASCA	-	16	core	500
HDMM	SCALASCA	-	16	cpu	500

Table A.5: Simulations at CINECA CNE0IN05

Gruppo	PROFILER	Npart	Ntask	Task_Affinity	Nstep
FORTE	SCALASCA	46^3	1	cpu	300
FORTE	SCALASCA	46^3	2	cpu	300
FORTE	SCALASCA	46^3	4	cpu	300
FORTE	SCALASCA	46^3	8	cpu	300
FORTE	SCALASCA	46^3	16	cpu	300
FORTE	SCALASCA	46^3	32	cpu	300
FORTE	SCALASCA	46^3	64	cpu	300
FORTE	SCALASCA	46^3	128	cpu	300
FORTE	SCALASCA	18^3	1	cpu	300
FORTE	SCALASCA	18^3	2	cpu	300
FORTE	SCALASCA	18^3	4	cpu	300
FORTE	SCALASCA	18^3	8	cpu	300
FORTE	SCALASCA	18^3	16	cpu	300
FORTE	SCALASCA	18^3	32	cpu	300
FORTE	SCALASCA	18^3	64	cpu	300
FORTE	SCALASCA	18^3	128	cpu	300
FORTE	SCALASCA	HDMM	1	cpu	300
FORTE	SCALASCA	HDMM	2	cpu	300
FORTE	SCALASCA	HDMM	4	cpu	300
FORTE	SCALASCA	HDMM	8	cpu	300
FORTE	SCALASCA	HDMM	16	cpu	300
FORTE	SCALASCA	HDMM	32	cpu	300
FORTE	SCALASCA	HDMM	64	cpu	300
FORTE	SCALASCA	HDMM	128	cpu	300
DEBOLE	SCALASCA	18^3	32	cpu	10
DEBOLE	SCALASCA	24^3	32	cpu	10
DEBOLE	SCALASCA	30^3	32	cpu	10
DEBOLE	SCALASCA	46^3	32	cpu	10
DEBOLE	SCALASCA	58^3	32	cpu	10
DEBOLE	SCALASCA	74^3	32	cpu	10

EvoL: The new Padova T-SPH parallel code for cosmological simulations - I. Basic code: gravity and hydrodynamics

Emiliano Merlin¹, Umberto Buonomo¹, Tommaso Grassi¹, Lorenzo Piovan¹ & Cesare Chiosi¹

¹ Department of Astronomy, University of Padova, Vicolo dell'Osservatorio 3, 35122 Padova, Italy

e-mail: emiliano.merlin@unipd.it; cesare.chiosi@unipd.it; umberto.buonomo@unipd.it; tommaso.grassi@unipd.it; lorenzo.piovan@unipd.it

Received: November 2009; Accepted:

ABSTRACT

Context. We present EvOL, the new release of the Padova N-body code for cosmological simulations of galaxy formation and evolution. In this paper, the basic Tree + SPH code is presented and analysed, together with an overview on the software architectures.

Aims. EvOL is a flexible parallel Fortran95 code, specifically designed for simulations of cosmological structure formation on cluster, galactic and sub-galactic scales.

Methods. EvOL is a fully Lagrangian self-adaptive code, based on the classical Oct-tree by Barnes & Hut (1986) and on the Smoothed Particle Hydrodynamics algorithm (SPH, Lucy 1977). It includes special features such as adaptive softening lengths with correcting extra-terms, and modern formulations of SPH and artificial viscosity. It is designed to be run in parallel on multiple CPUs to optimize the performance and save computational time.

Results. We describe the code in detail, and present the results of a number of standard hydrodynamical tests.

Key words. Methods: N-body simulations

1. Introduction

The importance of numerical simulations in modern science is constantly growing, because of the complexity, the multi-scaling properties, and the non-linearity of many physical phenomena. When analytical predictions are not possible, we are forced to compute the evolution of physical systems numerically. Typical examples in astrophysical context are the problems of structure and galaxy formation and evolution. Over the past two decades, thanks to highly sophisticated cosmological and galaxy-sized numerical simulations a number of issues concerning the formation of cosmic systems and their evolution along the history of the Universe have been clarified. However, an equivalent number of new questions have been raised, so that complete understanding of how galaxies and clusters formed and evolved along the Hubble time is still out of our reach. This is especially true at the light of many recent observations that often appear at odds with well established theories (see for instance the long debated question of the monolithic versus hierarchical mechanism of galaxy formation and their many variants) and to require

new theoretical scenarios able to match the observational data.

To this aim, more and more accurate and detailed numerical simulations are still needed. They indeed are the best tool to our disposal to investigate such complex phenomena as the formation and evolution of galaxies within a consistent cosmological context. A number of astrophysical codes for galaxy-sized and cosmological simulations are freely and publicly available. They display a huge range of options and functions; each of them is best suited for a set of particular problems and experiments, and may suffer one or more drawbacks. Among the best known, we recall FLASH (Fryxell et al. 2000), GADGET (Springel et al. 2001) and its second release GADGET2 (Springel 2005), GASOLINE (Wadsley et al. 2004), HYDRA (Couchman et al. 1995), ENZO (Norman et al. 2007), and VINE (Wetzstein et al. 2008).

EvOL is the new release of the Padova N-body code (PD-TSPH, Carraro et al. 1998; Lia et al. 2002; Merlin & Chiosi 2007). It is a flexible, fully Lagrangian, parallel, and self-adaptive N-body code, written in Fortran95 in a straightforward and user-friendly format.

Send offprint requests to: E. Merlin

EvoL describes the dynamical evolution of a system of interacting discrete masses (particles) moving under the mutual gravitational forces and/or the gravitational action of external bodies, plus, when appropriate, under mutual hydrodynamical interactions. Such particles can represent real discrete bodies or a fluid contained in volume elements of suitable size. A numerical simulation of a physical system follows the temporal evolution of it using small but finite *time-steps* to approximate the equations of motion to a finite-differences problem. In the Lagrangian description, no reference grid is superposed to the volume under consideration, while the particles move under their mutual (or external) interactions. Each particle carries a mass, a position, a velocity, plus (when necessary) a list of physical features such as density, temperature, chemical composition, etc.

To simulate the dynamical evolution of a region of the Universe, one has to properly model the main different material components, namely Dark Matter, Gas, and Stars), representing them with different species of particles. Moreover, a physically robust model for the fundamental interactions (in addition to gravity) is required, at the various scales of interest. Finally, a suitable cosmological framework is necessary, together with a coherent setting of the boundary conditions and with an efficient algorithm to follow the temporal evolution of the system. EvoL is designed to respond to all of these requirements, improving upon the previous versions of the Padova Tree-SPH code under many aspects.¹ In the following sections, the main features of EvoL are presented in detail, together with a review of some general considerations about the adopted algorithms whenever appropriate.

This paper presents the basic features of the code; namely, the Tree-SPH (i.e. Oct-Tree plus Smoothing Particle Hydrodynamics) formalism, its implementation in the parallel architecture of the code, and the results of a number of classic hydrodynamic tests. The non-standard algorithms (e.g. radiative cooling functions, chemical evolution, star formation, energy feedback) will be presented in a following companion paper (Merlin et al. 2009, in preparation).

The plan of the paper is as follows. In Sect. 2 we describe the aspects of the code related to the gravitational interaction. In Sect. 3 we deal with the hydrodynamical treatment of fluids and its approximation in the SPH language. Sect. 4 illustrates some aspects related to the integration technique, such as the time steps, the periodic

boundary conditions, and the parallelization of the code. Sect. 5 contains and discusses numerous tests aimed at assessing the performance of the code. Sect. 6 summarizes some concluding remarks. Finally, Appendices A and B contain some technical details related to the N-dimensional spline kernel (Appendix A) and the equations of motion and energy conservation (Appendix B).

2. Description of the code: Gravity

Gravity is the leading force behind the formation of cosmic structures, on many scales of interest, from galaxy clusters down to individual stars and planets. Classical gravitation is a well understood interaction. As long as General Relativity, Particle Physics or exotic treatments such as Modified Dynamics are not considered², it is described by Newton's law of universal gravitation:

$$\mathbf{F}_{ij} = G \frac{m_i m_j}{|\mathbf{r}_j - \mathbf{r}_i|^3} (\mathbf{r}_j - \mathbf{r}_i), \quad (1)$$

where G is the gravitational constant. Given a density field $\rho(\mathbf{r})$, the gravitational potential Φ is obtained via Poisson equation:

$$\nabla^2 \Phi(\mathbf{r}) = 4\pi G \rho(\mathbf{r}), \quad (2)$$

and $\mathbf{F}_{ij} = -\nabla \Phi(\mathbf{r})$.

The gravitational force exerted on a given body (i.e., particle) by the whole system of bodies within a simulation can be obtained by the vectorial summation of all the particles' contributions (without considering, for the moment, the action of the infinite region external to the computational volume; this can indeed be taken into account using periodic boundary conditions, see Sect. 4.3). This is simply the straightforward application of Eq. 1. Anyway, in practice this approach is not efficient, and may also lead to artificial divergences, as explained in the following Sections.

2.1. Softening of the gravitational force

Close encounters between particles can cause numerical errors, essentially because of the time and mass resolution limits. Moreover, when dealing with continuous fluids rather than with single, isolated objects, one tries to solve Eq. 2 rather than Eq. 1, and must therefore try to model a *smooth* density field given a distribution of particles with mass. In addition, dense clumps of particles may also steal large amounts of computational time. To cope with all these issues, it is common practice to *soften* the gravitational force between close pairs of bodies: if the distance between two particles becomes smaller than a suitable *softening length* ϵ , the force exerted on each body is corrected and progressively reduced to zero with decreasing distance.

¹ The core of the old PD-TSPH code was written during the '90s by C. Chiosi, G. Carraro, C. Lia and C. Dalla Vecchia (Carraro et al. 1998; Lia et al. 2002). Over the years, many researchers added their contribution to the development of the code. In its original release, the PD-TSPH was a basic Tree + SPH code, written in Fortran90, conceptually similar to TREE-SPH by Hernquist & Katz (1989). Schematically, PD-TSPH used an early formulation of SPH (Benz 1990) to solve the equations of motion for the gas component, and the Barnes & Hut (1986) Tree algorithm to compute the gravitational interactions.

² A relativistic formulation of both gravitational and hydrodynamical interactions is possible (for a general summary see e.g. Rosswog 2009), but is generally unessential in problems of cosmological structure formation.

Different forms of the force softening function can be used. A possible expression is given by the following formula:

$$\frac{F(\mathbf{r}, \epsilon)}{Gm_i} = m_j \frac{\mathbf{r}}{|\mathbf{r}|} \times \quad (3)$$

$$\begin{cases} 1/\epsilon^2 [\frac{4}{3}u - \frac{6}{5}u^3 + \frac{1}{2}u^4] & \text{if } 0 \leq u < 1, \\ 1/\epsilon^2 [\frac{8}{3}u - 3u^2 + \frac{6}{5}u^3 - \frac{1}{6}u^4 - \frac{1}{15u^2}] & \text{if } 1 \leq u < 2, \\ 1/|\mathbf{r}|^2 & \text{if } u \geq 2, \end{cases}$$

where $\mathbf{r}$ is the distance between particles i and j , and $u \equiv |\mathbf{r}|/\epsilon$. This expression for the force softening corresponds (via the Poisson equation) to a density distribution *kernel* function proposed by Monaghan & Lattanzio (1985) and widely adopted in Smoothed Particles Hydrodynamics algorithms (see Sect. 3)³:

$$W(r, \epsilon) = \frac{1}{\pi h^3} \times \begin{cases} 1 - \frac{3}{2}u^2 + \frac{3}{4}u^3 & \text{if } 0 \leq u < 1, \\ \frac{1}{4}(2 - u)^3 & \text{if } 1 \leq u < 2, \\ 0 & \text{if } u \geq 2. \end{cases} \quad (4)$$

Note that in Eq. 4 the softening length ϵ is assumed to be the same for the two particles i and j . In the more general situation in which each particle carries its own ϵ , a symmetrisation is needed to ensure energy and momentum conservation: this can be achieved either using $\epsilon = \bar{\epsilon}_{ij} = (\epsilon_i + \epsilon_j)/2$, or by symmetrizing the softened force after computing it with the two different values ϵ_i and ϵ_j .

The softening lengths can be fixed in space and time, or may be let vary with local conditions. If the softening length is kept constant, the choice of its numerical value is of primary importance: for too small a softening length it will result in noisy force estimates, while for too large a value it will systematically bias the force in an unphysical manner (Merritt 1996; Romeo 1998; Athanassoula et al. 2000; Price & Monaghan 2007, and see also Sect. 5.9). Unfortunately, the “optimal” value for the softening length depends on the particular system being simulated, and in general it is not known a priori. A standard solution is to assign to each particle a softening length proportional to its mass, keeping it fixed throughout the whole simulation⁴, or letting it vary with time, or redshift, if a cosmological background is included. A clear advantage of keeping ϵ fixed in space is that energy is naturally conserved, but on the other hand the smallest

resolvable length scale is fixed at the beginning of the simulation and remains the same in the whole spatial domain independently of the real evolution of the density field. A collapsing or expanding body may quickly reach a size where the flow is dominated by the softening in one region, while in another the flow may become unphysically point-like.

Obviously, the accuracy can be greatly improved if ϵ is let vary according to the local particle number density (see e.g. Dehnen 2001). Moreover, in principle, if particles in a simulation are used to sample a continuous fluid (whose physics is determined by the Navier-Stokes or the Boltzmann equations) the properties of such points should always change accordingly to the local properties of the fluid they are sampling, to optimize the self-adaptive nature of Lagrangian methods. On the other hand, if particles represent discrete objects (single stars, or galaxies in a cluster, etc.), their softening lengths might perhaps be considered an intrinsic property of such objects and may be kept constant, depending on their mass and not on the local density of particles. In cosmological and galaxy-sized simulations, gas and Dark Matter particles are point-masses sampling the underlying density field, and stellar particles represent clusters of thousands of stars prone to the gravitational action of the nearby distribution of matter; thus a fully adaptive approach seems to be adequate to describe the evolution of these fluids. However, it can be easily shown that simply letting ϵ change freely would result in a poor conservation of global energy and momentum, even if in some cases the errors could be small (see e.g. Price & Monaghan 2007; Wetzstein et al. 2008).

To cope with this, EvoL allows for the adaptive evolution of individual softening lengths, but includes in the equations of motion of particles self-consistent correcting additional terms. Such terms can be derived (see Price & Monaghan 2007) starting from an analysis of the Lagrangian describing a self-gravitating, softened, collisionless system of N particles, which is given by⁵

$$L = \sum_{i=1}^N m_i \left(\frac{1}{2} v_i^2 - \Phi_i \right), \quad (5)$$

where Φ is the gravitational potential of the i -th particle,

$$\Phi(\mathbf{r}_i) = -G \sum_{j=1}^N m_j \phi(|\mathbf{r}_{ij}|, \epsilon_i), \quad (6)$$

and ϕ is a softening kernel ($\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$). Assuming a density distribution described by the standard spline kernel Eq. 4, the latter becomes

³ Throughout the paper, we only use the 3-dimensional formulation of kernels. See Appendix A for the 1-D and 2-D forms.

⁴ Recently, Shirokov (2007) pointed out that since particles are of unequal mass, and hence unequal softening lengths, one should actually compute the pairwise gravitational force and potential by solving a double integral over the particle volumes. Therefore, the computation of the interactions using the classic approach is likely not accurate. Given the practical difficulty in evaluating those integrals, they also provide a numerical approximation.

⁵ A clear advantage of using the Lagrangian to derive the equations of motion is that, provided the Lagrangian is appropriately symmetrised, momentum and energy conservation are guaranteed. Note that this derivation closely matches the one described in Sect. 3 to derive the variational formulation of the SPH formalism and the so-called ∇h correcting terms.

$$\phi(\mathbf{r}, \epsilon) = \begin{cases} 1/\epsilon [\frac{2}{3}u^2 - \frac{3}{10}u^4 + \frac{1}{10}u^5 - \frac{7}{5}] & \text{if } 0 \leq u < 1, \\ 1/\epsilon [\frac{4}{3}u^2 - u^3 + \frac{3}{10}u^4 - \frac{1}{30}u^5 - \frac{8}{5} + \frac{1}{15u}] & \text{if } 1 \leq u < 2, \\ -1/|\mathbf{r}| & \text{if } u \geq 2, \end{cases} \quad (7)$$

where again $u \equiv |\mathbf{r}|/\epsilon$. Note that the force softening, Eq. 4, is obtained taking the spatial derivative of Eq. 8.

The equations of motion are obtained from the Euler-Lagrange equations,

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \mathbf{v}_i} \right) - \frac{\partial L}{\partial \mathbf{r}_i} = 0, \quad (8)$$

which give

$$m_i \frac{d\mathbf{v}_i}{dt} = \frac{\partial L}{\partial \mathbf{r}_i}. \quad (9)$$

The gravitational part of the Lagrangian is

$$L_{grav} = - \sum_{j=1}^N m_j \Phi_j = - \frac{G}{2} \sum_j \sum_k m_j m_k \phi_{jk}(\epsilon_j) \quad (10)$$

where for the sake of clarity $\phi_{ij}(\epsilon_i) = \phi(|\mathbf{r}_{ij}|, \epsilon_i)$; swapping indices, the partial derivative $\partial L_{grav}/\partial \mathbf{r}_i$ results

$$\frac{\partial L_{grav}}{\partial \mathbf{r}_i} = - \frac{G}{2} \sum_j \sum_k m_j m_k \left[\frac{\partial \phi_{jk}(\epsilon_j)}{\partial |\mathbf{r}_{jk}|} \Big|_{\epsilon} \frac{\partial |\mathbf{r}_{jk}|}{\partial \mathbf{r}_i} + \frac{\partial \phi_{jk}(\epsilon_j)}{\partial \epsilon_j} \Big|_r \frac{\partial \epsilon_j}{\partial \mathbf{r}_i} \right], \quad (11)$$

where

$$\frac{\partial |\mathbf{r}_{jk}|}{\partial \mathbf{r}_i} = \frac{\mathbf{r}_j - \mathbf{r}_k}{|\mathbf{r}_j - \mathbf{r}_k|} (\delta_{ji} - \delta_{ki}). \quad (12)$$

To obtain the required terms in the second addend on the right part of Eq. 12, the softening length must be related to the particle coordinates so that $\epsilon = \epsilon(n)$, where n is the number density of particles at particle i location. To this aim, one can start from the general interpolation rule that any function of coordinates $A(\mathbf{r})$ can be expressed in terms of its values at a disordered set of points, and the integral interpolating the function $A(\mathbf{r})$ can be defined by

$$A_{int}(\mathbf{r}) = \int A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', \epsilon) d\mathbf{r}', \quad (13)$$

where W is the density kernel Eq. 4. This is the same rule at the basis of the SPH scheme (see Sect. 3) ⁶.

⁶ Note that in principle W should be defined over the whole space, as in the first SPH calculations by Gingold & Monaghan (1977) where a Gaussian-shaped kernel was adopted. For practical purposes, anyway, its domain is made compact putting $W = 0$ for distances greater than $\eta\epsilon$, with $\eta > 0$, from the position of the active particle.

Approximating the integral to a summation over particles for practice purposes⁷, one obtains

$$A_{sum}(\mathbf{r}_i) = A_i = \sum_j \frac{A_j}{n_j} W(\mathbf{r}_{ij}, \epsilon_i), \quad (14)$$

where i marks the “active” particle that lays at the position at which the function is being evaluated, and j are its neighbouring particles (or simply *neighbours*). The error by doing so depends on the disorder of particles and is in general $O(h^2)$ (Monaghan 1992). Thus,

$$n_i = \sum_j W(|\mathbf{r}_{ij}|, \epsilon_i), \quad (15)$$

where it has been put $A = n$; so, there is no need to know in advance the value of the density of neighbouring particles to compute the density of the active one.

To link ϵ and n , a safe prescription is to use

$$\epsilon_i = \eta \left(\frac{1}{n_i} \right)^{1/3}, \quad (16)$$

where η is a dimensionless parameter. With this law, the weighted number of particles within a softening sphere is tentatively held constant, i.e.

$$\frac{4}{3} (2\epsilon_i)^3 n_i = N_{nei,id}, \quad (17)$$

with $N_{nei,id} 3\pi(2\eta)^3$. Numerical experiments have shown that a safe choice is $1.2 \leq \eta \leq 1.5$, corresponding to $60 \leq N_{nei,id} \leq 110$ (Price & Monaghan 2007).

The term needed in Eq. 12 is

$$\frac{\partial \epsilon_j}{\partial \mathbf{r}_i} = \frac{\partial \epsilon_j}{\partial n_j} \frac{\partial n_j}{\partial \mathbf{r}_i}. \quad (18)$$

The first factor is given by the derivative of Eq. 16,

$$\frac{\partial \epsilon_j}{\partial n_j} = - \frac{\epsilon_j}{3n_j}, \quad (19)$$

whereas the second factor is the spatial derivative of Eq. 15, which results:

$$\frac{\partial n_j}{\partial \mathbf{r}_i} = \frac{1}{\Upsilon_j} \sum_p \frac{\partial W_{jp}(\epsilon_j)}{\partial \mathbf{r}_i} (\delta_{ij} - \delta_{ip}), \quad (20)$$

⁷ Price & Monaghan (2007) use mass the density in place of number density to achieve the desired solutions. The formulation in terms of number density is equivalent and can be advantageous when dealing with particles of different masses. Also note that this is a reformulation of the well known SPH density summation, see Sect. 3.

where it was defined

$$\Upsilon_i = \left[1 - \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(\epsilon_i)}{\partial \epsilon_i} \right]. \quad (21)$$

Rearranging and putting $G \equiv 1$, one finally finds

$$\begin{aligned} \frac{\partial L_{grav}}{\partial \mathbf{r}_i} = & \quad (22) \\ & -m_i \sum_j m_j \left[\frac{\phi'_{ij}(\epsilon_i) + \phi'_{ij}(\epsilon_j)}{2} \right] \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} \\ & -m_i \sum_j m_j \frac{1}{2} \left[\frac{\xi_i}{\Upsilon_i} \frac{\partial W_{ij}(\epsilon_i)}{\partial \mathbf{r}_i} + \frac{\xi_j}{\Upsilon_j} \frac{\partial W_{ij}(\epsilon_i)}{\partial \mathbf{r}_j} \right], \end{aligned}$$

where

$$\xi_i = \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial \phi_{ij}(\epsilon_i)}{\partial \epsilon_i}. \quad (23)$$

The $\partial \phi / \partial \epsilon$ terms used in Eq. 23 can be tabulated together with the other kernel functions:

$$\begin{aligned} \frac{\partial \phi}{\partial \epsilon} = & \quad (24) \\ & \begin{cases} 1/\epsilon^2 [-2u^2 + \frac{3}{2}u^4 - \frac{3}{5}u^5 + \frac{7}{5}] & \text{if } 0 \leq u < 1, \\ 1/\epsilon^2 [-4u^2 + 4u^3 + \frac{3}{2}u^4 + \frac{1}{5}u^5 + \frac{8}{5}] & \text{if } 1 \leq u < 2, \\ 0 & \text{if } u \geq 2, \end{cases} \end{aligned}$$

where, as usual, $u \equiv |\mathbf{r}|/\epsilon$. Finally, the $\partial W / \partial \mathbf{r}$ and $\partial W / \partial \epsilon$ terms are easily obtained deriving the explicit expression of $W(|\mathbf{r}|, \epsilon)$.

In Eq. 22 (and Eq. 12), the first term is the classical gravitational interaction, whereas the second term is a new, extra-term which restores energy conservation for varying softening lengths. Since ξ is defined as a negative quantity for positive kernels, this term acts as a negative pressure, *increasing* the gravitational force. To obtain the ξ and Υ correcting terms, each particle i must perform a loop over the other particles, summing the contributions from the ones that satisfy the criterion $|\mathbf{r}_{ij}| \leq 2 \times \max(\epsilon_i, \epsilon_j)$ ⁸.

The relation between ϵ and n , defined by Eq. 16, leads to a non-linear equation which can be solved self-consistently for each particle. Price & Monaghan (2007)

⁸ Thus, *all* particles, and not only SPH particles, need to find their neighbours to compute these terms (see Sect. 2.2). While the Ω term described in Sect. 3 is only needed for the SPH algorithm, and therefore only SPH particles concur to compute it (considering only SPH neighbours in turn), the ξ and Υ gravitational terms are required for any kind of gravitating particle, and must be computed looping on *all* neighbouring particles, both SPH and not. The time lost in the neighbour searching routine for non-gaseous particles can be at least partially balanced adopting the individual time-stepping scheme (see Sect. 4.2.1). In this case, large softening lengths in low-density regions result in larger individual time-steps for particles belonging to those regions.

proposed an iterative method in which, beginning from a starting guess for both n and ϵ , the solution of the equation

$$f(\epsilon) = |n_i(\epsilon_i) - n_{sum,i}(\epsilon_i)| = 0, \quad (25)$$

where $n_{sum,i}(\epsilon_i)$ is the mass density computed via summation over neighbouring particles (Eq. 15) and $n_i(\epsilon_i)$ is the density obtained from Eq. 16, is searched by means of an iterative procedure, that can be solved adopting a classical bisection scheme, or more efficient routines such as the Newton-Raphson method. In this case, a loop is iterated until $|\epsilon_{i,new} - \epsilon_i|/\epsilon_{i,init} < \gamma_{TOL}$, where γ_{TOL} is a tolerance parameter $\sim 10^{-2} - 10^{-3}$, $\epsilon_{i,init}$ is the value of ϵ for the particle i at the beginning of the procedure, ϵ_i is its current value, and $\epsilon_{i,new} = \epsilon_i - f(\epsilon_i)/f'(\epsilon_i)$; the derivative of Eq. 25 is given by

$$f'(\epsilon_i) = \frac{\partial n_i}{\partial \epsilon_i} - \sum_j m_j \frac{\partial W_{ij}(\epsilon_i)}{\partial \epsilon_i} = -\frac{3n_i}{\epsilon_i} \Upsilon_i. \quad (26)$$

To increase the efficiency of this process, a predicted value for both ϵ and n can be obtained at the beginning of each time-step using a discretized formulation of the Lagrangian continuity equation,

$$\frac{dn}{dt} = -n(\nabla \cdot \mathbf{v}). \quad (27)$$

Such formulation can be obtained taking the time derivative of Eq. 15, which results

$$\frac{dn_i}{dt} = \frac{1}{\Upsilon_i} \sum_j (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W(\mathbf{r}_{ij}, \epsilon), \quad (28)$$

while

$$\frac{d\epsilon_i}{dt} = \frac{\partial \epsilon_i}{\partial n_i} \frac{dn_i}{dt}. \quad (29)$$

Combining Eq. 28 with Eq. 27, one can also see that the velocity divergence at i particle location is given by

$$\nabla \cdot \mathbf{v}_i = -\frac{1}{n_i \Upsilon_i} \sum_j (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W(\mathbf{r}_{ij}, \epsilon). \quad (30)$$

The adoption of this *adaptive softening length* formalism, with the correcting extra-terms in the equation of motion, results in small errors, always comparable in magnitude to the ones found with the “optimal” ϵ (see Price & Monaghan 2007, their Figs. 2, 3 and 4).

At the beginning of a simulation, the user can select whether he/she prefers to adopt the constant or the adaptive softening lengths formalism, switching on or off a suitable flag.

2.2. Hierarchical oct-tree structure

A direct summation of the contribution of all particles should in principle be performed for each particle at each time-step to correctly compute the gravitational interaction, leading to a computational cost increasing with N^2 (N being the number of particles). A convenient alternative to this unpractical approach are the so-called *tree* structures, in which particles are arranged in a hierarchy of groups or “cells”. In this way, when the force on a particle is computed, the contribution by distant groups of bodies can be approximated by their lowest multipole moments, reducing the computational cost to evaluate the total force to $O(N \log N)$. In the classical Barnes & Hut (1986) scheme, the computational spatial domain is hierarchically partitioned into a sequence of cubes, where each cube contains eight siblings, each with half the side-length of the parent cube. These cubes form the nodes of an oct-tree structure. The tree is constructed such that each node (cube) contains either a desired number of particles (usually one, or one per particles type - Dark Matter, gas, stars), or is progenitor of further nodes, in which case the node carries the monopole and quadrupole moments of all the particles that lie inside the cube. The force computation then proceeds by walking along the tree, and summing up the appropriate force contributions from tree nodes. In the standard tree walk, the global gravitational force exerted by a node of linear size l on a particle i is considered only if $r > l/\theta$, where r is the distance of the node from the active particle and θ is an *accuracy parameter* (usually ≤ 1): if a node fulfills the criterion, the tree walk along this branch can be terminated, otherwise it is “opened”, and the walk is continued with all its siblings. The contribution from individual particles is thus considered only when they are sufficiently close to the particle i .

To cope with some problems pointed out by Salmon & Warren (1994) about the worst-case behaviour of this standard criterion for commonly employed opening angles, Dubinski (1996) introduced the simple modification

$$r > l/\theta + \delta, \quad (31)$$

where the δ is the distance of the geometric center of a cell to its center of mass. This provides protection against pathological cases where the center of mass lies close to an edge of a cell.

In practice, this whole scheme only includes the monopole moment of the force exerted by distant groups of particles. Higher orders can be included, and the common practice is to include the quadrupole moment corrections (EvoL indeed includes them). Still higher multipole orders would result in a worst performance without significant gains in computational accuracy (McMillan & Aarseth 1993).

Note that if one has the total mass, the center of mass, and the weighted average softening length of a node of the oct-tree structure, the softened expression of the gravita-

tional force can be straightforwardly computed treating the cell as a single particle if the opening criterion is fulfilled but the cell is still sufficiently near for the force to need to be softened.

As first suggested by Hernquist & Katz (1989), the tree structure can be also used to obtain the individual lists of neighbours for each body. At each time step each (active) particle can build its list of interacting neighbouring particles while walking the tree and opening only sufficiently nearby cells, until nearby bodies are reached and linked.

3. Description of the code: Hydrodynamics

Astrophysical gaseous plasmas can generally be reasonably approximated to highly compressible, unviscous fluids, in which anyway a fundamental role is played by violent phenomena such as strong shocks, high energy point-like explosions and/or supersonic turbulent motions. EvoL follows the basic gas physics by means of the Smoothed Particles Hydrodynamics (SPH, Lucy 1977; Monaghan 1992), in a modern formulation based on the review by Rosswog (2009), to which the reader is referred for details. Anyway, some different features are present in our implementation and we summarize them in the following, along with a short review of the whole SPH algorithm.

To model fluid hydrodynamics by means of a discrete description of a continuous the fluid, in the SPH approach the properties of single particles are *smoothed* in real space through the kernel function W , and thus *weighted* by the contributions of neighbouring particles. In this way, the physical properties of each point in real space can be obtained by the summation over particles of their individual, discrete properties. Note that, on the contrary of what happens when softening the gravitational force (which is gradually reduced to zero within the softening sphere), in this case the smoothing sphere is the only “active” region, and only particles inside this region contribute to the computation of the local properties of the central particle.

Starting again from the interpolation rule described in Sect. 2.1, but replacing the softening length ϵ by a suitable *smoothing length* h , the integral interpolating the function $A(\mathbf{r})$ becomes

$$A_i(\mathbf{r}) = \int A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h) d\mathbf{r}', \quad (32)$$

(the kernel function W can be the density kernel Eq. 4).

The relative discrete approximation becomes

$$A_s(\mathbf{r}_i) = \sum_j \frac{A_j}{\rho_j} m_j W(\mathbf{r}_i - \mathbf{r}_j, h_i), \quad (33)$$

and the physical density of any SPH particle can be computed as

$$\rho_i = \sum_j m_j W(|\mathbf{r}|, h_i) \quad (34)$$

(here and throughout this Section, it is implied that all summations and computations are extended on SPH particles only).

A differentiable interpolating function can be constructed from its values at the interpolation points, i.e. the particles:

$$\nabla A(\mathbf{r}) = \sum_j \frac{A_j}{\rho_j} m_j \nabla W(\mathbf{r} - \mathbf{r}', h). \quad (35)$$

Anyway, better accuracy is found rewriting the formulae with the density *inside* operators and using the general rule

$$\rho \nabla A = \nabla(\rho A) - A \nabla \rho \quad (36)$$

(e.g., Monaghan 1992). Thus the divergence of velocity is customarily estimated by means of

$$\nabla \cdot \mathbf{v}_i = \sum_j m_j [(\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}] / \rho_i, \quad (37)$$

where $\nabla_i W_{ij}$ denotes the gradient of $W(\mathbf{r} - \mathbf{r}', h)$ with respect to the coordinates of a particle i .

The dynamical evolution of a continuous fluid is governed by the well known laws of conservation: the continuity equation which ensures conservation of mass, the Euler equation which represents the conservation of momentum, and the equation of energy conservation (plus a suitable equation of state to close the system). These equations must be written in discretized form to be used with the SPH formalism, and can be obtained by means of the Lagrangian analysis, following the same strategy used to obtain the gravitational accelerations.

The continuity equation is generally replaced by Eq. 34⁹. Like in the case of the gravitational softening length ϵ , the smoothing length h may in principle be a constant parameter, but the efficiency and accuracy of the SPH method is greatly improved adapting the resolution lengths to the local density of particles. A self-consistent method is to adopt the same algorithm described in Sect. 2.1, i.e. obtaining h from

$$\frac{dh}{dt} = \frac{dh}{dn} \frac{dn}{dt}, \quad (39)$$

where of course n is the local number density of SPH particles only, and relating h to n by requiring that a fixed number of kernel-weighted particles is contained within a smoothing sphere, i.e.

$$h_i = \eta \left(\frac{1}{n_i} \right)^{1/3}. \quad (40)$$

⁹ Alternatively, it could be written as

$$\frac{d\rho_i}{dt} = \sum_j m_j (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}, \quad (38)$$

which is advantageous in case the simulated fluid has boundaries or edges, but has the disadvantage of not conserving the mass exactly.

Note that in this case, while still obtaining the mass density by summing Eq. 34, one can compute the number density of particles at particle i 's location, i.e.:

$$n_i = \sum_j W(|\mathbf{r}_{ij}|, h_i), \quad (41)$$

which is clearly formally equivalent to Eq. 15¹⁰.

It is worth recalling here that in the first versions of SPH with spatially-varying resolution, the spatial variation of the smoothing length was generally not considered in the equations of motion, and this resulted in secular errors in the conservation of entropy. The inclusion of extra-correcting terms (the so-called ∇h terms) is therefore important to ensure the conservation of *both* energy and entropy. For example, Serna et al. (1996) studied the pressure-driven expansion of a gaseous sphere, finding that while the energy conservation is generally good even without the inclusion of the corrections (and sometimes get slightly worse if these are included!), errors up to $\sim 5\%$ can be found in the entropy conservation, while all this does not occur with the inclusion of the extra-terms. Although the effects of entropy violation in SPH codes are not completely clear and need to be analysed in much more detail, especially in simulations where galaxies are formed in a cosmological framework, there are many evidences that the problem must be taken into consideration. Alimi et al. (2003) have analysed this issue in the case of the collapse of isolated objects and have found that, if correcting terms are neglected, the density peaks associated with central cores or shock fronts are overestimated at a $\simeq 30\%$ level.

These ∇h terms were first introduced explicitly (Nelson & Papaloizou 1994), with a double-summation added to the canonical SPH equations. Later, an implicit formulation was obtained (Monaghan 2002; Springel & Hernquist 2002), starting from the Lagrangian derivation of the equations of motion and self-consistently obtaining correcting terms which accounts for the variation of h . Obviously such terms are formally similar to those obtained for the locally varying gravitational softening lengths (Sect. 2.1).

Following Monaghan (2002), the Lagrangian for non-relativistic fluid dynamics can be written as

$$L = \int \rho \left(\frac{1}{2} v^2 - u \right) dV, \quad (42)$$

(the gravitational part is not included here since SPH does not account for gravity), which in the SPH formalism becomes

$$L_{SPH} = \sum_j m_j \left[\frac{1}{2} v_j^2 - u_j \right]. \quad (43)$$

¹⁰ Some authors (e.g. Hu & Adams 2005; Ott & Schnetter 2003) proposed a different "number density" formulation of SPH in which $\rho_i = m_i \times n_i$, and n_i is obtained via Eq. 41. While this can help in resolving sharp discontinuities and taking into account the presence of multi-phase flows, it may as well lead to potentially disastrous results if mixed unequal mass particle are *not* intended to model density discontinuities but are instead used as mass tracers in a homogeneous field.

As already made for Eq. 8, the equations of motion of a particle can be obtained from the Euler-Lagrange equations, giving

$$\frac{d\mathbf{v}_i}{dt} = - \sum_j m_j \left(\frac{\partial u}{\partial \rho} \right) \frac{\partial \rho_j}{\partial \mathbf{r}_i}. \quad (44)$$

To obtain the term $\partial \rho_j / \partial \mathbf{r}_i$, one can note that using the density ρ given by Eq. 34 and letting the smoothing length vary according to Eq. 40 one gets

$$\frac{\partial \rho_j}{\partial \mathbf{r}_i} = \frac{1}{\Omega_j} \sum_k m_k \nabla_i W_{ik}(h_i) \delta_{ij} - m_i \nabla_j W_{ij}(h_j), \quad (45)$$

where δ_{ij} is the Kronecker delta function, $\nabla_i W$ is the gradient of the kernel function W taken with respect to the coordinates of particle i keeping h constant, and

$$\Omega_j = 1 - \frac{\partial h}{\partial \rho} \sum_k m_c \frac{\partial W_{jk}(h_j)}{\partial h_j} \quad (46)$$

is an extra-term that accounts for the dependence of the kernel function on the smoothing length. Note that Ω is formally identical to the Υ term introduced in Sect. 2, replacing ϵ with h . This is obvious since the underlying mathematics is exactly the same; both terms arise when the motion equations are derived from the Lagrangian formulation.

In case of particles with different mass, the term $\partial \rho_j / \partial \mathbf{r}_i$ is

$$\frac{\partial \rho_j}{\partial \mathbf{r}_i} = \sum_k m_k \nabla_j W_{ij}(h_j) \left[1 + \frac{\zeta_j / m_k}{\Omega_j^*} \right] (\delta_{ij} - \delta_{ik}), \quad (47)$$

where

$$\zeta_i = \frac{\partial h_i}{\partial n_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i} \quad (48)$$

and

$$\Omega_i^* = 1 - \frac{\partial h_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(h_i)}{\partial h_i}. \quad (49)$$

(Price 2009, private communication). These terms can be easily computed for each particle while looping over neighbours to obtain the density from Eq. 34.

The term $\partial u / \partial \rho$ in Eq. 44 can be derived from the first law of thermodynamics, $dU = dQ - PdV$, dropping the dQ term since only adiabatic processes are considered. Rewriting in terms of specific quantities, the volume V becomes volume “per mass”, i.e. $1/\rho$, and $dV = d(1/\rho) = -d\rho/\rho^2$. Thus,

$$du = \frac{P}{\rho^2} d\rho, \quad (50)$$

and, if no dissipation is present, the specific entropy is constant, so

$$\left(\frac{\partial u}{\partial t} \right)_s = \frac{P}{\rho^2}. \quad (51)$$

Inserting Eqs. 51 and 47 into Eq. 44, the equations of motion finally become

$$\frac{d\mathbf{v}_i}{dt} = \sum_j m_j \times \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i / m_j}{\Omega_i^*} \right) \nabla_i W_{ij}(h_i) + \frac{P_j}{\rho_j^2} \left(1 + \frac{\zeta_j / m_i}{\Omega_j^*} \right) \nabla_i W_{ij}(h_j) \right], \quad (52)$$

The equation for the thermal energy equation is

$$\frac{du_i}{dt} = \sum_j m_j \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i / m_j}{\Omega_i^*} \right) (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}(h_i) \right]. \quad (53)$$

It can be shown that if all SPH particles have the same mass, Eqs. 53 and 54 reduce to the standard Eqs. 2.10 and 2.23 in Monaghan (2002) ¹¹.

Equations 41 (density summation) and Eq. 40 form a non-linear system to be solved for h and n , adopting the same method described in Sect. 2.1.

A drawback of this scheme can be encountered when a sink of internal energy such as radiative cooling is included. In this case, very dense clumps of cold particles can form and remain stable for long periods of time. Since the contribution of neighbouring particles to the density summation (Eq. 41) is weighted by their distance from the active particle, if such clump is in the outskirts of the smoothing region then each body will give a very small contribution to the summation, and this will result in an unacceptably long neighbour list. In this cases, the adoption of a less accurate but faster method is appropriate. The easiest way to select h is to require that a constant number of neighbouring particles is contained within a sphere of radius ηh (perhaps allowing for small deviations). This type of procedure was generally adopted in the first SPH codes, and provided sufficiently accurate results. If the scheme in question is adopted (EvoL may

¹¹ As noted by Schuessler & Schmitt (1981), the gradient of the spline kernel Eq. 4 can lead to unphysical clustering of particles. To prevent this, Monaghan (2000) introduced a small artificial repulsive pressure-like force between particles in close pairs. In EvoL a modified form of the kernel gradient is instead adopted, as suggested by Thomas & Couchman (1992):

$$\frac{dW}{d\mathbf{r}} = -\frac{1}{4\pi} \times \begin{cases} 4 & \text{if } 0 \leq u < 2/3, \\ 3u(4-3u) & \text{if } 2/3 \leq u < 1, \\ 3(2-u)^2 & \text{if } 1 \leq u < 2, \\ 0 & \text{if } u \geq 2, \end{cases} \quad (54)$$

with the usual meaning of the symbols.

choose between the two algorithms), the terms ∇h are neglected, because the relation Eq. 40 is no longer strictly valid. In the following, we will refer to this scheme as to the *standard SPH scheme*, and to previous one based on the $n - h$ (or $\rho - h$ relation) as to the *Lagrangian SPH scheme*.

A test to check the ability of the different algorithms to capture the correct description of an (ideally) homogeneous density field has been performed, using particles of different masses mixed together in the domain $0 < x < 1, 0 < y < 0.1, 0 < z < 0.1$, to obtain the physical densities $\rho = 1$ if $x < 0.5$ and $\rho = 0.25$ if $x \geq 0.5$. To this aim, a first set of particles were displaced on a regular grid and were given the following masses:

$$m = 0.9 \times 10^{-6} \text{ for } x < 0.5; m = 2.25 \times 10^{-7} \text{ for } x \geq 0.5.$$

Then, a second set of particles were displaced on another superposed regular grid, shifted along all three dimensions with respect to the previous one by a factor $\Delta x = \Delta y = \Delta z = 5 \times 10^{-3}$. These particles were then assigned the following masses

$$m = 0.1 \times 10^{-6} \text{ for } x < 0.5; m = 0.25 \times 10^{-7} \text{ for } x \geq 0.5.$$

Then, four different schemes were used to compute the density of particles:

- standard SPH, mass density summation
- standard SPH, number density summation
- Lagrangian SPH, mass density summation
- Lagrangian SPH, number density summation

where the expressions “mass density” and “number density” refer to the different scheme adopted to compute the density of a particle, i.e. to Eq. 34 and 41, respectively. Looking at Fig. 1, it is clear that the discrete nature of the regular displacement of particles gives different results depending on the adopted algorithm. The mass density formulation is not able to properly describe the uniform density field: low-mass particles strongly underestimate their density, on both sides of the density discontinuity. The situation is much improved when a number density formulation is adopted. Little differences can be noted between the determinations using the constant neighbours scheme and the density- h relation.

3.1. On symmetrisation

To ensure the conservation of energy and momenta, a proper symmetrisation of the equations of motion is required. It is worth noticing that while in the first formulations of SPH such symmetrisation had to be imposed “ad hoc”, in the Lagrangian derivation it is naturally built-in without any further assumption.

Anyway, the symmetrisation is strictly necessary only when pairwise forces act on couples of particles. Thus, it is not needed when obtaining the density from Eq. 41 and 34, or when the internal energy of a particle is evolved according to Eq. 54. Indeed, symmetrisation of the energy equation may lead to unphysical negative temperatures. For

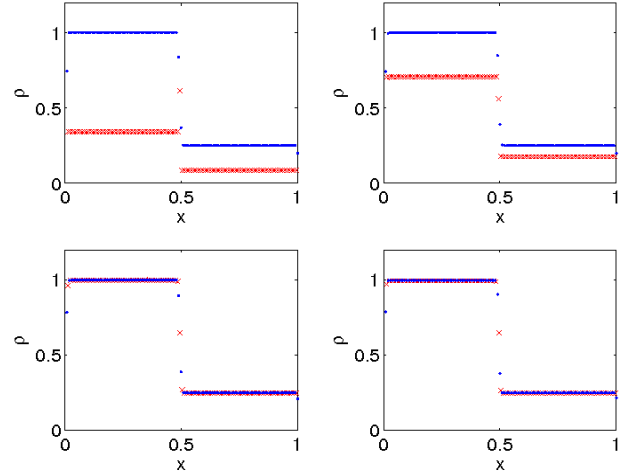


Fig. 1. Determination of the Initial density adopting four different SPH algorithms. Left to right, top to bottom: standard SPH, mass density summation; Lagrangian SPH, mass density summation; standard SPH, number density summation; Lagrangian SPH, number density determination. Blue dots: high mass particles; red crosses: low mass particles.

example when cold particles are pushed by a strong source of pressure: in this case, the mechanical expansion leads to a loss of internal energy which, if equally subdivided between the cold and hot phase, may exceed the small energy budget of cold bodies (see e.g. Springel & Hernquist 2002).

Since symmetrisation is not needed in the density summation algorithm, the smoothing length of neighbouring particles is not required to compute the density of a particle i . This allows an exact calculation simply gathering position and masses of neighbours, which are known at the beginning of the time-step. On the other hand, after the density loop each particle has its new smoothing length h and it is now possible to use these values in the acceleration calculations, where symmetrisation is needed. Anyway, while in the previous case only particles within $2h_i$ were considered neighbours of the particle i , in this latter case the symmetrisation scheme requires a particle j to be considered neighbour of particle i if $r_{ij} < 2 \times \max(h_i, h_j)$ ¹².

Note that with this scheme *two* routines of neighbour searching are necessary: the first one in the density summation, and the second in the force calculation. In practice, when the evaluation of density is being performed, a *gather* view of SPH is adopted (see Hernquist & Katz

¹² It would also be possible to compute the interactions by only considering neighbours within h_i (or ϵ_i). To do so, each particle should “give back” to its neighbours its contribution to the acceleration. Anyway, this approach is a more complicated with individual timesteps (where one has to compute inactive contributions to active particles), and in a parallel architecture. We therefore decided to adopt the fully symmetric scheme in this release of EvoL

1989); thus, each particle i searches its neighbours using only its own smoothing length h_i , until the convergence process is completed. In this way, if a particle finds too high or too low a number of neighbours (and therefore must change its smoothing length to repeat the search) it is not necessary to inform the other particles (including those belonging to other processors) about the change that is being made. This would in principle be necessary if a standard *gather/scatter* mode is adopted. Thus, each particle evaluates its density independently, adjusting its smoothing length on its own, and no further communication among processors is necessary, after the initial gathering of particles positions. Moreover, the search is initially performed using an effective searching radius which is slightly larger than the correct value $2h_i$. In this way, the neighbour list will include slightly more particles than effectively needed. However, if at the end of the loop, the smoothing length has to be increased by a small factor, it is not necessary to repeat the tree-walk to reconstruct the neighbour list. Of course, if the adaptive softening length scheme is adopted the tree-walk will be performed to upgrade h and ρ for SPH particles only, and ϵ and ρ_{tot} for all particles.

Then, when forces (hydrodynamical forces and, if the adaptive softening length scheme is adopted, gravitational forces too) are evaluated, a new search for neighbours must be made, in which a particle is considered neighbour to another one if the distance between the two is lower than the maximum between the two searching radii. During this second tree-walk, the gravitational force is evaluated together with the construction of the particle's neighbour list. Finally, correcting terms are computed for hydrodynamics equations of motion (∇h terms) and, if necessary, for gravitational interactions (if softening lengths are adaptive), summing the neighbouring particles' contributions.

3.2. Smoothing and softening lengths for SPH particles

SPH particles under self-gravity have two parameters characterizing their physical properties: the softening length ϵ , which tunes their gravitational interactions with nearby particles, and the smoothing length h , used to smooth their hydrodynamical features.

In principle there is no need for these two quantities to be related to one another, because they refer to two distinct actions, and have somehow opposite functions as noted above. However, Bate & Burkert (1997) claimed that a quasi-stable clump of gas could become un-physically stable against collapse if $\epsilon > h$, because in this case pressure forces would dominate over the strongly softened gravity, or on the other hand, if $\epsilon < h$, collapse on scales smaller than the formal resolution of SPH may be induced, causing artificial fragmentation. They recommend that gravitational and hydrodynamical forces should be resolved at the same scale length, imposing $\epsilon = h$ (requiring both of them to fixed, or ϵ to be adaptive like

h , with the introduction of the correcting terms described above). In other studies (e.g. Thacker & Couchman 2000) the softening length is fixed, but the smoothing length is allowed to vary imposing a lower limit close to the value of the softening length (this is a typical assumption in SPH simulations of galaxy formation).

Anyway, Williams et al. (2004) have studied the effects of such procedure, finding that in many hydrodynamical studies this can cause a number of problems. The adoption of large values of h can result in over-smoothed density fields, i.e. a decrease in the computed values of density ρ and of density gradients $\nabla\rho$, which in turn unphysically decreases the computed pressure accelerations in the Euler equation. Problems with angular momentum transfer may also arise. Therefore, they suggest to allow h to freely vary. This eventually yields improved accuracy and also, contrary to expectations, a significant saving in computational time (because the number of neighbours is kept fixed instead of largely increasing as it may happen in dense environments with fixed h). Moreover, they pointed out that, in contrast with the claims by Bate & Burkert (1997), the smoothing length should not be kept equal to the softening length, because in many physical situations (shocks for example) hydrodynamical forces dominate over gravity and likely need to be properly resolved on size scales smaller than the softening length.

Finally, a numerical problem arises when trying to keep $\epsilon = h$ in cosmological simulations. The presence of a Dark Matter component makes it impossible to keep constant the number of both hydrodynamical and gravitational neighbours. For example, a collapsed Dark Matter halo could retain few gaseous particles (because of shocks and/or stellar feedback), and these will thus have many gravitational neighbours but few hydrodynamic neighbours with a unique softening/smoothing length. The opposite problem could arise in the case of very dense clumps of cold gas that may form behind shocks, without a corresponding concentration of Dark Matter.

Thus, in EvoL both ϵ and h are let free to vary independently from one another. Future tests may be done trying to keep the two parameters linked, for example allowing for some variations in both their ratio ϵ/h (imposing it to be *around*, but not exactly, 1) and in the number of hydrodynamical and gravitational neighbours.

3.3. Discontinuities

3.3.1. Artificial viscosity

The SPH formalism in its straight form is not able to handle discontinuities. By construction, SPH smooths out local properties on a spatial scale of the order of the smoothing length h . Anyway, strong shocks are of primary importance in a number of astrophysical systems. Thus, to model a real discontinuity in the density and/or pressure field *ad hoc* dissipational, pressure-like terms must be added to the perfect fluid equations, to spread the discontinuities over the numerically resolvable length. This

is usually achieved by introducing an *artificial viscosity*, a numerical artifact that is not meant to mimic physical viscosity, but to reproduce on large scale the action of smaller, unresolved scale physics, with a formulation consistent with the Navier-Stokes terms.

Among the many formulations that have been proposed, the most commonly used is obtained adding a term Π_{ij} to the Eqs. 53 and 54, so that

$$\frac{d\mathbf{v}_i}{dt} = \sum_j m_j \times \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) \nabla_i W_{ij}(h_i) + \frac{P_j}{\rho_j^2} \left(1 + \frac{\zeta_j/m_i}{\Omega_j^*} \right) \nabla_i W_{ij}(h_j) + \Pi_{ij} \nabla_i \bar{W}_{ij} \right], \quad (55)$$

and

$$\frac{du_i}{dt} = \sum_j m_j \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}(h_i) + \frac{1}{2} \Pi_{ij} w_{ij} |\nabla_i \bar{W}_{ij}| \right]. \quad (56)$$

Here $w_{ij} = (\mathbf{v}_{ij} \cdot \mathbf{r}_{ij})/|\mathbf{r}_{ij}|$,

$$\Pi_{ij} = \begin{cases} \left(\frac{-\alpha c_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\rho_{ij}} \right) & \text{if } \mathbf{v}_{ij} \cdot \mathbf{r}_{ij} < 0, \\ 0 & \text{if } \mathbf{v}_{ij} \cdot \mathbf{r}_{ij} \geq 0, \end{cases} \quad (57)$$

$$\mu_{ij} = \frac{h \mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{\mathbf{r}_{ij}^2 + \eta^2}, \quad (58)$$

and

$$\nabla_i \bar{W}_{ij} = \frac{1}{2} \left[\nabla_i W_{ij}(h_i) \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) + \nabla_i W_{ij}(h_j) \left(1 + \frac{\zeta_j/m_i}{\Omega_j^*} \right) \right] \quad (59)$$

(Monaghan 1988). c_{ij} is the average sound speed, which is computed for each particle from a suitable equation of state (EoS), like the ideal monatomic gas EoS $P = (\gamma - 1)\rho u$, using $c_i = \sqrt{\gamma(\gamma - 1)u_i}$ (γ is the adiabatic index); the parameter η is introduced to avoid numerical divergences, and it is usually taken to be $\eta = 0.1h$. α and β are two free parameters, of order unity, which account respectively for the shear and bulk viscosity in the Navier-Stokes equation (Watkins et al. 1996), and for the von Neuman-Richtmyer viscosity, which works in high Mach number flows.

A more recent formulation is the so-called “signal velocity” viscosity (Monaghan 1997), which works better

for small pair separations between particles, being a first-order expression of h/r ; it reads

$$\Pi_{ij} = \frac{-\alpha v_{ij}^{sig} w_{ij}}{\rho_{ij}}, \quad (61)$$

with the *signal velocity* defined as

$$v_{ij}^{sig} = 2c_{ij} - w_{ij}. \quad (62)$$

One or the other of the two formulations above can be used in EvoL.

In general, artificial dissipation should be applied only where really necessary. However, it has long been recognized that artificial viscosity un-physically boosts post-shock shear viscosity, suppressing structures in the velocity field on scales well above the nominal resolution limit, and damping the generation of turbulence by fluid instabilities.

To avoid the overestimate of the shear viscosity, Balsara (1995) proposed to multiply the chosen expression of Π_{ij} by a function $\bar{f}_{ij} = (f_i + f_j)/2$; the proposed expression for the “limiter” f_i for a particle i is

$$f_i = \frac{|\nabla_i \cdot \mathbf{v}_i|}{|\nabla_i \cdot \mathbf{v}_i| + |\nabla_i \times \mathbf{v}_i| + \eta c_i/h_i}, \quad (63)$$

where $\eta \sim 10^{-4}$ is a factor introduced to avoid numerical divergences. It can be seen that f acts as a limiter reducing the efficiency of viscous forces in presence of rotational flows.

Another possible cure to the problem is as follows. The most commonly used values for the parameters α and β in Eq. 57 are $\alpha = 1$ and $\beta = 2$. Bulk viscosity is primary produced by α , and this sometimes over-smoothen the post-shock turbulence. To cope with this, Morris & Monaghan (1997) have proposed a modification to Eq. 57, in which the parameter α depends on the particle and changes with time according to:

$$\frac{d\alpha_i}{dt} = -\frac{\alpha_i - \alpha_{min}}{\tau} + G_i, \quad (64)$$

where $\alpha_{min} = 0.01$ is the minimum allowed value, $\tau = 0.1h_i/v_i^{sig}$ is an e -folding time scale (with $v_i^{sig} = \max_j[v_{ij}^{sig}]$), and G_i is a source term which can be parameterized as $G_i = 0.75f_i \max[0, -|\nabla \cdot \mathbf{v}_i|]$. This formulation embodies some different prescriptions (e.g. in Rosswog & Price 2007; Price 2008). In Eq. 61, one can then put $\alpha = \frac{1}{2}(\alpha_i + \alpha_j)$. Dolag et al. (2005) have shown that this scheme captures shocks as well as the original formulation of SPH, but, in regions away from shocks, the numerical viscosity is much smaller. In their high resolution cluster simulations, this resulted in higher levels of turbulent gas motions in the ICM, significantly affecting radial gas profiles and bulk cluster properties. Interestingly, this tends to reduce the differences between SPH and adaptive mesh refinement simulations of cluster formation.

3.3.2. Artificial thermal conductivity

As Price (2008) pointed out, an approach similar to that described above should be used to smooth out discontinuities in *all* physical variables. In particular, an *artificial thermal conductivity* is necessary to resolve discontinuities in thermal energy (even if only a few formulations of SPH in literature take this into account).

The artificial thermal conductivity is represented by the term

$$\Pi^u = -\frac{\alpha_{ij}^u v_{sig}^u (u_i - u_j)}{\rho_{ij}}. \quad (65)$$

Here α^u is a conductivity coefficient, $\frac{1}{2}(\alpha_i^u + \alpha_j^u)$ that varies with time according to

$$\frac{d\alpha_i^u}{dt} = -\frac{\alpha_i^u - \alpha_{min}^u}{\tau} + S_i^u, \quad (66)$$

where τ is the same time scale as above, α_{min}^u is usually 0, and the source term can be defined as $S_i^u = h_i |\nabla^2 u_i| / \sqrt{u_i}$ (Price 2008), in which the expression for the second derivative is taken from Brookshaw (1986)

$$\nabla^2 u_i = \sum_j 2m_j \frac{u_i - u_j}{\rho_j} \frac{|\nabla_i \bar{W}_{ij}|}{r_{ij}} \quad (67)$$

which reduces particles noise; v_{ij}^{sig} can be either the same quantity defined in Eq. 62 or

$$v_{ij}^{sig} = \sqrt{\frac{|P_i - P_j|}{\rho_{ij}}} \quad (68)$$

In passing, we note that adopting the source term suggested by Price & Monaghan (2005), $S_i^u = 0.1h_i |\nabla^2 u_i|$, would be less accurate in problems requiring an efficient thermal conduction such as the standard shock tube test with un-smoothed initial conditions, see Sect. 5.

The term Π^u must finally be added to Eq. 57, giving

$$\begin{aligned} \frac{du_i}{dt} = & \sum_j m_j \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\zeta_i/m_j}{\Omega_i^*} \right) (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}(h_i) \right. \\ & \left. + \left(\frac{1}{2} \Pi_{ij} + \Pi_{ij}^u \right) w_{ij} |\nabla_i \bar{W}_{ij}| \right]. \end{aligned} \quad (69)$$

As in the case of artificial viscosity, it is worth noting that this conductive term is not intended to reproduce a physical dissipation; instead, it is a merely numerical artifact introduced to smooth out unresolvable discontinuities.

3.4. Entropy equation

Instead of looking at the thermal energy, we may follow the evolution of a function of the total particle entropy, for instance the *entropic function*

$$A(s) = \frac{P}{\rho^\gamma}, \quad (70)$$

where γ is the adiabatic index. This function is expected to grow monotonically in presence of shocks, to change because of radiative losses or external heating, and to remain constant for an adiabatic evolution of the gas. Springel & Hernquist (2002) suggest the following SPH expression for the equation of entropy conservation:

$$\begin{aligned} \frac{dA_i}{dt} = & -\frac{\gamma-1}{\rho_i^\gamma} S(\rho_i, u_i) \\ & + \frac{1}{2} \frac{\gamma-1}{\rho_i^{\gamma-1}} \sum_j m_j \Pi_{ij} \mathbf{v}_{ij} \cdot \nabla_i W_{ij}, \end{aligned} \quad (71)$$

where S is the *source function* describing the energy losses or gains due to non adiabatic physics apart from viscosity. If $S < 1$, the function $A(s)$ can only grow in time, because of the shock viscosity. Eq. 72 can be integrated instead of Eq. 57 giving a more stable behaviour in some particular cases. Note that internal energy and entropic function of a particle can be related via

$$u = \frac{A(s)}{\gamma-1} \rho^{\gamma-1}. \quad (72)$$

The entropic function can be used to detect situations of shocks, since its value only depends on the strength of the viscous dissipation.

In general, if the energy equation is used (integrated), the entropy is not exactly conserved, and viceversa. Moreover, if the density is evolved according to the continuity Eq. 38 and the thermal energy according to Eq. 54, both energy and entropy are conserved; but in this case the mass is not be conserved (to ensure this latter, density must be computed with Eq. 34). This problem should be cured with the inclusion of the ∇h terms in the SPH method, as already discussed.

3.5. X-SPH

As an optional alternative, EvoL may adopt the smoothing of the velocity field by replacing the normal equation of motion $d\mathbf{r}_i/dt = \mathbf{v}_i$ with

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i + \eta \sum_j m_j \left(\frac{\mathbf{v}_{ji}}{\rho_{ji}} \right) W_{ij}, \quad (73)$$

(Monaghan 1992) with $\rho_{ji} = (\rho_i + \rho_j)/2$, $\mathbf{v}_{ji} = \mathbf{v}_j - \mathbf{v}_i$, and η constant, $0 \leq \eta \leq 1$. In this variant of the SPH method, known as *X-SPH*, a particle moves with a velocity closer to the average velocity in the surroundings. This formulation can be useful in simulations of weakly incompressible fluids, where it keeps particles in order even in absence of viscosity, or to avoid undesired inter-particle penetration.

4. Description of the Code: Integration

4.1. Leapfrog integrator

Particles positions and velocities are advanced in time by means of a standard *leapfrog* integrator, in the so-called

Kick-Drift-Kick (**KDK**) version, where the **K** operator evolves velocities and the **D** operator evolves positions:

$$\begin{aligned} \mathbf{K}_{1/2}: \mathbf{v}_{n+1/2} &= \mathbf{v}_n + \frac{1}{2}\mathbf{a}(\mathbf{r}_n)\delta t \\ \mathbf{D}: \mathbf{r}_{n+1} &= \mathbf{r}_n + \mathbf{v}_{n+1/2}\delta t \\ \mathbf{K}_{1/2}: \mathbf{v}_{n+1} &= \mathbf{v}_{n+1/2} + \frac{1}{2}\mathbf{a}(\mathbf{r}_{n+1})\delta t. \end{aligned}$$

A predicted value of physical quantities at the end of each time-step is also predicted at the beginning the same step, to guarantee synchronization in the calculation of the accelerations and other quantities. In particular, in the computation of the viscous acceleration on gas particles the synchronized predicted velocity $\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}(\mathbf{r}_n)\delta t$ is used.

Moreover, if the individual time-stepping is adopted (see below, Sect. 4.2.1), non-active particles use predicted quantities to give their own contributions to forces and interactions.

4.2. Time-stepping

A trade-off between accuracy and efficiency can be reached by a suitable choice of the time-step. One would like to obtain realistic results in a reasonable amount of CPU time. To do so, numerical stability must be ensured, together with a proper description of all relevant physical phenomena, at the same time reducing the computational cost keeping the time-step as large as possible.

To this aim, at the end of each step, for each particle the optimal time-step is given by the shortest among the following times:

$$\begin{aligned} \delta t_{acc,i} &= \eta_{acc} \sqrt{\frac{\min(\epsilon_i, h_i)}{|\mathbf{a}_i|}} \\ \delta t_{vel,i} &= \eta_{vel} \sqrt{\frac{\min(\epsilon_i, h_i)}{|\mathbf{v}_i|}} \\ \delta t_{av,i} &= \eta_{av} \sqrt{\frac{|\mathbf{v}_i|}{|\mathbf{a}_i|}} \end{aligned}$$

with obvious meaning of symbols; the parameters η 's are of the order of 0.1. The third of these ratios, under particular situations, can assume extremely small values, and should therefore be used with caution.

We can compute two more values, the first obtained from the well known *Courant condition* and the other constructed to avoid large jumps in thermal energy:

$$\begin{aligned} \delta t_{C,i} &= \eta_C \frac{h_i}{h_i |\nabla \cdot \mathbf{v}_i| + c_i + 1.2(\alpha c_i + \beta \max_j |\mu_{ij}|)} \\ \delta t_{u,i} &= \eta_u \left| \frac{u_i}{du_i/dt} \right| \end{aligned}$$

where α, β and μ are the quantities defined in the artificial viscosity parametrization, see Sect. 3.3.1, and c is the sound speed.

If the simulation is run with a *global* time-stepping scheme, all particles are advanced adopting the minimum of all individual time-steps calculated in this way; synchronization is clearly guaranteed by definition.

If desired, a minimum time-step can be imposed. In this case, however, limiters must be adopted to avoid numerical divergences in accelerations and/or internal energies. This of course leads to a poorer description of the physical evolution of the system; anyway, in some cases a threshold may be necessary to avoid very short time-steps due to violent high energy phenomena.

4.2.1. Individual time-stepping

Cosmological systems usually display a wide range of densities and temperatures, so that an accurate description of very different physical states is necessary within the same simulation. Adopting the *global* time-stepping scheme may cause a huge waste of computational time, since gas particles in low-density environments and collisionless particles generally require much longer time-steps than the typical high-density gas particle.

In EvoL an *individual* time-stepping algorithm can be adopted, which makes a better use of the computational resources. It is based on a powers-of-two scheme, as in Hernquist & Katz (1989). All individual particles real time steps, $\delta t_{i,true}$, are chosen to be a power-of-two subdivision of the largest allowed time-step (which is fixed at the beginning of the simulation), i.e.

$$\delta t_{i,true} = \delta t_{max}/2^{n_i}, \quad (74)$$

where n_i is chosen as the minimum integer for which the condition $\delta t_{i,true} \geq \delta t_i$ is satisfied. Particles can move to a smaller time bin (i.e. a longer time step) at the end of their own time step, but are allowed to do so only if that time bin is currently time-synchronized with their own time bin, thus ensuring synchronization at the end of every largest time step.

A particle is then marked as “active” if $t_i + \delta t_{i,true} = t_{sys}$, where the latter is the global system time, updated as $t_{sys,new} = t_{sys,old} + \delta t_{sys}$, where $\delta t_{sys} = \min(\delta t_i)$. At each time-step, only active particles re-compute their density and update their accelerations (and velocities) via summation over the tree and neighboring particles. Non-active particles keep their acceleration and velocity fixed and only update their position using a prediction scheme at the beginning of the step. Note that some other evolutionary features (i.e. cooling or chemical evolution) are instead updated at every time-step for *all* particles.

Since in presence of a sudden release of very hot material (e.g. during a Supernova explosion) one or a few particles may have to drastically and abruptly reduce their time-step, the adoption of individual time-stepping scheme would clearly lead to unphysical results, since

neighbouring non-active particles would not notice any change until they become active again. This is indeed a key issue, very poorly investigated up to now. To cope with this, a special recipe has been added, forcing non-active particles to “wake up” if they are being shocked by a neighbour active particle, or in general if their time-step is too long (say more than 4-8 times bigger) with respect to the minimum time-step among their neighbours. In this way, when an active particle suddenly changes its physical state, all of its neighbouring particles are soon informed of the change and are able to react properly. Saitoh & Makino (2008) recently studied the problem, coming to similar conclusions and recommending a similar prescription when modelling systems in which sudden releases of energy are expected.

Note that when a non-active particle is waken up energy and momentum are not perfectly conserved, since its evolution during the current time-step is not completed. Anyway, this error is negligible if compared to the errors introduced by ignoring this correction.

4.3. Periodic boundary conditions

Periodic boundary conditions are implemented in EvoL using the well known Ewald (1921) method, i.e. adding to the acceleration of each particle an extra-term due to the replicas of particles and nodes, expanding in Fourier series the potential and the density fluctuation, as described in Hernquist et al. (1991) and in Springel et al. (2001). If $\mathbf{x}_i$ is the coordinate at the point of force-evaluation relative to a particle or a node j of mass m_j , the *additional* acceleration that must be added to account for the action of the infinite replicas of j is given by

$$\begin{aligned} acc_{per}(\mathbf{x}) = m_j & \left[\frac{\mathbf{x}}{|\mathbf{x}|^3} - \sum_n \frac{\mathbf{x} - \mathbf{n}L}{|\mathbf{x} - \mathbf{n}L|^3} \times \right. \\ & \left(\operatorname{erfc}(\omega|\mathbf{x} - \mathbf{n}L|) + \frac{2\omega|\mathbf{x} - \mathbf{n}L|}{\sqrt{\pi}} \times e^{-\omega^2|\mathbf{x} - \mathbf{n}L|^2} \right) \\ & \left. - \frac{2}{L^2} \sum_{\mathbf{h} \neq 0} \frac{\mathbf{h}}{|\mathbf{h}|^2} e^{-\frac{\pi^2|\mathbf{h}|^2}{\omega^2 L^2}} \sin\left(\frac{2\pi}{L}\mathbf{h} \cdot \mathbf{x}\right) \right] \end{aligned} \quad (75)$$

where $\mathbf{n}$ and $\mathbf{h}$ are integer triplets, L is the periodic box size, and ω is an arbitrary number. Convergence is achieved for $\omega = 2/L$, $|\mathbf{n}| < 5$ and $|\mathbf{h}| < 5$ (Springel et al. 2001). The correcting terms acc_{per}/m_j are tabulated, and trilinear interpolation off the grid is used to compute the correct accelerations. It must be pointed out that this interpolation significantly slows down the tree algorithm, almost doubling the CPU time.

Periodism in the SPH scheme is obtained straightforwardly finding neighbours across the spatial domain and taking the modulus of their distance, in order to obtain their nearest image with respect to the active particle. In this way, no ghost particles need to be introduced.

4.4. Parallelization

EvoL uses the MPI communication protocol to run in parallel on multiple CPUs. First of all, particles must be subdivided among the CPUs, so that each processor has to deal only with a limited subset of the total number of bodies. To this aim, at each time-step the spatial domain is subdivided using the *Orthogonal Recursive Bisection* (ORB) scheme. In practice, each particle keeps track of the time spent to compute its properties during the previous time-step in which it has been active, storing it in a work-load variable. At the next time-step the spatial domain is subdivided trying to assign to each CPU the same total amount of work-load (only considering active particles if the individual time-stepping option is selected), re-assigning bodies near the borders among the CPUs. To this aim, the domain is first cut along the x -axis, at the coordinate x_{cut} that better satisfies the equal work-load condition among the two groups of CPUs formed by those with geometrical centers $x_{centre} \leq x_{cut}$ and those with $x_{centre} > x_{cut}$. In practice, each CPU exchanges particles only if its borders are across x_{cut} . Then, the two groups of CPUs are themselves subdivided into two new groups, this time cutting along the y -axis (at two different coordinates, because they are independent from one another having already exchanged particles along the x -axis). The process is iterated as long as required (depending to the number of available CPUs) cutting recursively along the three dimensions. It should be noted that this scheme allows to use only a power-of-two number of CPUs, whereas other procedures (e.g. a Peano-Hilbert domain decomposition, see e.g. Springel 2005) can more efficient exploit the available resources.

It may sometimes happen that a CPU wants to acquire more particles than permitted by the dimensions of its arrays. In this case, data structures are re-allocated using temporary arrays to store extra-data, as described in the next Section.

Apart from the allocation of particles to the CPUs, the other task for the parallel architecture is to spread the information about particles belonging to different processors. This is achieved by means of “ghost” structures, in which such properties are stored when necessary. A first harvest among CPUs is made at each time step *before* computing SPH densities: at this point, positions, masses and other useful physical features of nearby nodes and particles, belonging to other processors, are needed. Each CPU “talks” only to another CPU per time, first importing *all* the data-structures belonging to it in temporary arrays, and then saving within its “ghost-tree” structure only the data relative to those nodes and particles which will be actually used. For example, if a “ghost node” is sufficiently far away from the active CPU geometrical position so that the gravitational opening criterion is satisfied, there will be no need to open it to compute gravitational forces, and no other data relative to the particles and sub-nodes it contains will be stored in the ghost-tree.

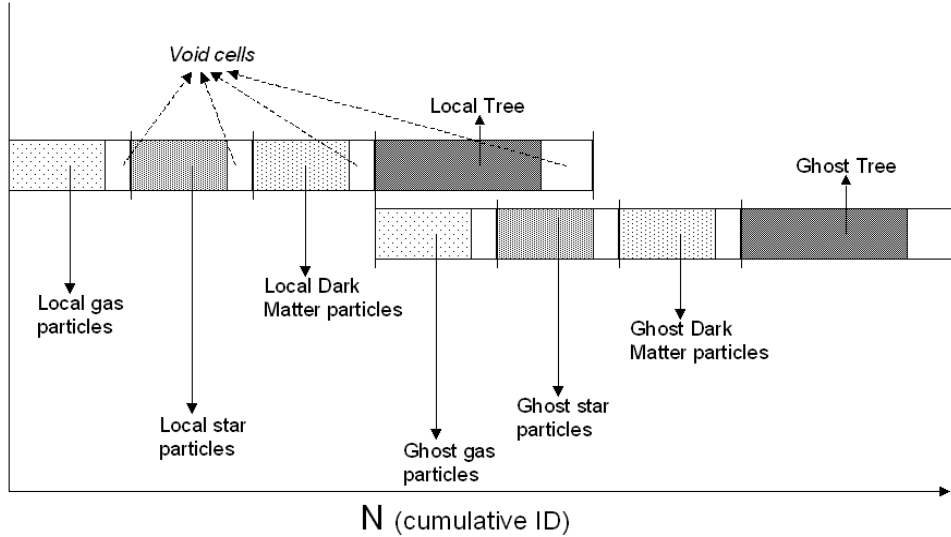


Fig. 2. Data structures in EvoL. Void cells are included to allow for new particles to be created and/or imported from other CPUs. Ghost structures are reallocated at each time-step.

A second communication among CPUs is necessary before computing accelerations. Here, each particle needs to know the exact value of the smoothing and softening lengths (which have been updated during the density evaluation stage) as well as many other physical values of neighbouring (but belonging to different CPUs) particles. The communication scheme is exactly the same as before, involving ghost structures. It was found that, instead of upgrading the ghost-tree built before the density evaluation, a much faster approach is to completely re-build it.

The dimensions of the ghost arrays are estimated at the beginning of the simulation, and every N time-steps (usually, $N = 100$), by means of a “preliminary walk” among all CPUs. In intermediate steps, the dimensions of these arrays are modified on the basis of the previous step evaluation and construction of ghost structures.

4.5. Data structures

All data structures are dynamically adapted to the size of the problem under exploration. In practice, at the beginning of a simulation, all arrays are allocated so that their dimension is equal to the number of particles per processor, plus some empty margin to allow for an increase of the number of particles to be handled by the processor (see Fig. 2).

Anyway, whenever a set of arrays within a processor becomes full of data and has no more room for new particles, the whole data structure of that processor is re-scaled increasing its dimensions, to allow for new particles to be included. Of course, also the opposite case (i.e. if the number of particles decreases, leaving too many void places) is considered.

To this aim, the data in excess are first stored in temporary T arrays. Then, the whole “old” data structure O is copied onto new temporary N arrays, of the new required dimensions. Finally, T arrays are copied into the N structure; this substitutes the O structure, which is then deleted. Clearly, this procedure is quite memory-consuming. Anyway, it ideally has to be carried out only a few times per simulation, at least in standard cases. Moreover, it is performed *after* the “ghost” scheme described above has been completed, and ghost-structures have been deallocated, leaving room for other memory-expensive routines.

5. Tests of the code

An extended set of hydrodynamical tests have been performed to check the performance of EvoL under different demanding conditions:

- Rarefaction and expansion problem (Einfeldt et al. 1991, 1D);
- Shock tube problem (Sod 1978, both in 1D and in 3D);
- Interacting blast waves (Woodward & Colella 1984, 1D);
- Strong shock collapse (Noh 1987, 2D);
- Kelvin-Helmholtz instability problem (2D);
- Gresho vorticity problem (Gresho & Chan 1990, 2D);
- Point-like explosion and blast wave (Sedov-Taylor problem, 3D);
- Adiabatic collapse (Evrard 1988, 3D);
- Isothermal collapse (Boss & Bodenheimer 1979, 3D);
- Collision of two polytropic spheres (3D);
- Evolution of a two-component fluid (3D).

Except where explicitly pointed out, the tests have been run on a single CPU, with the global time-stepping

algorithm, and adopting the following parameters for the SPH scheme: $\eta = 1.2$, $\alpha_{max} = 2$, $\alpha_{min} = 0.01$, $\beta = 2\alpha$, $\gamma = 5/3$, and equation of state $P = (\gamma - 1)\rho u$. Where gravitation is present, the opening angle for the tree code has been set $\theta = 0.8$. The tests in one and two dimensions have been run adopting the kernel formulations shown in Appendix A. The exact analytical solutions have been obtained using the software facilities freely available at the website http://cococubed.asu.edu/code_pages/codes.shtml.

5.1. Rarefaction wave

We begin our analysis with a test designed to check the code in extreme, potentially dangerous low-density regions, where some iterative Riemann solvers can return negative values for pressure/density. In the Einfeldt rarefaction test (Einfeldt et al. 1991) the two halves of the computational domain $0 < x < 1$ move in opposite directions, and thereby create a region of very low density near the initial velocity discontinuity at $x = 0.5$. The initial conditions of this test are

$$\begin{cases} \rho = 1.0, & P = 0.4, & v_x = -2.0, & \text{for } x < 0.5, \\ \rho = 1.0, & P = 0.4, & v_x = 2.0, & \text{for } x \geq 0.5. \end{cases} \quad (76)$$

The results at $t = 0.2$ for a 1-D, 1000-particle calculation are shown in Fig. 3. Clearly, the low-density region is well described by the code.

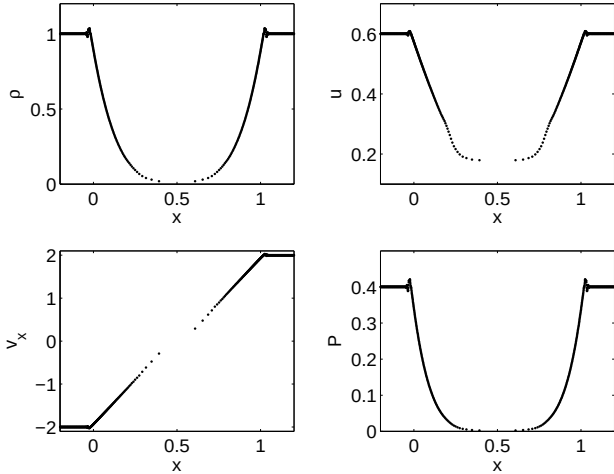


Fig. 3. Left to right, top to bottom: density, internal energy, x-velocity, and pressure profiles at $t=0.2$ for the Einfeldt test. All quantities are in code units.

5.2. Shock Tube

The classic Riemann problem to check the shock capturing capability is the Sod's shock tube test (Sod 1978). We run many different cases changing numerical parameters, to check the response to different settings. All tests have

been started from *un-smoothed* initial conditions, leaving the code to smear out the initial discontinuities.

5.2.1. 1-D tests

The first group of tests have been performed in one dimension. 1,000 equal mass particles have been displaced on the x axis, in the domain $0 \leq x \leq 1$, and changing inter-particle spacing, to obtain the following setting:

$$\begin{cases} \rho = 1.0, & u = 1.5, & v = 0, & \text{for } x < 0.5, \\ \rho = 0.25, & u = 1.077, & v = 0, & \text{for } x \geq 0.5. \end{cases} \quad (77)$$

($m = 6.25 \times 10^{-4}$). In this way, particles belonging to the left region of the tube initially have $P = 1$, whereas particles on the right side of the discontinuity have $P = 0.1795$, as in the classic Monaghan & Gingold (1983) test. We performed four runs:

- T1 - standard viscosity (with $\alpha = 1$ and $\beta = 2$) and no artificial conduction;
- T2 - standard viscosity plus artificial conduction;
- T3 - viscosity with variable α plus artificial conduction;
- T4 - the same as T2, but with the same density discontinuity obtained by using particles with different masses on a regular lattice instead of equal mass particles shrinking their spacing. To do so, particles in the left part of the tube have the mass $m = 10^{-3}$, whereas those in the right part have $m = 2.5 \times 10^{-4}$.

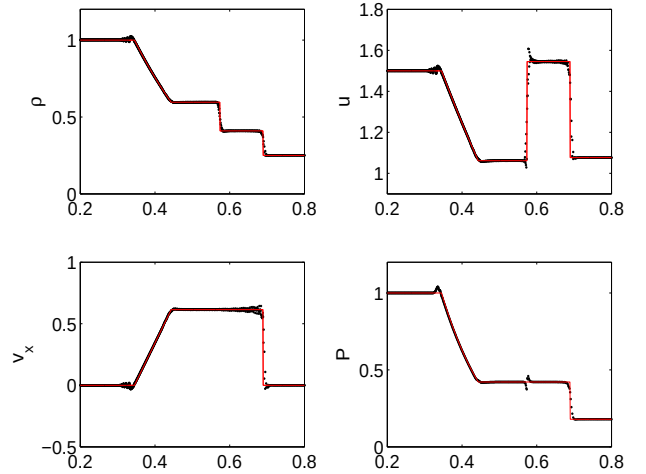


Fig. 4. Left to right, top to bottom: density, internal energy, x-velocity and pressure profiles at $t=0.12$ for test T1 see text for details. Red solid line: exact solution. All quantities are in code units.

Figs. 4 through 7 show the density, internal energy, x -velocity and pressure x -profiles at $t = 0.12$ for the four cases (in code units).

The overall behaviour is good in all cases. We note the typical *blip* in the pressure profile and the wall-like

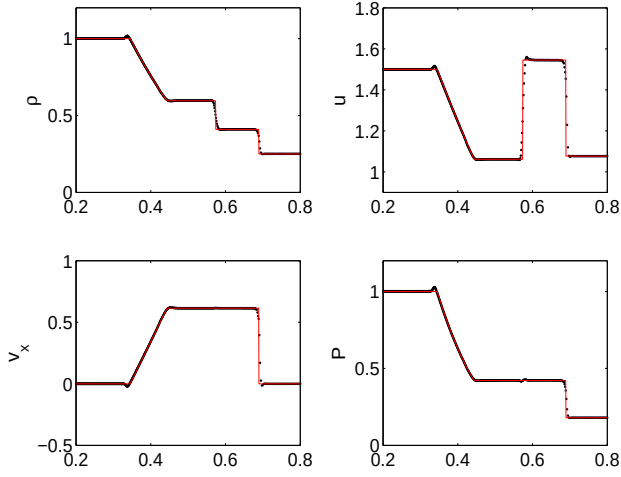


Fig. 5. the same as in Fig. 4, but for the test T2 (see text for details).

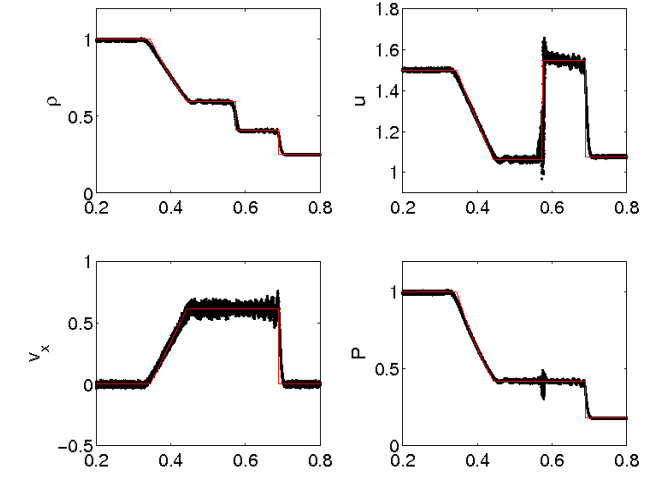


Fig. 8. The same as in Fig. 4, but for the test T5 (see text for details).

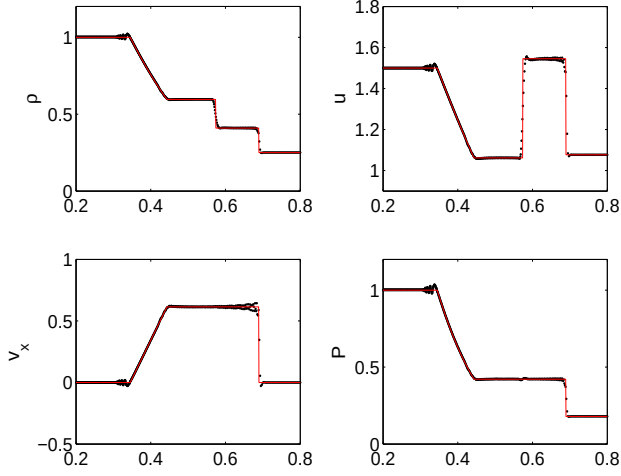


Fig. 6. The same as in Fig. 4, but for the test T3 (see text for details).

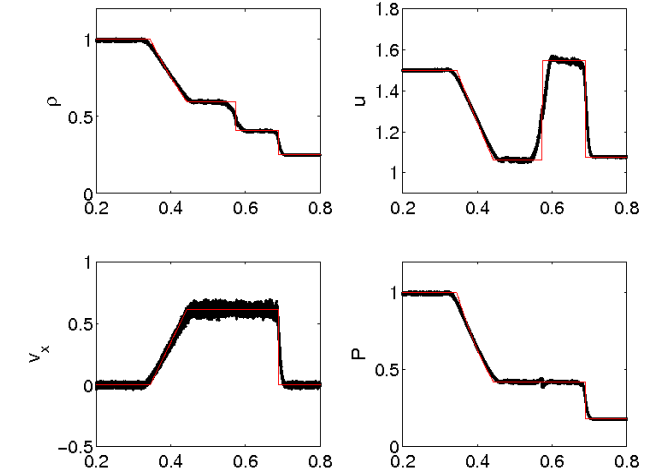


Fig. 9. The same as in Fig. 4, but for the test T6 (see text for details).

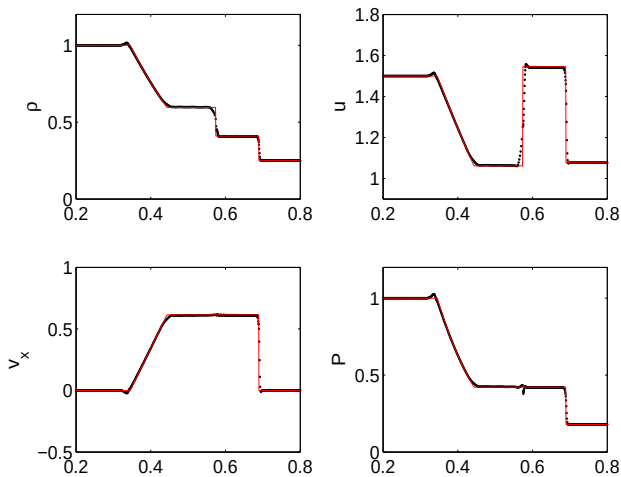


Fig. 7. The same as in Fig. 4, but for the test T4 (see text for details).

overshoot in thermal energy at the contact discontinuity in the T1 run. The introduction of the artificial thermal conductivity (T2) clearly cures the problem at the expense of a slightly shallower thermal energy profile. Introducing the variable α -viscosity (T3), the reduced viscosity causes post-shock oscillations in the density (and energy), and makes the velocity field noisier and turbulent, as expected.

Little differences can be seen in the T4 run, except for a slightly smoother density determination at the central contact discontinuity, which also makes the *blip* problem worse. Thus, the best results are obtained with the T2 configuration.

5.2.2. 3-D tests

As pointed out by Steinmetz & Mueller (1993), performing shock tube calculations with 1-D codes, with particles initially displaced on a regular grid, makes the effects of numerical diffusivity essentially negligible, but in realis-

tic cosmological 3-D simulations the situation is clearly more intriguing. To investigate this issue, we switch to a 3-D description of the shock tube problem. The initial conditions are set using a relaxed *glass* distribution of $\sim 60,000$ particles, in a box of dimensions $[0, 1]$ along the x axis and (periodic) $[0, 0.1]$ along the y and z axis. Particles are assigned a mass $m_{right} = 1.71 \times 10^{-7}$ and $m_{left} = 2.5 \times m_{right}$ on the two sides on the discontinuity at $x = 0.5$, with all other parameters as in the 1-D case. A first run (T5) was performed adopting the T1 configuration (standard viscosity, no artificial conduction), while a second one (T6) was performed switching on the artificial conduction term.

Figs. 8 and 9 show the state of the system at $t = 0.12$ in the two cases (note that *all particles* and not mean values are plotted). As in the 1-D case, the overall agreement with the theoretical expectation is quite good. However, the inclusion of the conductive term on one hand yields sharper profiles and reduces the pressure *blip*, on the other hand gives a smoother description of contact discontinuities (see the density profile at $x = 0.55$). We argue that part of the worse results may be due to the use of particles with different mass as a similar result has been found with the corresponding 1-D test (see T4 above). A limit on the conduction efficiency could give better results, but an *ad hoc* adjustment should be applied to each case.

5.3. Interacting blast waves

Woodward & Colella (1984) proposed a severe test in 1-D, i.e. two strong blast waves develop out of a gas of density $\rho = 1$ initially at rest in the domain $0 < x < 1$, with pressure $P = 1000$ for $x < 0.1$, $P = 100$ for $x > 0.9$ and $P = 0.01$ in between (the adiabatic index is $\gamma = 1.4$). The boundary conditions are reflective on both sides; this can be achieved introducing ghost particles created on-the-fly. Evolving in time, the two blast waves give rise to multiple interactions of strong shocks, rarefaction and reflections.

We performed two runs: one at high resolution with 1000 equally spaced particles, and one at low resolution with only 200 particles. In both cases, the code included the artificial conduction, and viscosity with constant $\alpha = 1$.

Fig. 10 shows the density x -profiles for the high resolution case, at $t = 0.010, 0.016, 0.026, 0.028, 0.030, 0.032, 0.034, 0.038$. The results can be compared with those of Woodward & Colella (1984), Steinmetz & Mueller (1993) and Springel (2009). The complex evolution of the shock and rarefaction waves is reproduced with good accuracy and with sharp profiles, in nice agreement with the theoretical expectations. The very narrow density peak at $t = 0.028$, which should have $\rho \approx 30$, is well reproduced and only slightly underestimated.

Fig. 11 shows the comparison between the density profiles of the high and low resolution cases, at $t = 0.038$. The low-resolution one shows a substantial smoothing of the contact discontinuities; nevertheless, the overall behaviour

is still reproduced, shock fronts are located at the correct positions, and the gross features of the density profile are roughly reproduced.

5.4. Kelvin-Helmoltz instability

The Kelvin-Helmoltz (KH) instability problem is a very demanding test. Recently, Price (2008) replied to the claim by Agertz et al. (2007) that the SPH algorithm is not able to correctly model contact discontinuities like the Kelvin-Helmoltz (KH) problem. He showed how the inclusion of the conductive term is sufficient to correctly reproduce the instability. We try to reproduce his results running a similar test, using the 2-D version of EvoL.

We set up a regular lattice of particles in the periodic domain $0 < x < 1, 0 < y < 1$, using 256^2 particles. The initial conditions are similar to those adopted by Price (2008). The masses of particles are assigned in such a way that $\rho = 2$ for $|y - 0.5| < 0.25$, and $\rho = 1$ elsewhere (note that in Price 2008 this density difference was instead obtained changing the disposition of equal mass particles). The regions are brought to pressure equilibrium with $P = 2.5$. In this case, although the initial gradients in density and thermal energy are not smoothed, the thermal energy is assigned to the particles *after* the first density calculation, to ensure a continuous initial pressure field.

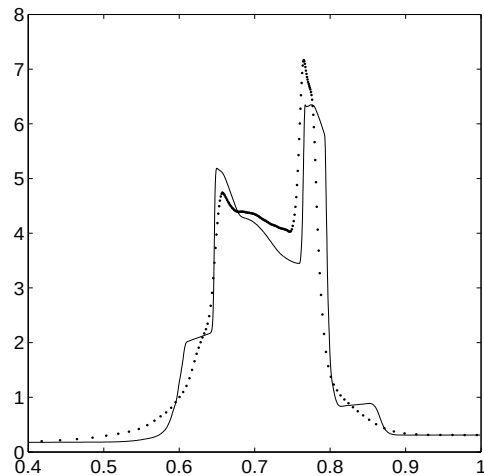


Fig. 11. Comparison between high-resolution (solid line) and low-resolution (dots) density profiles at $t = 0.038$ for the Woodward test (see text for details). The quantities are in code units.

The initial velocity of the fluid is set as follows:

$$v_x = \begin{cases} 0.5 & \text{if } |y - 0.5| < 0.25, \\ -0.5 & \text{if } |y - 0.5| \geq 0.25, \end{cases} \quad (78)$$

and

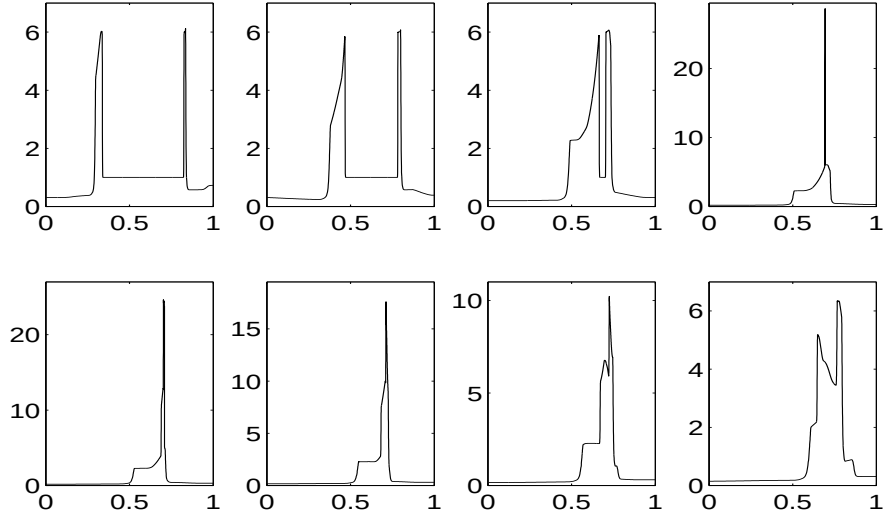


Fig. 10. Left to right, top to bottom: density profiles at $t = 0.010$, $t = 0.016$, $t = 0.026$, $t = 0.028$, $t = 0.030$, $t = 0.032$, $t = 0.034$, $t = 0.038$ for the Woodward high-resolution test (see text for details). All quantities are in code units.

$$v_y = w_0 \sin(4\pi x) \times \left\{ \exp \left[-\frac{(y - 0.25)^2}{2\sigma^2} \right] + \exp \left[-\frac{(y - 0.75)^2}{2\sigma^2} \right] \right\}, \quad (79)$$

where we use $\sigma = 0.05/\sqrt{2}$, and $w_0 = 0.1$ or $w_0 = 0.25$ for two different runs (K1 and K2, respectively). In this way, a small perturbation in the velocity field is forced near the contact discontinuity, triggering the instability.

Figs. 12 and 13 show the temporal evolution of the instability in the two cases. Clearly, our code is able to reproduce the expected trend of whirlpools and mixing between the fluids; the evolution is faster for a stronger initial perturbation. For comparison, we also run a test with the K2 initial conditions, but switching off artificial conduction; Fig. 14 shows the comparison between the final density fields (at $t = 1.79$), and between the initial pressure fields. In the latter, the *blip* that inhibits mixing is clearly visible at the discontinuity layer: the resulting spurious surface tension keeps the two fluids separated, a problem efficiently cured by the artificial conduction term.

5.5. Strong shock collapse

The strong shock collapse (Noh 1987) is another very demanding test, and is generally considered a serious challenge for AMR numerical codes. We run the test in 2-D, displacing $\sim 125,000$ particles on a regular grid and retiling those with $r \leq 2$ to model a gas disk, with initially uniform density $\rho = 1$, and negligible internal energy; we then add a uniform radial velocity towards the origin, $v_{in} = -1$. A spherical shock wave with formally infinite Mach number develops from the centre and travels outwards, leaving a constant density region inside the shock (in 2-D, $\rho_{int} = 16$). The test was run on 4 parallel CPUs.

In Fig. 15, the density in the $x > 0$, $y > 0$ region is shown at $t = 1.2$, while in Fig. 16 the density-radius relation at the same time is plotted for all particles. There is good agreement with the theoretical expectation, although a strong scatter is present in particles density within the shocked region, where moreover the density is slightly underestimated (anyway, this is a common feature in this kind of test: see e.g. Springel 2009).

Apart from these minor flaws, the code is able to handle the very strong shock without any particular difficulty.

5.6. Gresho vortex problem

The test for the conservation of angular momentum and vorticity, the so-called Gresho triangular vortex problem (Gresho & Chan 1990), in its 2-D version (Liska & Wendroff 2003), follows the evolution of a vortex initially described by an azimuthal velocity profile

$$v_\phi(r) = \begin{cases} 5r & \text{if } 0 \leq r < 0.2, \\ 2 - 5r & \text{if } 0.2 \leq r < 0.4, \\ 0 & \text{if } r \geq 0.4, \end{cases} \quad (80)$$

in a gas of constant density $\rho = 1$. The centrifugal force is balanced by the pressure gradient given by an appropriate pressure profile,

$$P(r) = \begin{cases} 5 + 12.5r^2 & \text{if } 0 \leq r < 0.2, \\ 9 + 12.5r^2 - 20r + 4 \ln(5r) & \text{if } 0.2 \leq r < 0.4, \\ 3 + 4 \ln(2) & \text{if } r \geq 0.4, \end{cases} \quad (81)$$

so that the vortex should remain independent of time.

As noted by Springel (2009), this test seems to be a serious obstacle for SPH codes. Due to shear forces, the

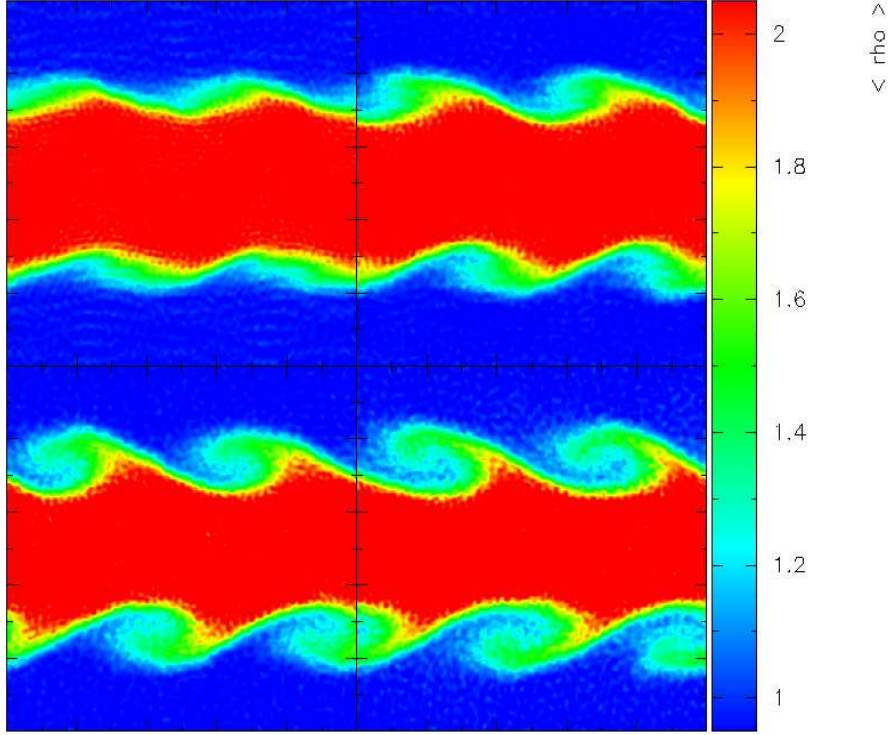


Fig. 12. Kelvin-Helmoltz instability according to test K1. Left to right, top to bottom: density field at $t = 0.768$, $t = 1.158$, $t = 1.549$ and $t = 1.940$. All quantities are in code units.

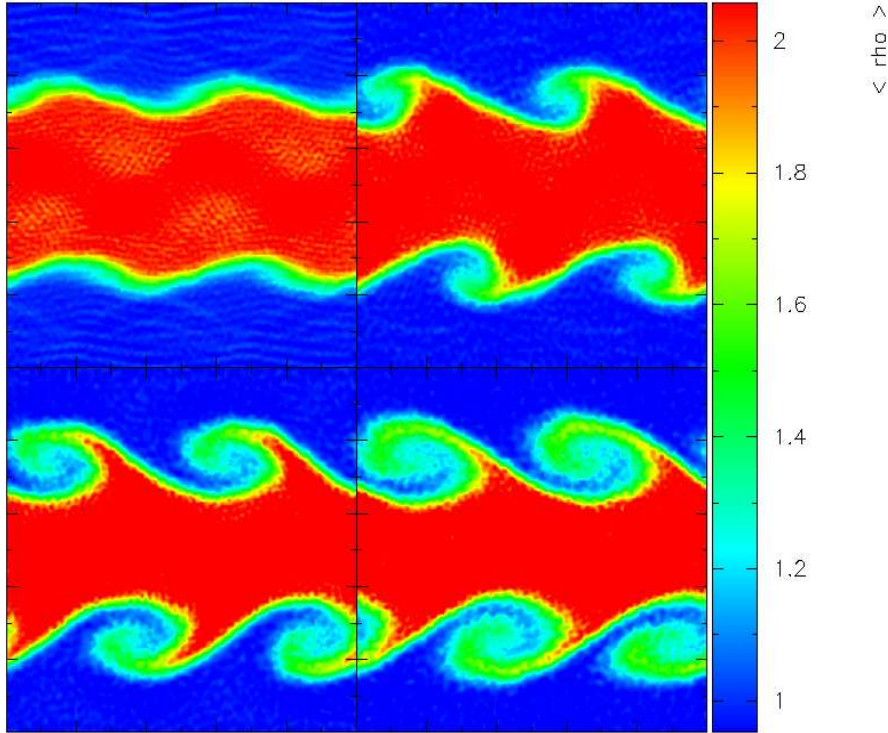


Fig. 13. Kelvin-Helmoltz instability according to test K2. Left to right, top to bottom: density field at $t = 0.377$, $t = 0.767$, $t = 1.158$ and $t = 1.549$. All quantities are in code units.

angular momentum tends to be transferred from the inner to the outer regions of the vortex, eventually causing

the rotational motion to spread within the $r > 0.4$ regions; these finally dissipate their energy because of the

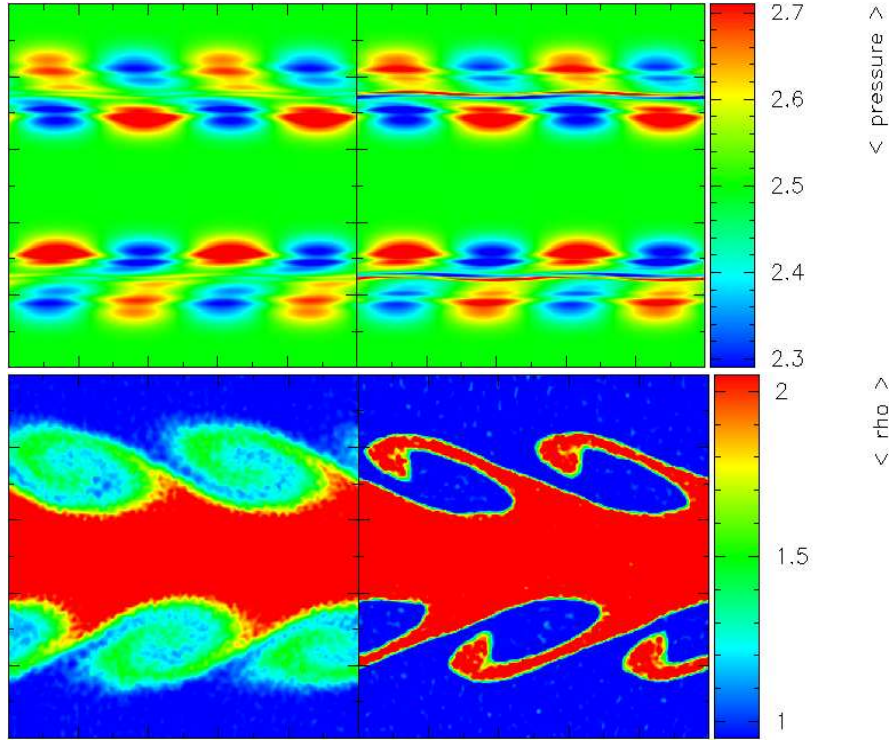


Fig. 14. Kelvin-Helmoltz instability according to test K2, with (left) and without (right) artificial thermal conductivity. Top panels: initial pressure field. Bottom panels: density fields at $t = 1.79$. All quantities are in code units.

interactions with the counter-rotating regions of the periodic replicas. With the GADGET3 code, the vortex did not survive up to $t = 3$. AMR codes generally give a better description, although under some conditions (for example, the addition of a global bulk velocity, which should leave the system unchanged due to its galileian invariance) they also show some flaws.

We tried to run the test starting from two different initial settings of particles, since as pointed out by Monaghan (2006) the initial particle setting may have some influence on the subsequent evolution of the system. Thus, in the first case we set up a regular cartesian lattice of $\sim 10,000$ particles (in the periodic domain $-0.5 < x < 0.5, -0.5 < y < 0.5$), and the second one put the same number of particles on concentric rings, allowing for a more ordered description of the rotational features of the gas (see Fig. 17).

The results are shown in Fig. 18. We find a residual vorticity still present at $t \sim 3$, although strongly degraded and reduced in magnitude. As expected, a substantial amount of the angular momentum has been passed to the outer regions of the volume under consideration. This is confirmed by comparing the initial to the final pressure fields (Fig. 19): at $t = 2.7$ the pressure exerted by the outskirts is higher, due to the increased temperature of the regions confining with the periodic replicas. This leads to a compression of the internal region, which in turn must increase its pressure as well.

However, we could not find in literature other direct comparisons for this test with an SPH code. It must be

pointed out that the resolution in these tests is quite low, and a larger number of particles may help improving the results.

5.7. Point explosion

The presence of the artificial conduction becomes crucial in the point explosion experiment, a classic test to check the ability of a code to follow the evolution of strong shocks in three dimensions. In this test, a spherically symmetric Sedov-Taylor blast-wave develops and must be modelled consistently with the known analytical solution.

We run the test in 3-D. Our initial setting of the SPH particles is a uniform grid, in a box of size $[-0.5, 0.5]$ along each dimension, with uniform density $\rho = 1$ and negligible initial internal energy. The grid is made of 31^3 particles. At $t = 0$ an additional amount $E = 1$ of thermal energy is given to the central particle; the system is subsequently let free to evolve self-consistently (without considering self-gravity). It must be noted that, again, the central additional thermal energy is *not* smoothed, so to obtain extreme conditions.

The following test runs are made (using the Lagrangian SPH scheme):

- S1 - constant viscosity with $\alpha = 1$ viscosity, no artificial conduction;
- S2 - variable viscosity with α plus artificial conduction;
- S3 - as S2, but without the ∇h correcting terms;
- S4 - as S2, with the individual time-stepping scheme;

– S5 - as S2, with higher resolution (51^3 particles).

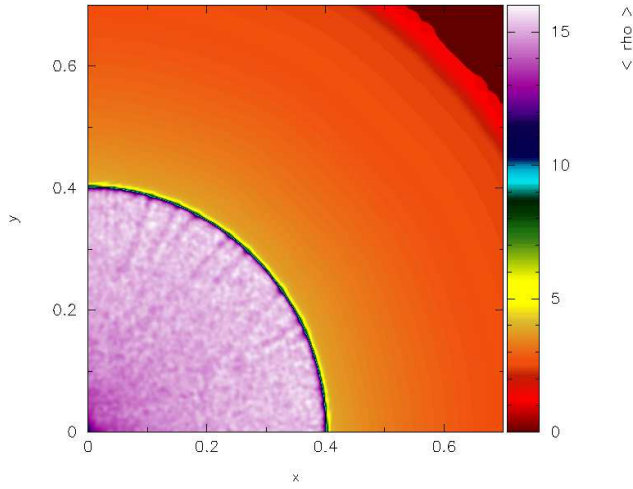


Fig. 15. The Noh’s problem: density field in the first quadrant at $t = 1.2$. All quantities are in code units. See text for details.

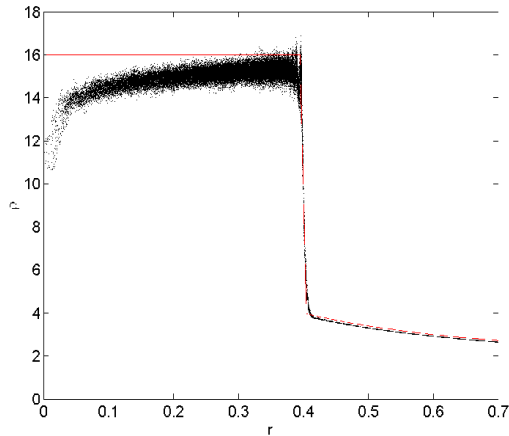


Fig. 16. The Noh’s problem: density vs. radius relation for all particles at $t = 1.2$. Solid line: theoretical expectation. All quantities are in code units. See text for details.

Fig. 20 shows the xy positions of particles belonging to the $z = 0$ plane at $t = 0.09$ for the S1 test, and the corresponding radial density profile, compared to the analytical expectation. A strong noise (which develops soon after the initial explosion) is evident in the particle positions, due to the huge unresolved discontinuity in thermal energy. The description of the shock evolution is quite poor, although its gross features are still captured.

Fig. 21 presents the same data for the S2 run, in which both the viscosity with variable α and artificial conduction are switched on. The increased precision in the description of the problem is immediately evident: the conductive term acts smoothing out the thermal energy budget of the central particle, strongly reducing the noise and allowing

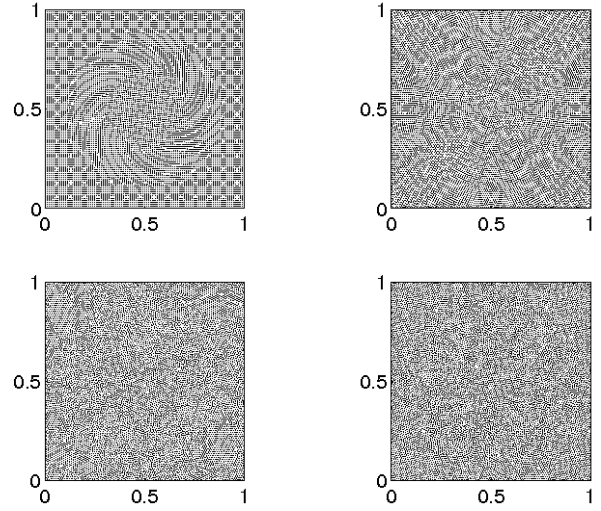


Fig. 17. Positions of all particles at $t = 0.02$ (top) and $t = 2.7$ (bottom) for the two different initial configurations for the Gresho test: grid (left) and rings (right). All quantities are in code units. See text for details.

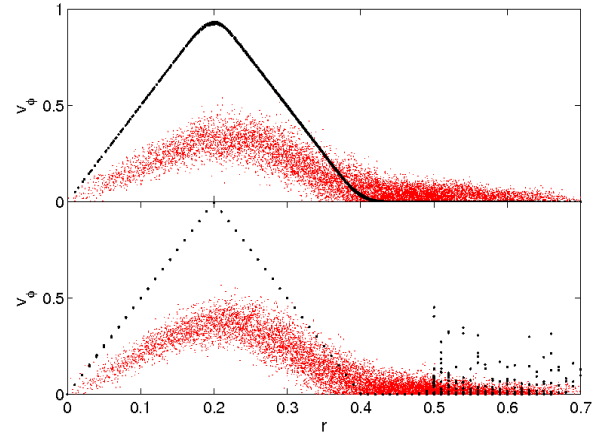


Fig. 18. Initial (black dots) and final (red small points) azimuthal velocity profiles for the two Gresho tests: grid (top) and rings (bottom). All quantities are in code units. See the text for details.

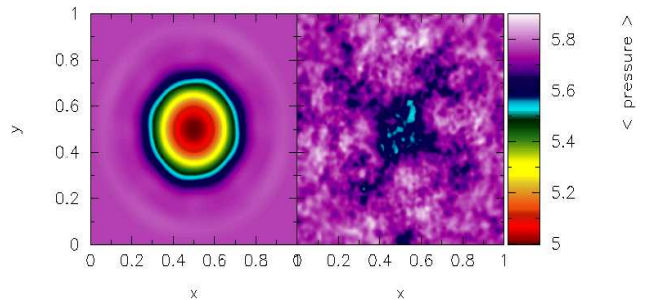


Fig. 19. Initial ($t = 0.02$, left) and final ($t = 2.7$, right) pressure fields in the grid Gresho test. Note the increased pressure in the outskirts (white) and in the central regions of the vortex. All quantities are in code units. See the text for details.

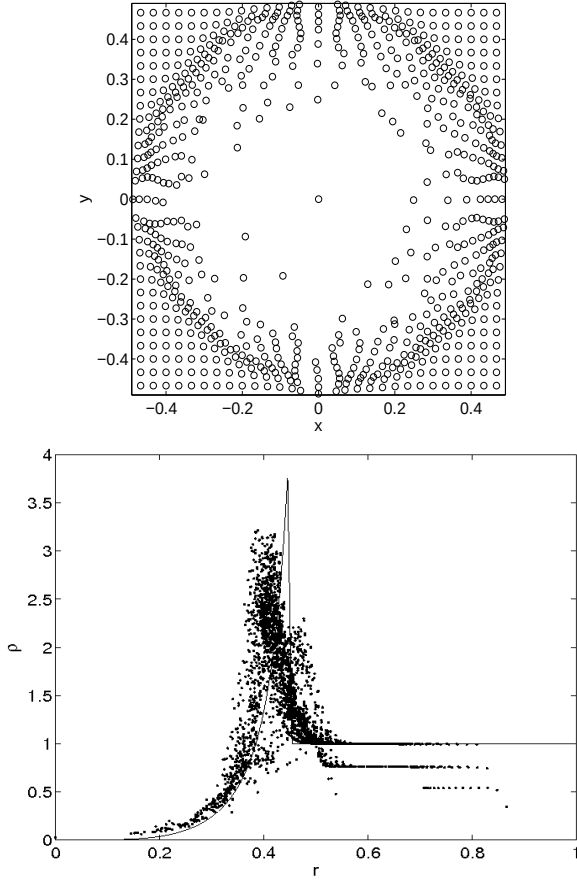


Fig. 20. Top: spatial distribution of particles in a slice taken at $z=0$ and $t=0.09$ for the Sedov blast wave of test S1. Bottom: Radial density profile for the same test. Points: SPH particles; solid line: analytical prediction. All quantities are in code units.

a symmetric and ordered dynamical evolution. The radial density profile (in which single particles, and not averages, are plotted) shows that the shock is well described. The density peak is a factor of ~ 2 lower than the analytical expectation, due to the intrinsic smoothing nature of the SPH technique.

Another test (S3) is carried dropping the ∇h correcting terms (see Sect. 3). While the differences in particle positions and density profiles are almost negligible, the comparison of the total energy conservations $E(t)/E(t_0)$ in the two cases (Fig. 22) shows that the correction helps in containing losses due to wrong determinations of the gradient.

A fourth test (S4) is also run to check the behaviour of the individual time-stepping algorithm; the results are virtually indiscernible from the those of the previous case (not shown here). This is noteworthy as all particles must be “waken up” (using the method described in Sect. 4.2.1), being still and cold at the beginning of the simulation.

Finally, a test (S5) with increased numerical resolution is performed (Fig. 23). As expected, in this case, while the particles positions again accurately describe the evolution

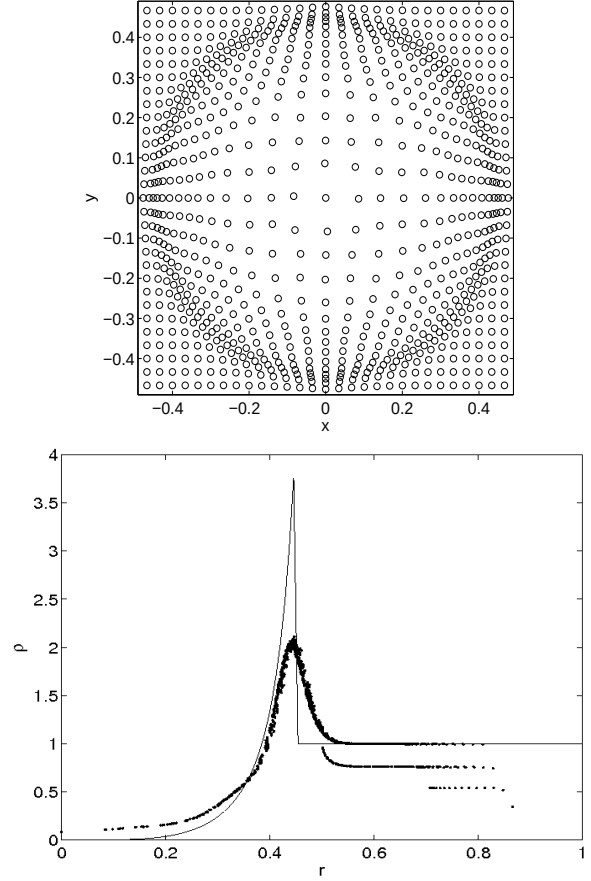


Fig. 21. The same as in Fig. 20, but for the S2 test.

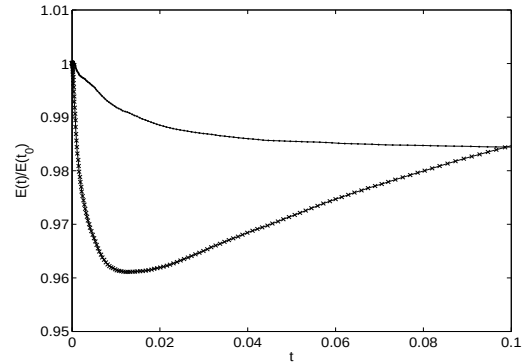


Fig. 22. Energy conservations for Sedov test with artificial conductivity and variable viscosity. Points: with ∇h terms; crosses: without ∇h terms. All quantities are in code units.

of the system, the shock boundary is sharper, and closer to the peak value of the analytical solution.

5.8. Self-gravitating collapse

The collapse of a gaseous self-gravitating sphere is another standard SPH 3-D test (Evrard 1988). The combined action of hydrodynamics and self-gravity leads the system, a sphere of gas initially far from equilibrium, with negli-

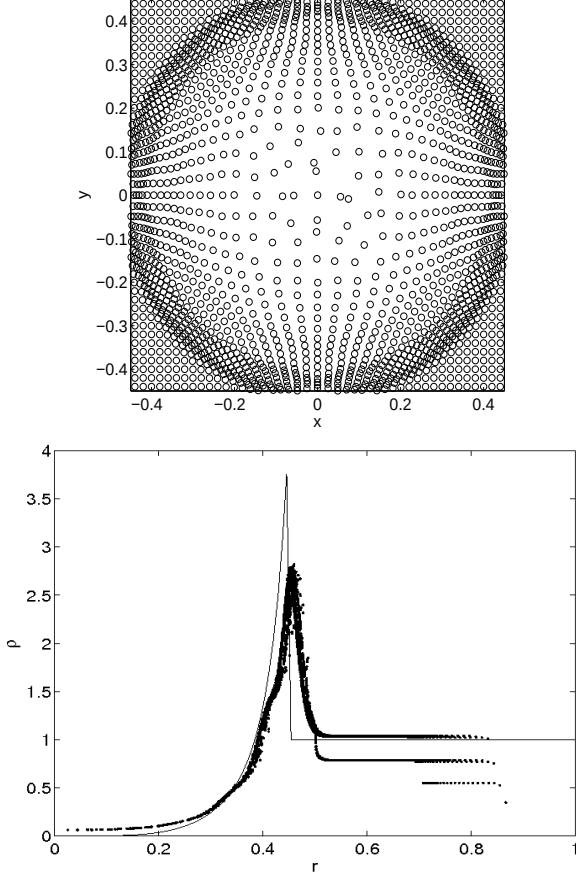


Fig. 23. The same as in Fig. 20, but for the test S5.

gible internal energy and density profile $\rho \propto r^{-1}$, to a collapse in which most of its kinetic energy is converted into heat. An outward moving shock develops, a slow expansion follows and, at late times, a core-halo structure develops with nearly isothermal inner regions and the outer regions cooling adiabatically. To set the initial conditions, 10,000 particles of mass $1.25 \times 10^{-2} M_{\odot}$ are placed in a sphere of radius 5×10^{-6} Mpc, in such a way that their initial density radial profile scales as $1/r$. Another sphere with 40,000 particles and consequently smaller particles' masses (higher resolution) is also prepared. Two cases are examined: namely the collisionless collapse and the adiabatic collapse.

5.8.1. Collisionless collapse

To check the performance of the adaptive softening length algorithm, tests are run switching off hydrodynamical interactions to mimic the collisionless collapse. A first case drops the hydrodynamical interactions, includes the adaptive softening algorithm but neglects the correcting terms presented in Section 2.1. It is meant to provide the reference case. The second one includes also the effects of these correcting terms.

Fig. 24 shows, in the upper panels, the spatial distribution of particles projected onto the xy plane after during

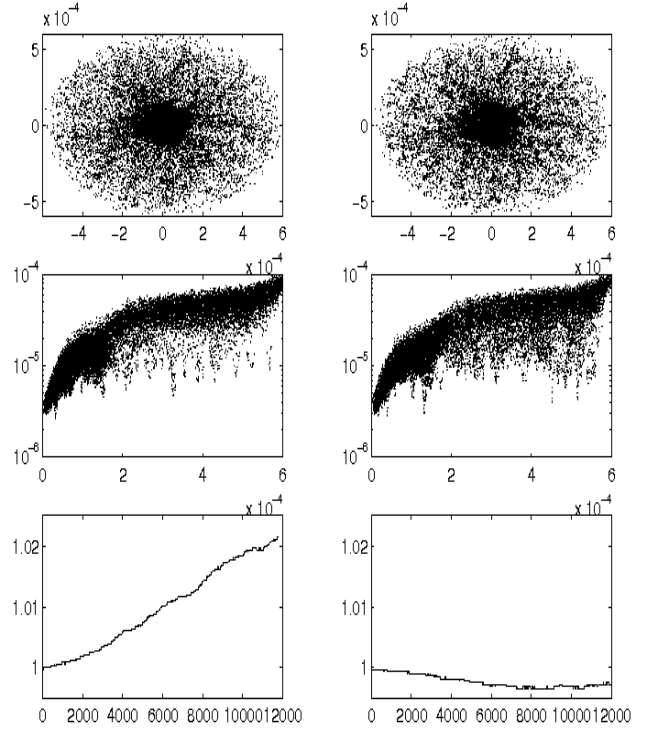


Fig. 24. Collisionless collapse with adaptive softening lengths. Top: spatial distribution of particles projected onto the xy plane after 12,000 steps; middle: softening lengths (code units) as a function of the radial coordinate; bottom: temporal conservation of the total energies of the systems $E(t)/E(t_0)$, for the un-corrected (left) and corrected (right) tests.

the collapse of the sphere, for the un-corrected (left) and corrected (right) cases: no evident difference is present. The same holds for (middle panels) the value of the softening lengths of particles as a function of the radial coordinate. However, the temporal conservation of the total energies of the systems $E(t)/E(t_0)$ in the two cases are quite different (bottom panels of Fig. 24). If the correcting terms are not included a secular error in the energy conservation develops, reaching 2%, while the error is ~ 10 times smaller (albeit not completely eliminated) with the introduction of the extra-terms.

5.8.2. Adiabatic collapse

The standard hydrodynamical tests (Evrard 1988) are calculated with the following settings:

- E1 - constant softening lengths ($\epsilon_i = 0.1R \times (m_i/M)^{0.2}$, with R and M total radius and mass of the sphere at the beginning of the simulation); Lagrangian SPH scheme, with constant viscosity ($\alpha = 1$) and no artificial conduction;
- E2 - as E1, except for using adaptive softening lengths;
- E3 - adaptive softening lengths, viscosity with variable α , and artificial conduction;

– E4 – as E3, higher resolution (40,000 particles).

The global energy trends are shown in Fig. 25, while Fig. 26 shows a magnification of the energy conservation $E(t)/E(t_0)$ for the E1 and E2 runs. The energy conservation is always good, reaching a maximum error of $\sim 0.4\%$ at the moment of maximum compression. For comparison, in the same test the finite differences method by Evrard (1988) ensures conservation at $\simeq 1\%$, while Harfst et al. (2006) finds a maximum error of $\sim 3\%$ with a standard SPH implementation. Almost no difference can be noted between the variable and the constant ϵ runs, despite a variation of more than two orders of magnitude in the softening length values in the E2 run (see Fig. 27).

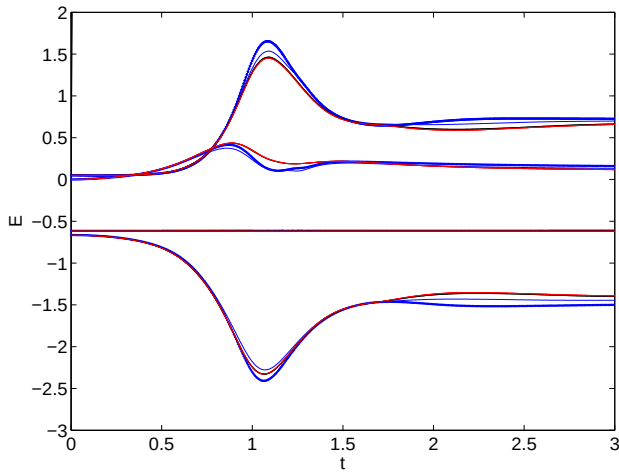


Fig. 25. Energy trends in the cold collapse test. Light lines: red: test E1; black: test E2; blue: test E3. Heavy dotted blue line: test E4 (see text for details). Top to bottom: thermal, kinetic, total and potential energies. All quantities are in code units.

The density and internal energy profiles at $t = 0.8$, for the E1 to E3 runs, are shown in Fig. 28; the comparison between the E3 and the hi-res E4 runs is shown in Fig. 29 (the shock is moving outward leaving an inner, heated core). The discontinuity in density is smoothed out by the kernel smoothing, which in the current implementation requires a high number of neighbors (~ 60), but the effects of the inclusion of the artificial conductive term are evident. We didn't find any particular problem with the inclusion of the conductive term in a self-gravitating system.

We also run a X-SPH test, adopting $\eta = 0.25$. While the energy trends are essentially identical to the ones the previous tests (not shown), the energy conservation is slightly worse, reaching an error of $\sim 2.5\%$. Indeed, Monaghan (2002) pointed out that a more complex treatment of the equations of motion should be adopted to achieve perfect energy conservation.

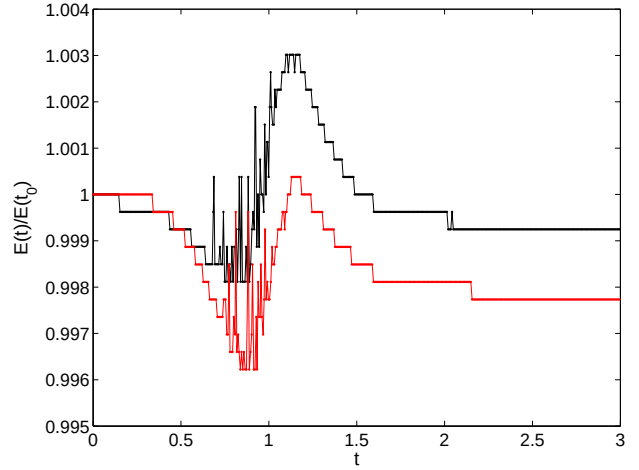


Fig. 26. Magnification of the energy conservations $E(t)/E(t_0)$ in the cold collapse tests. Black: testE1; red: testE2. See text for details.

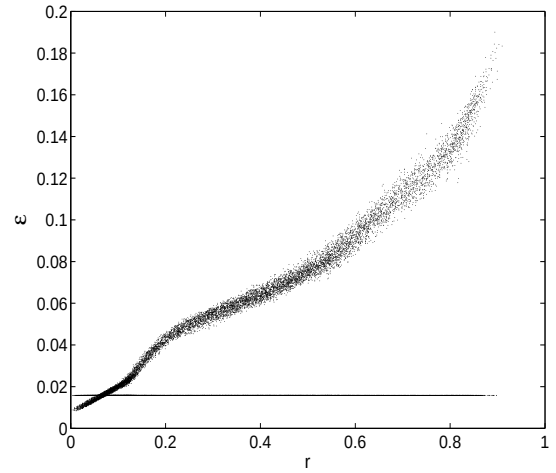


Fig. 27. Softening lengths in the cold collapse test T2 at $t = 0.8$. The solid line is the value of the constant softening length in test T1. All quantities are in code units.

5.9. Isothermal collapse

Introduced by Boss & Bodenheimer (1979) and later revisited by Burkert & Bodenheimer (1993), the test on the isothermal collapse couples hydrodynamics and gravity, also adding rotation. It develops in a complex game of collapse and fragmentation. We run the test in the version proposed by Bate & Burkert (1997).

A homogeneous sphere of cold gas with initial radius $R = 5 \times 10^{16}$ cm and mass $M = 1M_{\odot}$ is modelled retaining $\sim 78,000$ particles out of a uniform lattice. The sphere is put in solid body rotation around the z axis, with angular velocity $\Omega = 7.2 \times 10^{-13}$ rad s $^{-1}$, and the otherwise flat density field is perturbed by 10% in such a way that

$$\rho(\phi) = \rho_0[1 + 0.1\cos(2\phi)],$$

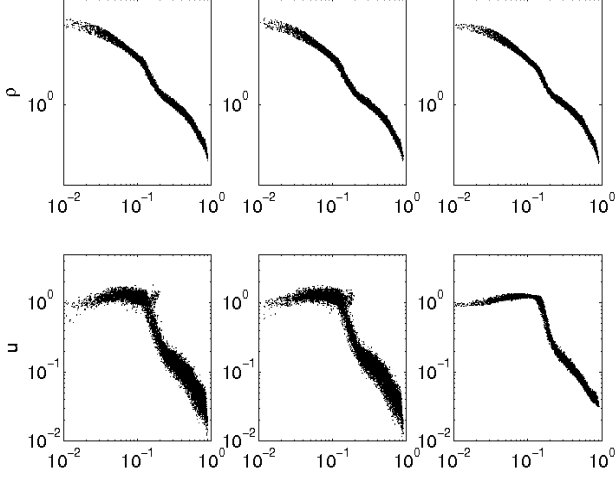


Fig. 28. Density (top) and internal energy (bottom) profiles at $t = 0.8$ in the cold collapse E1 (left), E2 (center) and E3 (right) tests. All particles are plotted.

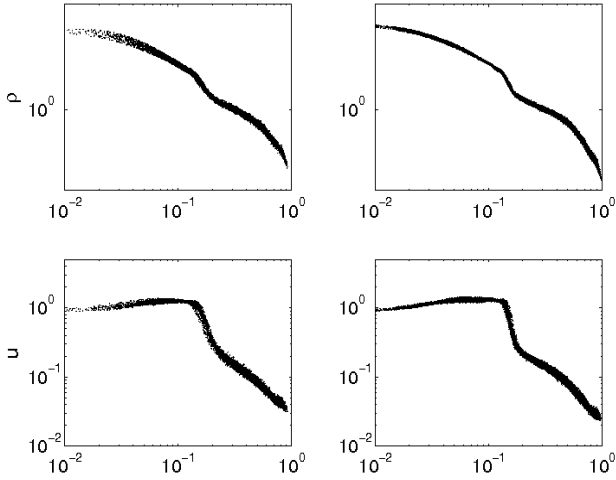


Fig. 29. Comparison between cold collapse E3 (low-res, left) and E4 (hi-res, right) test: density (up) and internal energy (bottom) profiles at $t = 0.8$.

where ϕ is the azimuthal angle about the rotation axis and $\rho_0 = 3.82 \times 10^{-18} \text{ g cm}^{-3}$ (we achieve the density perturbation by varying the masses of particles instead of their positions). The gas is described by an isothermal equation of state, $P = c_s^2 \rho$, with initial sound speed $c_s = 1.66 \times 10^4 \text{ cm s}^{-1}$. It is assumed to be optically thin so that no mechanical heating can occur, and the adiabatic index is $\gamma = 1.4$.

The test is very challenging and has been used to check the behaviour of both AMR and SPH codes. In particular, Bate & Burkert (1997) used it to study the effects of wrong calibrations of the smoothing and softening lengths in SPH (see Sect. 3.2). We run the test in four different cases:

- B1 - Lagrangian SPH with constant softening lengths ($\epsilon \simeq 5.26 \times 10^{14} \text{ cm}$, an intentionally large value);

- B2 - Lagrangian SPH with adaptive softening lengths (no limits imposed on ϵ and h);
- B3 - as B2, but imposing a minimum smoothing *and* softening length $h_{\min} = \epsilon_{\min} = 10^{14} \text{ cm}$, to avoid artificial clumping (as in Bate & Burkert 1997);
- B4 - as B2, with the X-SPH method (adopting $\eta = 0.25$).

Figs. 30 to 33 plot the temporal evolution of the density field in the $z = 0$ plane; shown are snapshots at $t = 1.0, 1.15, 1.23$, and 1.26 , in units of free fall time, $t_{ff} = 1.0774 \times 10^{12} \text{ s}$. Fig. 34 plots the final density field at $t = 1.29$ for the four tests. These plots can be compared to those by Bate & Burkert (1997) and Springel (2005).

The large value of ϵ assumed in the B1 test clearly demonstrates how a wrong choice of the softening length can result in catastrophic errors. The evolution of the system is clearly different from the grid solution used by Bate & Burkert (1997) as a proxy of the exact solution of the problem. Because of the excessive softening, clumps may be stable against collapse and a rotating structure forms, resembling a barred spiral galaxy. More realistic choices for ϵ may give better results, but the opposite problem (artificial clumping of unresolved structures) may arise in case of too small a softening length. Indeed, the case B2, in which the adaptive algorithm is adopted (note that $\epsilon = h$), gives much better results, but disorderly clumped sub-structures may form.

Imposing a minimum ϵ ($=h$, case B3) cures the problem (note however that the evolution of the system seems to be somewhat faster than the grid solution). Anyway, this prescription cannot be generalised. A more physically sounded solution may be the introduction of a *pressure floor* in unresolved regions, i.e. where the Jeans mass of the gas is not resolved by the SPH technique. This issue will be discussed in the forthcoming companion paper (Merlin et al. 2009, in preparation).

Finally, the X-SPH test B4 gives results that are essentially indiscernible from those of case B2, apart from a slightly less disordered density field in the regions surrounding the central collapsed structure.

Fig. 35 plots the temporal evolution of the maximum value of the density. This plot can be compared with Fig. 5a in Bate & Burkert (1997) and Fig. 12 in Springel (2005). The results for all tests are very similar and agree quite well with their SPH tests of comparable resolution (with the plateau at $\rho \simeq 2 \times 10^{-15} \text{ g cm}^{-3}$ reached at $t = 1.14$), except for the case B1 which clearly evolves faster.

For reference, Fig. 36 shows the initial and final values of both ϵ and h (which are equal in the cases B2, B3 and B4) as a function of ρ in the four runs.

5.9.1. Parallel run

Finally, we re-computed the case B2 test using 16 parallel CPUs. Fig. 37 shows the comparison between the final density field for this run and for the original one

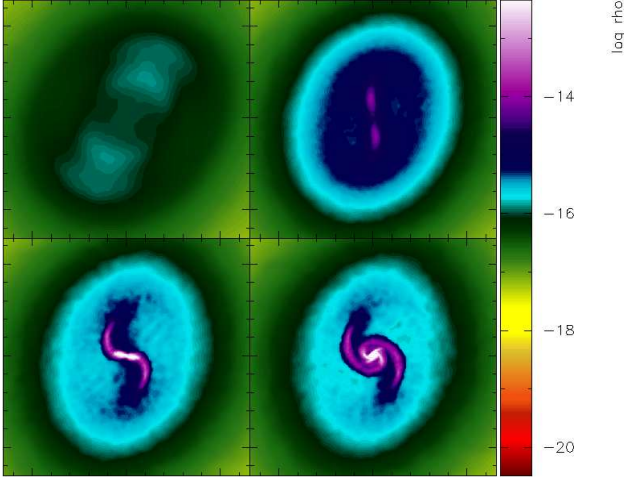


Fig. 30. Evolution of the density field in the central region for the test B1. Plotted is the region within 1.54×10^{16} cm from the origin, on the xy plane. Left to right, top to bottom: $t = 1.0, 1.15, 1.23, 1.26$, in units of free fall time, $t_{ff} = 1.0774 \times 10^{12}$ s.

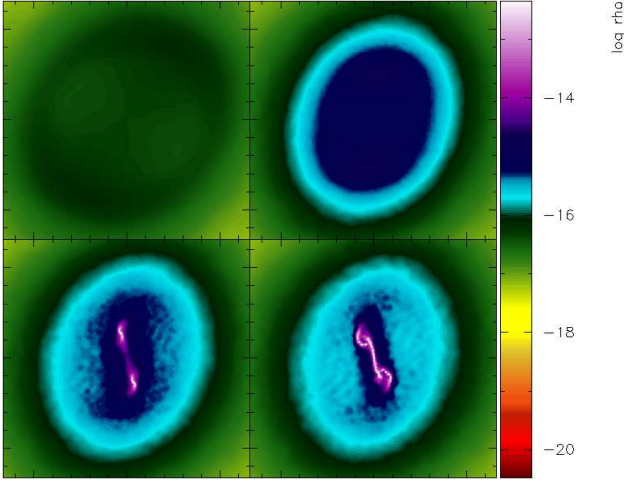


Fig. 31. The same as in Fig. 30, but for test B2.

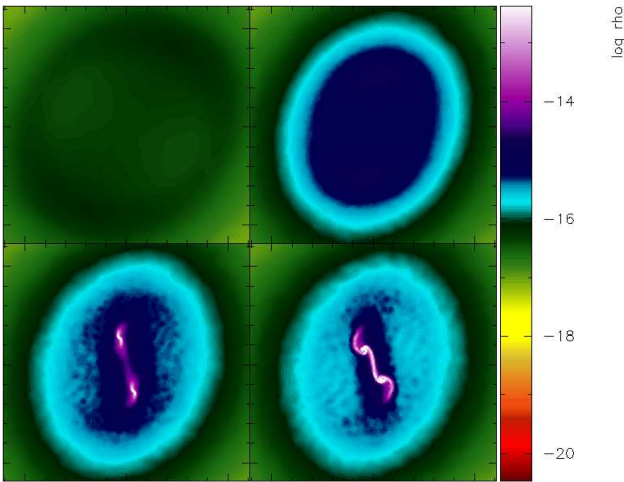


Fig. 32. The same as in Fig. 30, but for test B3.

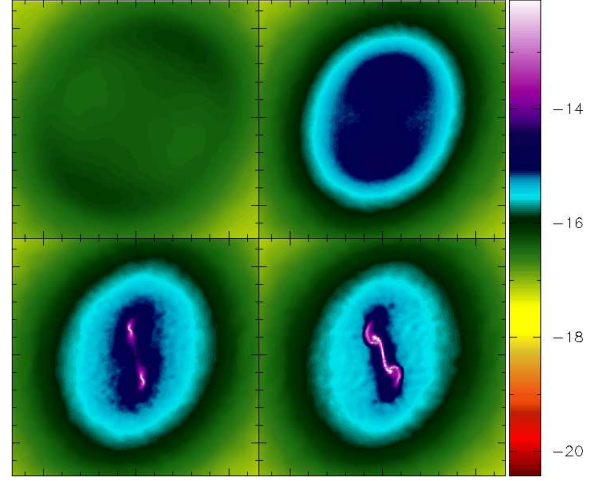


Fig. 33. The same as in Fig. 30, but for test B4.

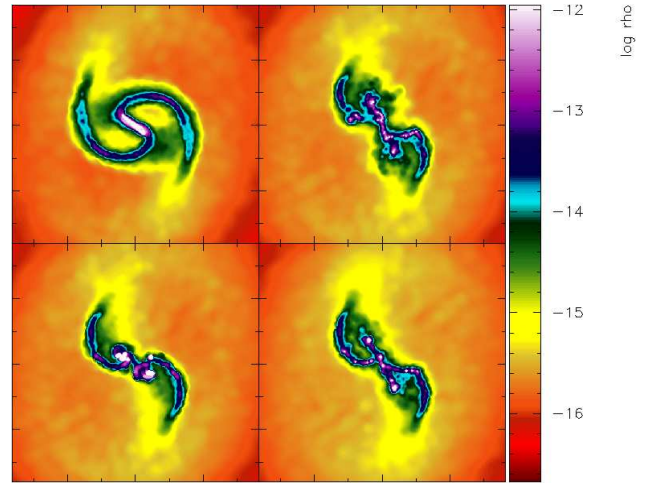


Fig. 34. Final density field at $t = 1.29$ (in units of free fall time) in the central region. Left to right, top to bottom: tests B1, B2, B3, B4. See text for details.

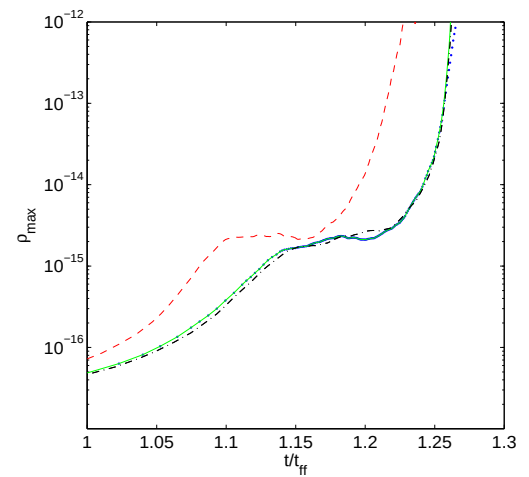


Fig. 35. Maximum density in the isothermal collapse tests. B1: red dashed line; B2: green solid line; B3: blue thick dots; B4: black dot-dashed line. See text for details.

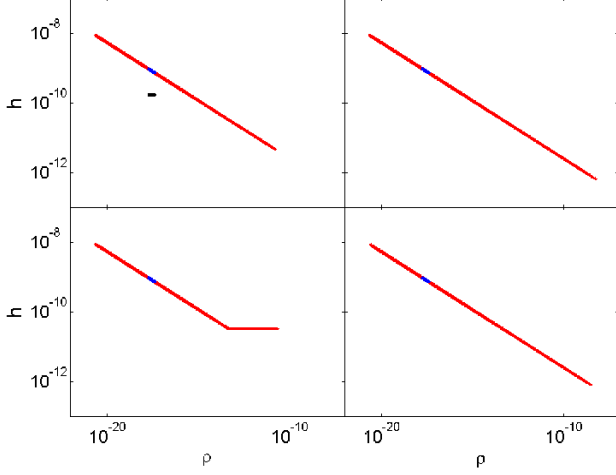


Fig. 36. Initial (blue) and final (red) smoothing length h of all particles in the four isothermal collapse tests (left to right, top to bottom: B1, B2, B3, B4). The softening length ϵ is equal to h in the B2, B3 and B4 runs; the black dots in the top left panel show its constant value in the test B1. The abrupt truncation in the bottom left panel is due to the imposed h_{min} ($= \epsilon_{min}$) in the test B3.

with single CPU. Although the agreement is good, and no sign of a multi-processor subdivision of the domain is present, some differences in local morphological features are present; these are more evident looking at Fig. 38, where the comparison is made for the smoothing lengths values. While the gross features and the minimum and maximum values are very similar, differences in the local clumping of particles are clearly present.

We ascribe this flaw to the parallel Oct-Tree scheme. The global tree constructed by $N > 1$ CPUs is slightly different from the one built by a single processor on the same spatial domain: the regions in which two or more different CPUs interact are described by a different cell architecture depending on N (note that the reason for this is that the cells have to be always cubic; indeed, the problem should not occur if a binary tree, in which cells have adaptive shapes, was used). This causes a slightly different approximation of the gravitational force, due to the opening cell criterion, giving in turn a slightly different dynamical evolution.

To prove the validity of this speculation, we run again the same tests reducing the opening angle θ to 0.1 instead of the standard 0.8. In this way, the approximation given by the adoption of the tree structure should become of negligible importance (at the expenses of a much longer computation time). Indeed, looking at Figs. 39 and 40, we can notice an improvement in that the density fields are more similar, although still not exactly identical. An ideal run with $\theta = 0$ (i.e., exact particle-particle interaction) should give almost identical distributions of particles.

We must point out that this flaw may be of non-negligible importance when dealing with phenomena that directly depend on the local density field, such as the star

formation and feedback processes. The same simulation run on a different number of CPUs may give slightly different results because of this issue.

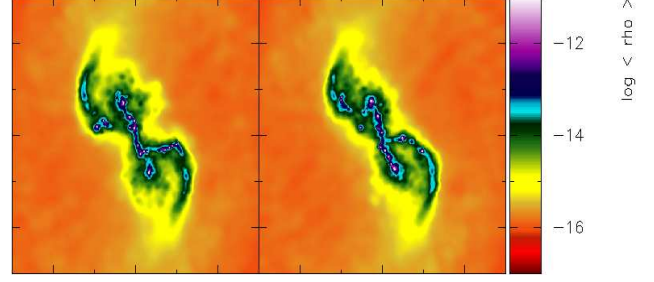


Fig. 37. Comparison between the final density fields for a single CPU (left) and a 16 parallel CPUs (right) runs of the B2 test. See the text for details.

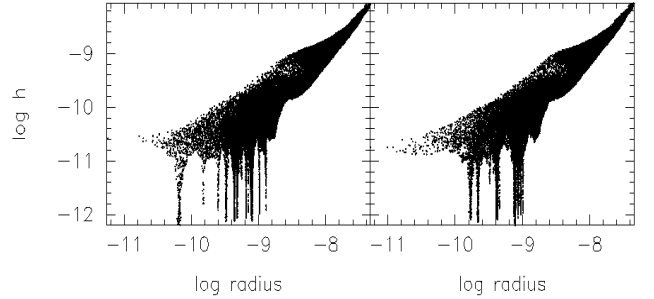


Fig. 38. Comparison between the final h -radius relation, for a single CPU (left) and a 16 parallel CPUs (right) runs of the B2 test. See the text for details.

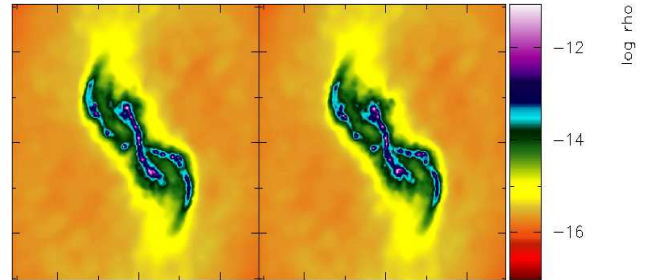


Fig. 39. Comparison between the final density fields for a single CPU (left) and a 16 parallel CPUs (right) runs of the B2 test, in a run with tree opening angle $\theta = 0.1$. See the text for details.

5.10. Collision of two gas spheres

The collision between two gaseous Plummer spheres is very crucial test for the energy conservation. The tests are run in 3-D, with adaptive softening lengths, variable α viscosity, and thermal conduction.

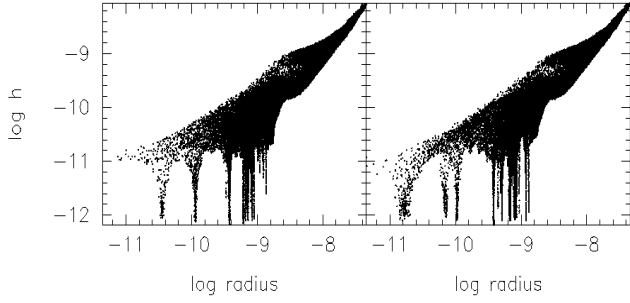


Fig. 40. Comparison between the final h -radius relation, for a single CPU (left) and a 16 parallel CPUs (right) runs of the B2 test, in a run with tree opening angle $\theta = 0.1$. See the text for details.

Each of the two spheres (of unit mass, unit radius, and negligible initial internal energy) are set up randomly selecting 10,000 particles in a way that their initial density radial profile resembles the Plummer profile, $\rho \propto (1 + r)^{-5/2}$. Their centres are on the x axis.

In a first run (C1) we follow a head-on collision, assigning a relative velocity of 1.5 in the x direction to the spheres. In a second run (C2) we added a shear component, giving relative velocities $|\Delta v_x| = 0.15$ and $|\Delta v_y| = 0.075$.

According to Hernquist (1993) the violent head-on collision of two polytropic stars does not conserve the energy within $\sim 10\%$. Modern formulations of SPH (see e.g. Rosswog & Price 2007) can handle the problem with much more accuracy, reducing the error below 0.1%.

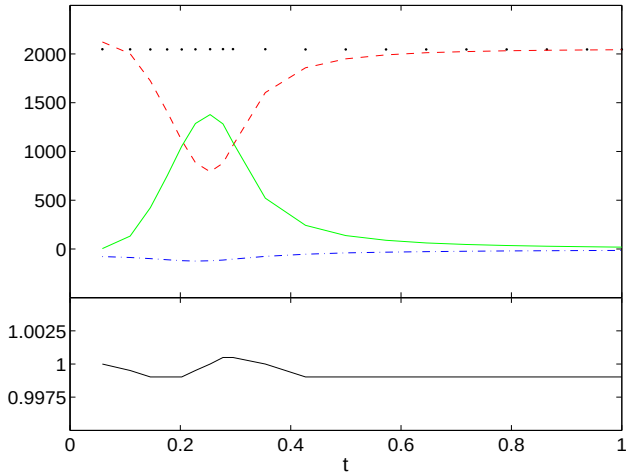


Fig. 42. Top: energy trends in the collision test C1; black dots: total energy, blue dot-dashed line: potential, red dashed line: kinetic, green solid line: thermal. Bottom: magnification of the total energy conservation $E(t)/E(t_0)$.

Figs. 41 and 43 plot the density and velocity fields at different times, on a slice taken at $z = 0$. In our tests, the energy conservation is good (see Figs. 42 and 44), with a

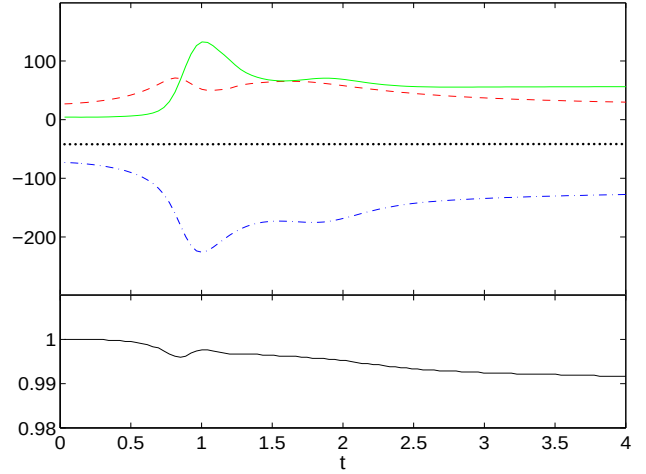


Fig. 44. Top: energy trends in the collision test C2; black dots: total energy, blue dot-dashed line: potential, red dashed line: kinetic, green solid line: thermal. Bottom: magnification of the total energy conservation $E(t)/E(t_0)$.

maximum error of $\sim 0.2\%$ in the head-on collision, and $\sim 1\%$ error when the shear component is added.

5.11. Two-components evolution

As a final test, we studied the evolution of a two-components fluid in order to investigate the behaviour of the adaptive smoothing and softening length algorithm in presence of more than one material. We set up two spheres of the same radius, $R = 10^3$ pc, and mass, $M = 4 \times 10^4 M_\odot$, using for each sphere the particle distribution adopted adiabatic collapse test by Evrard. One of the two systems was considered as made of collisionless bodies, with negligible pressure, thus mimicking the behaviour of a Dark Matter halo; the other one was instead considered as made of gas, with an initial temperature of 1000 K. The system was let free to evolve under the action of self-gravity and hydrodynamical interactions, in four different runs with the following settings:

- TC1 - constant softening lengths, $\epsilon_i = 0.1R \times (m_i/M)^{0.2}$; null initial velocities
- TC2 - adaptive softening lengths, null initial velocities
- TC3 - as TC2, with an initial solid body rotation $\Omega = 2.5 \times 10^{-17}$ rad s $^{-1}$, for both components
- TC4 - as TC3, but with counter-rotating initial tangential velocities (same magnitude).

While the collisionless component soon collapses under self-gravity, later reaching relaxation, the gas undergoes a phase of expansion because of its internal pressure, but in the central region, where it also interacts with collapsing Dark Matter, reaching very high densities.

The free expansion of a system of particles is by itself a demanding test, and a loss of total energy is a common flaw for Lagrangian codes. Furthermore, the rotation imposed in the last two runs makes the problem more com-

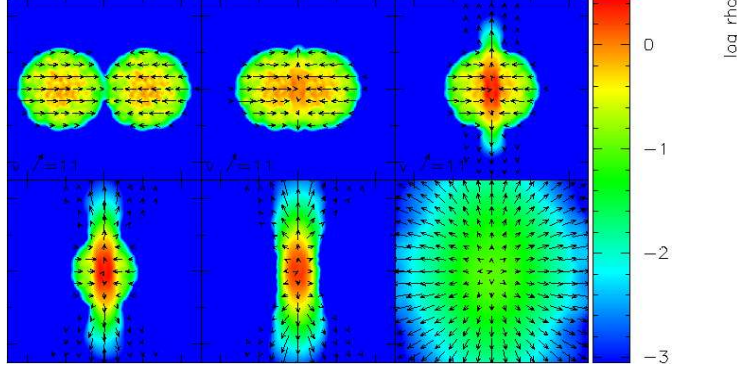


Fig. 41. Evolution of the density field in the $z = 0$ plane for the collision test C1. Left to right, top to bottom: $t = 0.05, 0.15, 0.2, 0.25, 0.3, 0.5$. Superposed is the velocity field.

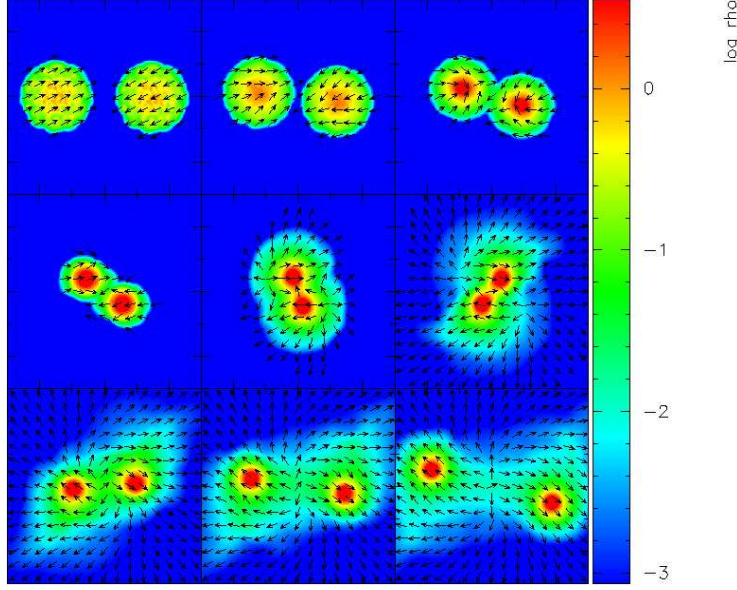


Fig. 43. Evolution of the density field in the $z = 0$ plane for the collision test C2. Left to right, top to bottom: $t = 0.03, 0.38, 0.77, 1.16, 1.55, 1.94, 2.72, 3.5, 4.29$. Superposed is the velocity field.

plicated. In our tests, a further complication was given by the dynamical interaction within the central regions of the system, where the hydrodynamical pressure of the gas struggled against self-gravity and the (stronger) gravitational action of the collisionless component. Moreover, when the adaptive ϵ scheme was used, gas particles had to cope with a large variation of their interaction sphere, due to the presence of Dark Matter particles in the central regions, and to their absence in the outskirts, where in addition gas particles are far away from one another.

Fig. 45 plots the energy trends for all the four tests. The conservation is very good in all cases, with a maximum error well below 1%; quite surprisingly, the worst performance is given by the constant ϵ scheme, while the corresponding test with adaptive softening performs better staying below a $\sim 0.5\%$ error.

Fig. 46 shows the evolution of the Dark Matter component in the test TC3, projected on the xy and on the xz planes (the case TC4 run gives almost undistinguish-

able plots). Clear is the early formation of a disk due to the initial rotational velocity, and a subsequent expansion which leaves a dense core with a loosely disk-like shape. On the other hand, the gas component expands with almost radial motion (Fig. 47), but in the central regions the interaction with the collisionless component plays an important role. Indeed, looking at Fig. 48 where a magnification of the velocity field in the central region at late times is shown for the tests TC3 and TC4, the difference in the two cases is clear: in the counter-rotation run the gas in the innermost zone has inverted its sense of motion and co-rotates with the Dark Matter, creating a complex pattern of velocities.

The conservation of angular momenta in the tests TC3 and TC4 is shown in Fig. 49. We find a maximum error of $\sim 6\%$ in the relative error for the total momentum in the counter-rotation test; in this case the absolute violation is anyway small because of the very small total initial mo-

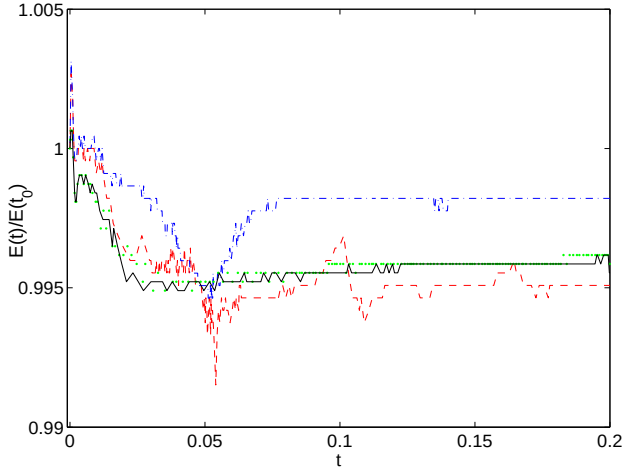


Fig. 45. Two components test, energy conservation $E(t)/E(t_0)$ for the four runs: dashed red line, TC1; blue dot-dashed line, TC2; green dots, TC3; black solid line, TC4.

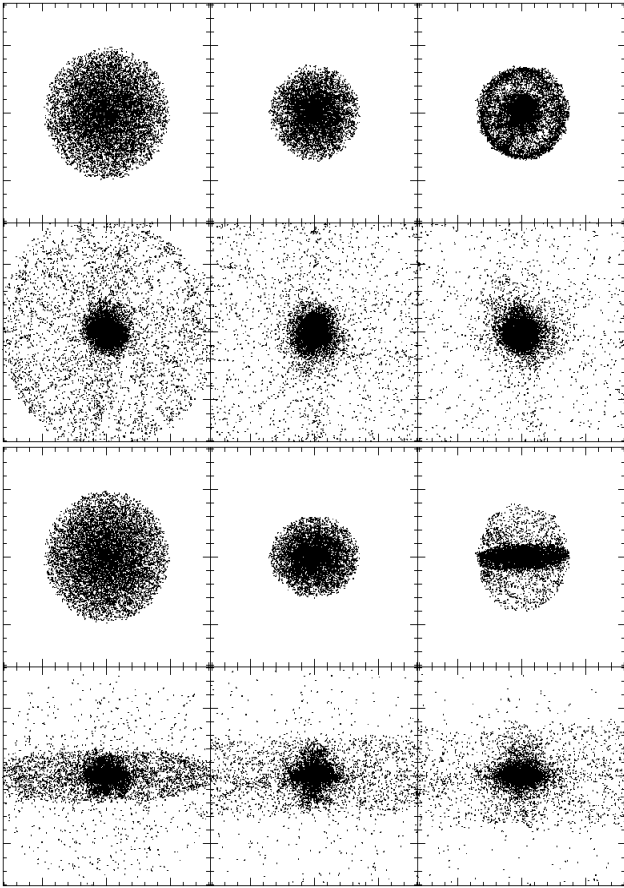


Fig. 46. Two components test, case TC3. Dark Matter particles projected onto the xy (top panels) and xz (bottom panels) planes: left to right, top to bottom, $t = 0.5, 1.0, 1.5, 2.0, 2.5$ Gyr.

momentum. In all the other cases the conservation is always very satisfactory.

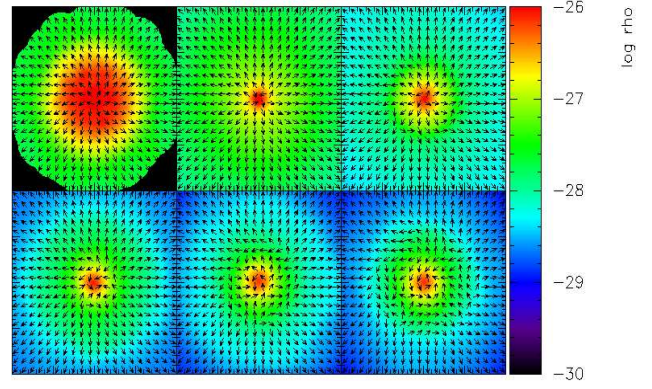


Fig. 47. Two components test, case TC3. Gas density and velocity fields in a slice taken at $z = 0$: left to right, top to bottom, $t = 0.5, 1.0, 1.5, 2.0, 2.5$ Gyr.

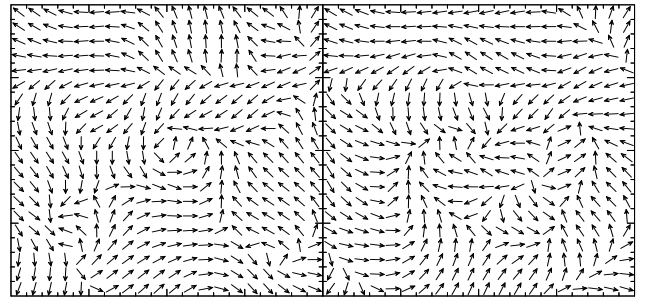


Fig. 48. Two components test: comparison between the central velocity fields in cases TC3 (left) and TC4(right), in a slice taken at $z = 0$, at $t = 2.5$ Gyr.

6. Conclusions

We have presented the basic features of EvoL, the new release of the Padova parallel N-body code for cosmological simulations of galaxy formation and evolution. In this first paper, the standard gravitational and hydrodynamical algorithms have been extensively reviewed and discussed, as well as the parallel architecture and the data structures of the code. EvoL includes some interesting options such as adaptive softening lengths with self-consistent extra-terms to avoid large errors in energy conservation, ∇h terms in the SPH formalism, variable α viscosity formulation, and artificial thermal conduction terms to smooth out pressure at contact discontinuities.

We have also performed and presented an extended series of standard hydrodynamical tests to check the ability of the code to handle potentially difficult situations. The results are encouraging. Almost all tests have given results in nice agreement with theoretical expectations and previous calculations with similar codes, sometimes even showing better performance. In particular, we showed how the inclusion of an artificial thermal conduction term as suggested by Price (2008) significantly improves the modeling of demanding problems such as the Kelvin-Helmoltz instability or the Sedov-Taylor point-like explosion. Furthermore, the adoption of a variable softening

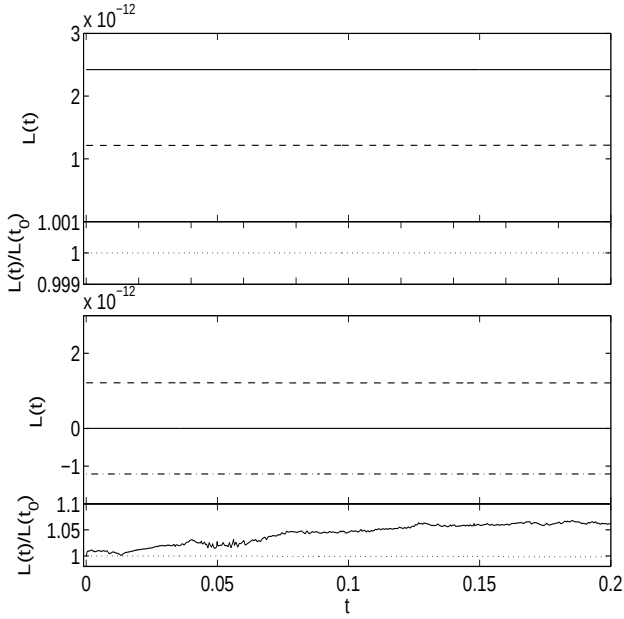


Fig. 49. Two components test, angular momenta conservations. Top panels: test TC3, bottom panels: test TC4. Shown are the total (solid lines), gas (dashed lines) and Dark Matter (dash-dotted lines) angular momenta as a function of time $L(t)$, and conservations as a function of time $L(t)/L(t_0)$ (same meaning of the symbols). Note that in the TC3 test the gas and Dark Matter lines are superposed because they have identical angular momentum.

lengths algorithm allows for a higher degree of adaptivity without resulting in appreciable losses in terms of precision and conservation of energy. While some typical flaws of SPH codes are still present (e.g., problems in the conservation of vorticity because of spurious shear viscosity), these new features clearly improve the method and positively aid in curing well-known drawbacks of the SPH algorithm.

It must be however pointed out that the new features must be extensively tested, especially in situations of interest for cosmological simulations (i.e., gas dynamics in presence of a Dark Matter and/or stellar components). Here, we have restricted our analysis to standard hydrodynamical problems. Other and new tests will be presented in the companion paper by Merlin et al. (2009, in preparation), dedicated to the inclusion in EvoL of radiative cooling, chemical evolution, star formation, energy feedback, and other non-standard algorithms.

Acknowledgements. The authors would like to thank D.J. Price, S. Borgani, P.A. Thomas, L. Secco, S. Pasetto and G. Tormen for the useful discussions and for their help. All simulations were run using the CINECA facilities and/or the MONSTER cluster at the Department of Astronomy (Padova). The pictures have been produced using MATLAB or SPLASH by D. Price (2007).

References

- Agertz, O., Moore, B., Stadel, J., Potter, D., Miniati, F., Read, J., Mayer, L., Gawryszczak, A., Kravtsov, A., Nordlund, Å., Pearce, F., Quilis, V., Rudd, D., Springel, V., Stone, J., Tasker, E., Teyssier, R., Wadsley, J., & Walder, R. 2007, *MNRAS*, 380, 963
- Alimi, J.-M., Serna, A., Pastor, C., & Bernabeu, G. 2003, *Journal of Computational Physics*, 192, 157
- Athanassoula, E., Fady, E., Lambert, J. C., & Bosma, A. 2000, *MNRAS*, 314, 475
- Balsara, D. S. 1995, *Journal of Computational Physics*, 121, 357
- Barnes, J. & Hut, P. 1986, *Nat*, 324, 446
- Bate, M. R. & Burkert, A. 1997, *MNRAS*, 288, 1060
- Benz, W. 1990, in *Numerical Modelling of Nonlinear Stellar Pulsations Problems and Prospects*, ed. J. R. Buchler, 269–+
- Boss, A. P. & Bodenheimer, P. 1979, *ApJ*, 234, 289
- Brookshaw, L. 1986, *Proceedings of the Astronomical Society of Australia*, 6, 461
- Burkert, A. & Bodenheimer, P. 1993, *MNRAS*, 264, 798
- Carraro, G., Lia, C., & Chiosi, C. 1998, *MNRAS*, 297, 1021
- Couchman, H. M. P., Thomas, P. A., & Pearce, F. R. 1995, *ApJ*, 452, 797
- Dehnen, W. 2001, *MNRAS*, 324, 273
- Dolag, K., Vazza, F., Brunetti, G., & Tormen, G. 2005, *MNRAS*, 364, 753
- Dubinski, J. 1996, *New Astronomy*, 1, 133
- Einfeldt, B., Roe, P. L., Munz, C. D., & Sjogreen, B. 1991, *Journal of Computational Physics*, 92, 273
- Evrard, A. E. 1988, *MNRAS*, 235, 911
- Ewald, P. P. 1921, *Annalen der Physik*, 369, 253
- Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNeice, P., Rosner, R., Truran, J. W., & Tufo, H. 2000, *ApJS*, 131, 273
- Gingold, R. A. & Monaghan, J. J. 1977, *MNRAS*, 181, 375
- Gresho, P. M. & Chan, S. T. 1990, *International Journal for Numerical Methods in Fluids*, 11, 621
- Harfst, S., Theis, C., & Hensler, G. 2006, *A&A*, 449, 509
- Hernquist, L. 1993, *ApJ*, 404, 717
- Hernquist, L., Bouchet, F. R., & Suto, Y. 1991, *ApJS*, 75, 231
- Hernquist, L. & Katz, N. 1989, *ApJS*, 70, 419
- Hu, X. Y. & Adams, N. A. 2005, *Journal of Computational Physics*, 213, 844
- Lia, C., Portinari, L., & Carraro, G. 2002, *MNRAS*, 330, 821
- Liska, W. & Wendroff, B. 2003, *SIAM Journal on Scientific Computing*, 25, 995
- Lucy, L. B. 1977, *AJ*, 82, 1013
- McMillan, S. L. W. & Aarseth, S. J. 1993, *ApJ*, 414, 200
- Merlin, E. & Chiosi, C. 2007, *A&A*, 473, 733
- Merritt, D. 1996, *AJ*, 111, 2462
- Monaghan, J. J. 1988, *Computer Physics Communications*, 48, 89

- . 1992, *ARA&A*, 30, 543
- . 1997, *Journal of Computational Physics*, 136, 298
- . 2000, *Journal of Computational Physics*, 159, 290
- . 2002, *MNRAS*, 335, 843
- . 2006, *MNRAS*, 365, 199
- Monaghan, J. J. & Gingold, R. A. 1983, *Journal of Computational Physics*, 52, 374
- Monaghan, J. J. & Lattanzio, J. C. 1985, *A&A*, 149, 135
- Morris, J. & Monaghan, J. J. 1997, *Journal of Computational Physics*, 136, 41
- Nelson, R. P. & Papaloizou, J. C. B. 1994, *MNRAS*, 270, 1
- Noh, W. F. 1987, *Journal of Computational Physics*, 72, 78
- Norman, M. L., Bryan, G. L., Harkness, R., Bordner, J., Reynolds, D., O'Shea, B., & Wagner, R. 2007, *ArXiv e-prints*
- Ott, F. & Schnetter, E. 2003, *ArXiv Astrophysics e-prints*, astro-ph/0303112
- Price, D. J. 2007, *Publications of the Astronomical Society of Australia*, 24, 159
- . 2008, *Journal of Computational Physics*, 227, 10040
- Price, D. J. & Monaghan, J. J. 2005, *MNRAS*, 364, 384
- . 2007, *MNRAS*, 374, 1347
- Romeo, A. B. 1998, *A&A*, 335, 922
- Rosswog, S. 2009, *ArXiv e-prints*
- Rosswog, S. & Price, D. 2007, *MNRAS*, 379, 915
- Saitoh, T. R. & Makino, J. 2008, *ArXiv e-prints*
- Salmon, J. K. & Warren, M. S. 1994, *Journal of Computational Physics*, 111, 136
- Schuessler, I. & Schmitt, D. 1981, *A&A*, 97, 373
- Serna, A., Alimi, J.-M., & Chieze, J.-P. 1996, *ApJ*, 461, 884
- Shirokov, A. 2007, *ArXiv e-prints*
- Sod, G. A. 1978, *Journal of Computational Physics*, 27, 1
- Springel, V. 2005, *MNRAS*, 364, 1105
- . 2009, *ArXiv Astrophysics e-prints*, astro-ph/9506070
- Springel, V. & Hernquist, L. 2002, *MNRAS*, 333, 649
- Springel, V., Yoshida, N., & White, S. D. M. 2001, *New Astronomy*, 6, 79
- Steinmetz, M. & Mueller, E. 1993, *A&A*, 268, 391
- Thacker, R. J. & Couchman, H. M. P. 2000, *ApJ*, 545, 728
- Thomas, P. A. & Couchman, H. M. P. 1992, *MNRAS*, 257, 11
- Wadsley, J. W., Stadel, J., & Quinn, T. 2004, *New Astronomy*, 9, 137
- Watkins, S. J., Bhattal, A. S., Francis, N., Turner, J. A., & Whitworth, A. P. 1996, *A&AS*, 119, 177
- Wetzstein, M., Nelson, A. F., Naab, T., & Burkert, A. 2008, *ArXiv e-prints*
- Williams, P. R., Churches, D. K., & Nelson, A. H. 2004, *ApJ*, 607, 1
- Woodward, P. & Colella, P. 1984, *Journal of Computational Physics*, 54, 115

Appendix A: N -dimensional kernel

The general form for the N -dimensional spline kernel function ($1 \leq N \leq 3$) is

$$W_N(r, \epsilon) = \eta_N \times \begin{cases} \frac{2}{3} - u^2 + \frac{1}{2}u^3 & \text{if } 0 \leq u < 1, \\ \frac{1}{6}(2-u)^3 & \text{if } 1 \leq u < 2, \\ 0 & \text{if } u \geq 2, \end{cases} \quad (\text{A.1})$$

where

$$\eta_N = \frac{1}{h^N} \times \begin{cases} 1 & \text{if } N = 1, \\ 15/(7\pi) & \text{if } N = 2, \\ 3/(2\pi) & \text{if } N = 3. \end{cases} \quad (\text{A.2})$$

Derivatives are straightforwardly obtained from this expression. The 1-D and 2-D kernels have been used where necessary in some of the hydrodynamical tests presented in Sect. 5.

Appendix B: Summary of equations of motion and conservation

Gravitational acceleration:

$$\begin{aligned} \frac{d\mathbf{v}_{i,grav}}{dt} = & \quad (\text{B.1}) \\ & - \sum_j m_j \left[\frac{\phi'_{ij}(\epsilon_i) + \phi'_{ij}(\epsilon_j)}{2} \right] \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} \\ & - \sum_j m_j \frac{1}{2} \left[\frac{\xi_i}{\Upsilon_i} \frac{\partial W_{ij}(\epsilon_i)}{\partial \mathbf{r}_i} + \frac{\xi_j}{\Upsilon_j} \frac{\partial W_{ij}(\epsilon_i)}{\partial \mathbf{r}_j} \right], \end{aligned}$$

Hydrodynamical acceleration (for SPH particles only):

$$\begin{aligned} \frac{d\mathbf{v}_{i,hyd}}{dt} = & \sum_j m_j \times \quad (\text{B.2}) \\ & \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\xi_i/m_j}{\Omega_i^*} \right) \nabla_i W_{ij}(h_i) + \right. \\ & \left. \frac{P_j}{\rho_j^2} \left(1 + \frac{\xi_j/m_i}{\Omega_j^*} \right) \nabla_i W_{ij}(h_j) + \Pi_{ij} \nabla_i \bar{W}_{ij} \right], \end{aligned}$$

Specific internal energy evolution (for SPH particles only):

$$\begin{aligned} \frac{du_i}{dt} = & \quad (\text{B.3}) \\ & \sum_j m_j \left[\frac{P_i}{\rho_i^2} \left(1 + \frac{\xi_i/m_j}{\Omega_i^*} \right) (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla_i W_{ij}(h_i) \right. \\ & \left. + \left(\frac{1}{2} \Pi_{ij} + \Pi_{ij}^u \right) w_{ij} |\nabla_i \bar{W}_{ij}| \right]. \end{aligned}$$

In these expressions,

$$\Omega_i^* = 1 - \frac{\partial h_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(h_i)}{\partial h_i}, \quad (\text{B.4})$$

$$\Upsilon_i = \left[1 - \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial W_{ij}(\epsilon_i)}{\partial \epsilon_i} \right], \quad (\text{B.5})$$

$$\xi_i = \frac{\partial \epsilon_i}{\partial n_i} \sum_j \frac{\partial \phi_{ij}(\epsilon_i)}{\partial \epsilon_i}, \quad (\text{B.6})$$

and

$$\zeta_i = \frac{\partial h_i}{\partial n_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i}. \quad (\text{B.7})$$

The smoothing kernel W is given by Eq. 4, whereas the softened gravitational potential ϕ is given by Eq. 8. The standard equation of state is that of an ideal gas,

$$P = (\gamma - 1)\rho u, \quad (\text{B.8})$$

where γ is the adiabatic index (5/3 for a monatomic gas); a more general equation of state will be presented in Merlin et al. (2009, in preparation).

ROBO: a model and a code for studying the interstellar medium

T. Grassi¹, P. Krstić², E. Merlin¹, U. Buonomo¹, L. Piovan¹, C. Chiosi¹

¹ Department of Astronomy, Padova University, Vicolo dell'Osservatorio 3, I-35122, Padova, Italy

² Physics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6372, USA

e-mail: tommaso.grassi@unipd.it

Received: November 2009; Revised: April 2011; Accepted: May 2011

Abstract. We present ROBO, a model and its companion code for the study of the interstellar medium (ISM). The aim is to provide an accurate description of the physical evolution of the ISM and to set the ground for an ancillary tool to be inserted in NBody-Tree-SPH (NB-TSPH) simulations of large-scale structures in the cosmological context or of the formation and evolution of individual galaxies. The ISM model consists of gas and dust. The gas chemical composition is regulated by a network of reactions that includes a large number of species (hydrogen and deuterium-based molecules, helium, and metals). New reaction rates for the charge transfer in H^+ and H_2 collisions are presented. The dust contains the standard mixture of carbonaceous grains (graphite grains and PAHs) and silicates. In our model dust are formed and destroyed by several processes. The model accurately treats the cooling process, based on several physical mechanisms, and cooling functions recently reported in the literature. The model is applied to a wide range of the input parameters and the results for important quantities describing the physical state of the gas and dust are presented. The results are organized in a database suited to the artificial neural networks (ANNs). Once trained, the ANNs yield the same results obtained by ROBO with great accuracy. We plan to develop ANNs suitably tailored for applications to NB-TSPH simulations of cosmological structures and/or galaxies.

Key words. ISM: evolution - ISM: dust - galaxies: formation and evolution - methods: numerical

1. Introduction

Modeling the gas chemistry is an important step towards correctly describing the growth of cosmological structures, the formation and evolution of galaxies, and star formation in general. For instance, the molecular hydrogen is one of the most efficient coolants, and its abundance eventually determines the total amount of stars in the Universe. Structure growth and galaxy formation and evolution are customarily investigated by means of large numerical simulations in which a wide set of chemical reactions taking place in the ISM should be considered to get and follow the key molecules (elemental species in general) eventually governing the efficiency of the star formation and gas cooling. However, we must face the growing standard complexity of a typical NB-TSPH model that includes particles of dark matter, particles of baryonic matter (this in the form of stars and gas, in turn divided into several thermal and chemical phases, such as (i) cold, warm and hot, (ii) atomic and molecular, (iii) neutral and ionized), sources of energy heating and cooling, energy feedback, and easily many other physical processes. For this reason, too, a detailed chemical description of the ISM would drastically

reduce the computational performances of any numerical algorithm (code) that one may adopt to this purpose. This requires a strategy for optimizing the chemical accuracy of the ISM model and the computational speed.

In this paper we present a new model of the ISM and the associated code we have developed to explore the ISM properties over wide ranges of the physical parameters and, at the same time, to cope with the above difficulties. The model and companion code are named ROBO¹.

The model deals with an ideal ISM element of unit volume, containing gas and dust in arbitrary initial proportions, whose initial physical conditions are specified by a set of parameters, which is allowed to evolve for a given time interval. The history leading the element to that particular initial physical state is not of interest here. The ISM element is mechanically isolated from the host environment; i.e. it does not expand or contract under the action of large-scale forces. It can, however, be interested by the passage of shock waves caused by physical phenomena taking place elsewhere (e.g. supernova explosions). Furthermore, it neither acquires nor loses material,

¹ The name means “thing” in some northern Italian dialects.

so the conservation of total mass applies, even if its chemical composition can change with time. It is immersed in a bath of UV radiation generated either by nearby or internal stellar sources and in a field of cosmic ray radiation. It can generate its own radiation field by internal processes and so it has its own temperature, density, and pressure, each related by an Equation of State (EoS). If observed from outside, it would radiate with a certain spectral energy distribution. For the aims of this study, we do not need to know the whole spectral energy distribution of the radiation field pervading the element, but only its UV component. Given these hypotheses and the initial conditions, the ISM element evolves toward another physical state under the action of the internal network of chemical reactions changing the relative abundances of the elemental species and molecules, the internal heating and cooling processes, the UV radiation field, the field of cosmic rays and the passage of shock waves. In view of the future applications of this model in dynamical simulations of galaxies, the integration time interval is chosen in such a way that (i) it is long enough to secure that the secular evolution of the gas properties is achieved, and (ii) it is short enough to secure that the physical properties of the ISM at each instant are nearly independent of the external variations in the large-scale properties of the system hosting the ISM element. In other words, ROBO associates a final state (another point in the same space) to any initial state (a point in the multidimensional space of the physical parameters) through a path. The model is like an operator determining the vector field of the local transformations of the ISM from an initial state to a final one in the space of the physical parameters. This is the greatest merit of this approach, which secures the wide applicability of the model.

The new ISM model stands on recipes of the internal physical processes falling in between those developed by Glover & Jappsen (2007) and Smith et al. (2008). The first one follows the thermal and chemical evolution of the low-metallicity gas in large numerical simulations. The chemical network includes a detailed treatment of H and He but neglects the molecules formed with heavy elements as the CO molecule. Dust is included to compute its contribution to the formation of molecular hydrogen, but its evolution is not calculated. The chemical code is an on-the-fly routine, running as part of a wider code following cosmological simulations of structure growth.

The model proposed by Smith et al. (2008) uses the non-equilibrium treatment for hydrogen-like species and the standard equilibrium approximation for all the remaining chemical species. It does not take any type of dust into account. To calculate the cooling rates, Smith et al. (2008) use CLOUDY (Ferland et al. 1998) and get the cooling rates as a function of temperature, density, and metallicity. By doing this, it is possible to include a large chemical network and a wide set of coolants, but the price to pay is that several oversimplifications of the problem are mandatory, e.g. the assumption of ionization equilibrium. A similar approach has been proposed by Hocuk & Spaans

(2010) using the Meijerink & Spaans (2005) PDR model instead of CLOUDY.

Our ISM model and associated code ROBO can not only describe the gas evolution in great detail but also includes large chemical networks and the presence of various types of dust which follow the chemistry and the complex interplay between grain destruction and formation. Many gas and dust components are taken into account, among which we recall the molecular hydrogen and the metal coolants (Santoro & Shull 2006; Maio et al. 2007) or HD (McGreer & Bryan 2008). To track the evolution of these components, the ISM model and ROBO take various physical processes into account that may affect the behavior of the whole system. For instance, dust is very efficient in forming both H₂ and HD (Cazaux et al. 2008). Including dust formation and destruction is a formidable task. Among other processes, dust can be destroyed by shocks that deserve special care to be properly modeled. Here we have considered two different approaches. The first one makes use of a mean shock speed that is assumed to be the same for all the gas particles, neglecting the effects of the environment. In reality this is too crude a description. The second approach starts from the notions that the shocks develop when the motion of the gas particles becomes turbulent and that the distribution of turbulent velocities obeys the one predicted by the Kolmogorov law. This is significantly better than the previous case, so it is the approach we prefer.

Finally, we would like to include the gas model (and results of ROBO) in numerical simulations of galaxies in a simple way. We suppose that a numerical code calculates the formation and evolution of a model galaxy from an initial stage at high redshift using the standard NB-TSPH technique. At each time step, it requires an update of the chemical status of the gas particles. This is done for every gas particle of the simulation (typically from 10⁴ up to 10⁷ particles, depending on the simulation under consideration). We have two methods to our disposal. The first one is a real-time chemical updater. This approach must use simple physics and a powerful computer in order to save computing time. The second one is to use model grids, calculated in advance for a wide range of the input parameters, in such a way as to cover the plausible space of the initial conditions. Since increasing the parameters also increases the space dimensions of the grids, thus making data handling cumbersome, we make use of the ANNs technique to get rid of this difficulty. Once the ANNs are instructed to reproduce the (ROBO) results as a function of the parameters, they should replace ROBO in the NB-TSPH simulator of galaxies. In our case the NB-TSPH model is EvoL by Merlin & Chiosi (2006, 2007) and Merlin et al. (2010). The use of ANNs can greatly improve upon this point of difficulty. Here we briefly touch upon this problem and leave the detailed discussion of it to a forthcoming paper (Grassi et al. 2011).

The plan of the paper is as follows. In Section 2 we give a detailed description of the physics behind the ISM model and ROBO. This section is divided in three parts:

Table 1. Correspondence between elemental species or free particles and the indices of the differential equations governing the reaction network.

Ele.s	H	H ⁺	H ⁻	H ₂	H ₂ ⁺	
Ind.s	1	2	3	4	5	
Ele.s	D	D ⁺	D ⁻	D ₂	HD	HD ⁺
Ind.s	6	7	8	9	10	11
Ele.s	He	He ⁺	He ⁺⁺			
Ind.s	12	13	14			
Ele.s	C	C ⁺	CH	CH ₂	CH ₂ ⁺	CH ₃ ⁺ CO
Ind.s	15	16	17	18	19	20 21
Ele.s	O	O ⁺				
Ind.s	22	23				
Ele.s	Si	Si ⁺	Fe	Fe ⁺	e ⁻	
Ind.s	24	25	26	27	28	

the chemistry of the gas phase, the presence of dust grains of different types (including their formation by accretion and destruction), and finally the heating and cooling processes. In Section 3 we describe the general characteristics of the code. Section 4 is dedicated to describing the results of the calculations run to validate ROBO. Section 5 describes how to include the results of ROBO in the NB-TSPH simulations. Finally, some concluding remarks are presented in Section 6.

2. Physical model of the ISM

In this section we describe the physics of the ISM model that is used by ROBO. The problem and associated code are divided into three parts, mirrored by the structure of this section: gas chemistry, dust, and cooling. All these aspects are mutually coupled and overlapped.

2.1. The chemical network for the ISM

The kernel of the numerical code modelling the gas chemistry is the network of chemical reactions among different elemental species (atoms and molecules, neutral or ionized dust grains of different types) and free particles (such as the electrons), the network of photochemical reactions between the above elemental species, and the radiation field. In addition to this, we must include all the processes creating and destroying molecules and dust grains with particular attention to those that are most efficient for cooling the gas. For this reason our chemical network follows species like H₂, HD, and metals (C, O, Si, Fe, and their ions).

The number of reactions depends on how many species are tracked and how many details are included in their description. A number of codes study the behavior of the ISM (Cen 1992; Katz et al. 1996; Galli & Palla 1998; Anninos et al. 1997; Glover & Savin 2009), each of which has a different number of species to follow and a different degree of sophistication for the physical processes taken into consideration.

We keep track of the following 27 elemental species or molecules plus the free electrons: H, H⁺, H⁻, H₂, H₂⁺, D, D⁺, D⁻, D₂, HD, HD⁺, He, He⁺, He⁺⁺, C, C⁺, CH, CH₂, CH₂⁺, CH₃⁺, CO, O, O⁺, Si, Si⁺, Fe, Fe⁺, and e⁻.

These species are divided into four groups. The first one contains hydrogen-only based species. The second group is composed of deuterium-based species, in which HD plays the key role in the gas cooling. The third group lists helium and its ions. Carbon, oxygen, silicon, and iron with their ions and compounds form the fourth group. The free electrons link all the four groups together. The species from CH through CH₃⁺ are introduced to follow the formation and destruction of CO to be described below.

The reactions in which all the above species are involved are

- collisional ionization ($A + e^- \rightarrow A^+ + 2e^-$),
- photo-recombination ($A^+ + e^- \rightarrow A + \gamma$),
- dissociative recombination ($A_2^+ + e^- \rightarrow 2A$),
- charge transfer ($A^+ + B \rightarrow A + B^+$),
- radiative attachment ($A + e^- \rightarrow A^- + \gamma$),
- dissociative attachment ($A + B^- \rightarrow AB + e^-$),
- collisional detachment ($A^- + e^- \rightarrow A + 2e^-$),
- mutual neutralization ($A^+ + B^- \rightarrow A + B$),
- isotopic exchange ($A_2^+ + B \rightarrow AB^+ + A$),
- dissociations by cosmic rays ($AB + CR \rightarrow A + B$),
- neutral-neutral ($AB + AB \rightarrow A_2 + B_2$),
- ion-neutral ($AB^+ + AB \rightarrow AB_2^+ + A$),
- collider ($AB + C \rightarrow A + B + C$),
- ionizations by field photons ($A + \gamma \rightarrow A^+ + e^-$)

where A and B are two generic atoms. To these reactions we must add those regulating the abundance of CO to be described below.

The chemical network governing the ISM model is a classical system of differential equations in which each equation is a Cauchy problem of the form

$$\frac{dn_i(t)}{dt} = \sum_{lm} R_{lm}(T)n_l(t)n_m(t) - \sum_j R_{ij}(T)n_i(t)n_j(t), \quad (1)$$

where $n_i(t)$ is the number density of the i -th species with known initial value $n_i(0)$. In eqn. (1) $R_{lm}(T)$ is the rate of the reaction between the l -th and the m -th species expressed in units of cm³/s. Eqn. (1) is written as the sum of all the reactions forming the i -th species ($\sum_{lm} R_{lm}(T)n_l(t)n_m(t)$) minus the sum of all the reactions destroying the i -th species ($\sum_j R_{ij}(T)n_i(t)n_j(t)$). The indices i, j, l , and m run from 1 to 28 as in the list of elemental species and free electrons already mentioned. The one-to-one correspondence between the indices in the eqn. (1) and elemental species is given in Table 1.

To clarify the meaning of eqn. (1) we show the case of a two-reaction system



Table 2. The reaction rates among hydrogen, deuterium and helium. References: (a) Glover & Savin (2009); (b) Glover & Abel (2008). More details within the text.

	Reaction	Reaction Rate (cm ³ /s)	Ref
1	H ₂ + D → H + HD	if($T \leq 2 \times 10^3$ K) $k_1 = \text{dex}[-56.4737 + 5.88886 \log(T) + 7.196292 \log(T)^2 + 2.25069 \log(T)^3 - 2.16903 \log(T)^4 + 0.317887 \log(T)^5]$ if($T > 2 \times 10^3$ K) $k_1 = 3.17 \times 10^{-10} \exp(-5207/T)$	a
2	HD + H → H ₂ + D	if($T \leq 200$ K) $k_2 = 5.25 \times 10^{-11} \exp(-4430/T)$ if($T > 200$ K) $k_2 = 5.25 \times 10^{-11} \exp[(-4430/T) + (173900/T^5)]$	a
3	D ⁺ + H ₂ → H ⁺ + HD	$k_3 = 10^{-9} [0.417 + 0.846 \log(T) - 0.137 \log(T)^2]$	a
4	H ⁺ + HD → D ⁺ + H ₂	$k_4 = 1.1 \times 10^{-9} \exp(-488/T)$	a
5	H ⁺ + D → D ⁺ + H	if($T \leq 2 \times 10^5$ K) $k_5 = 2 \times 10^{-10} T^{0.402} \exp(-37.1/T) - 3.31 \times 10^{-17} T^{1.48}$ if($T > 2 \times 10^5$ K) $k_5 = 3.44 \times 10^{-10} T^{0.35}$	a
6	H + D ⁺ → D + H ⁺	$k_6 = 2.06 \times 10^{-10} T^{0.396} \exp(-33/T) + 2.03 \times 10^{-9} T^{-.332}$	a
7	HD + D ⁺ → D ₂ + H ⁺	$k_7 = 10^{-9}$	a
8	H ⁺ + D ₂ → D ⁺ + HD	$k_8 = 2.1 \times 10^{-9} \exp(-491/T)$	a
9	H ₂ + H ⁺ → H ₂ ⁺ + H	$k_9 = 10^A$ with $A = \sum_{i=0}^7 a_i \log(T)^i$, with a_i as in Table 8	
10	H + H ⁺ → H ₂ ⁺ + γ	$k_{10} = \text{dex}[-19.38 - 1.523 \log(T) + 1.118 \log(T)^2 - 0.1269 \log(T)^3]$	a
11	H + H ₂ ⁺ → H ₂ + H ⁺	$k_{11} = 6.4 \times 10^{-10}$	a
12	H + HD ⁺ → H ₂ + D ⁺	$k_{12} = 10^{-9}$	a
13	H ⁺ + e ⁻ → H + γ	$k_{13} = 2.753 \times 10^{-14} (315614/T)^{1.5} [1 + (1115188/T)^{0.407}]^{-2.242}$	a
14	D ⁺ + e ⁻ → D + γ	$k_{14} = k_{13}$	a
15	H ₂ ⁺ + e ⁻ → 2H	if($T \leq 617$ K) $k_{15} = 10^{-8}$ if($T > 617$ K) $k_{15} = 1.32 \times 10^{-6} T^{-0.76}$	a
16	HD ⁺ + e ⁻ → H + D	$k_{16} = 7.2 \times 10^{-8} T^{-0.5}$	a
17	H + e ⁻ → H ⁻ + γ	$k_{17} = \text{dex}[-17.845 + 0.762 \log(T) + 0.1523 \log(T)^2 - 0.03274 \log(T)^3]$	a
18	D + e ⁻ → D ⁻ + γ	$k_{18} = k_{17}$	a
19	H + e ⁻ → H ⁺ + 2e ⁻	$k_{19} = \exp[-32.71396786 + 13.5365560 \ln(T_e) - 5.73932875 \ln(T_e)^2 + 1.56315498 \ln(T_e)^3 - 0.287705600 \ln(T_e)^4 + 3.48255977 \times 10^{-2} \ln(T_e)^5 - 2.63197617 \times 10^{-3} \ln(T_e)^6 + 1.11954395 \times 10^{-4} \ln(T_e)^7 - 2.03914985 \times 10^{-6} \ln(T_e)^8]$	a
20	D + e ⁻ → D ⁺ + 2e ⁻	$k_{20} = k_{19}$	a
21	H ⁻ + H → H ₂ + e ⁻	$k_{21} = 10^{-9} \xi$, $\xi = [0.65, 5.0]$,	b
22	D ⁻ + H → HD + e ⁻	$k_{22} = 10^{-9} \xi/2$, $\xi = [0.65, 5.0]$	b
23	H ⁻ + D → HD + e ⁻	$k_{23} = k_{22}$	b
24	D ⁻ + D → D ₂ + e ⁻	$k_{24} = k_{21}$	b
25	H ⁺ + D ⁻ → HD ⁺ + e ⁻	$k_{25} = 1.1 \times 10^{-9} (T/300)^{-0.41}$	a
26	D ⁺ + H ⁻ → HD ⁺ + e ⁻	$k_{26} = k_{25}$	a
27	H ⁻ + e ⁻ → H + 2e ⁻	$k_{27} = \exp[-18.01849334 + 2.36085220 \ln(T_e) - 0.282744300 \ln(T_e)^2 + 1.62331664 \times 10^{-2} \ln(T_e)^3 - 3.36501203 \times 10^{-2} \ln(T_e)^4 + 1.17832978 \times 10^{-2} \ln(T_e)^5 - 1.65619470 \times 10^{-3} \ln(T_e)^6 + 1.06827520 \times 10^{-4} \ln(T_e)^7 - 2.63128581 \times 10^{-6} \ln(T_e)^8]$	b

where A_i is the number density of the generic species in units of cm^{-3} . Looking at the density variation of the species A_0 over the time step dt the density n_{A_0} changes as

$$dn_{A_0}(t + dt) = -k_{01}(T)n_{A_0}(t)n_{A_1}(t)dt + k_{45}(T)n_{A_4}(t)n_{A_5}(t)dt. \quad (3)$$

In this equation the positive term is due to the second reaction (between A_4 and A_5) that increases the total quantity of A_0 (and also A_3), while the negative term is due to the first reaction in which A_0 is destroyed because it reacts with A_1 to form A_2 and A_3 .

We adopt here a minimal description containing only the 28 species and/or the molecular compounds listed

in Table 1, the 64 reactions listed in Tables 2 and 3 (where reactions among hydrogen, deuterium, and helium are considered), in Table 4 (where metals and cosmic rays are involved), in Table 5 (where the reactions of the CO mini-network are listed), and finally, the 12 photochemical processes listed in Table 6. In all the tables the gas temperature T is in Kelvin; the electron temperature T_e is in eV. In Table 4 the cosmic ray field is $\zeta_{\text{CR}} \text{ s}^{-1}$ and it is the rate of H₂ ionization by cosmic rays. For Table 5 we have $T_{300} = T/300$. The expressions in Table 6 are $\xi = E/E_{\text{th}}$, $\Phi_{\text{ph}}(a_f, y_w, x_f, y_a, P) = [(x_f - 1)^2 + y_w^2]y^{0.5P-5.5}[1 + \sqrt{y/y_a}]^{-P}$, $y = \sqrt{x_f^2 + a_f^2}$, $x_f(a, b) = E/a - b$, and $x_{Si} = 1.672 \times 10^{-5}$.

Table 3. Continuation of Table 2. References: (a) Glover & Savin (2009); (b) Glover & Abel (2008); (c) Abel et al. (1997).

	Reaction	Reaction Rate (cm ³ /s)	Ref
28	$\text{H}^- + \text{H} \rightarrow 2\text{H} + \text{e}^-$	if($T_e \leq 0.1$ eV) $k_{28} = 2.5634 \times 10^{-9} T_e^{1.78186}$ if($T_e > 0.1$ eV) $k_{28} = \exp[-20.372609 + 1.13944933 \ln(T_e) - 0.14210135 \ln(T_e)^2 + 8.4644554 \times 10^{-3} \ln(T_e)^3 - 1.4327641 \times 10^{-3} \ln(T_e)^4 + 2.0122503 \times 10^{-4} \ln(T_e)^5 + 8.6639632 \times 10^{-5} \ln(T_e)^6 - 2.5850097 \times 10^{-5} \ln(T_e)^7 + 2.4555012 \times 10^{-6} \ln(T_e)^8 - 8.0683825 \times 10^{-9} \ln(T_e)^9]$	b
29	$\text{H}^- + \text{H}^+ \rightarrow 2\text{H}$	if($T \leq 10^4$ K) $k_{29} = 2.4 \times 10^{-6} T^{-0.5} (1 + 2 \times 10^{-4} T)$	b
30	$\text{H}_2^+ + \text{H}^- \rightarrow \text{H}_2 + \text{H}$	$k_{30} = 5 \times 10^{-7} \sqrt{100/T}$	b
31	$\text{H}^- + \text{H}^+ \rightarrow \text{H}_2^+ + \text{e}^-$	if($T_e \geq 1.719$ eV) $k_{31} = 8.4258 \times 10^{-10} T_e^{-1.4} \exp(-1.301/T_e)$ if($T_e < 1.719$ eV) $k_{31} = 2.291 \times 10^{-10} T_e^{-0.4}$	b
32	$\text{H}_2 + \text{e}^- \rightarrow 2\text{H} + \text{e}^-$	$k_{32} = 5.6 \times 10^{-11} \sqrt{T} \exp(-102124/T)$	b
33	$\text{H}_2 + \text{H} \rightarrow 3\text{H}$	$k_{33} = 1.067 \times 10^{-10} T_e^{2.012} \exp[-(4.463/T_e)(1 + 0.2472 T_e)^{3.512}]$	b
34	$\text{He} + \text{e}^- \rightarrow \text{He}^+ + 2\text{e}^-$	$k_{34} = \exp[-44.09864886 + 23.91596563 \ln(T_e) - 10.7532302 \ln(T_e)^2 + 3.05803875 \ln(T_e)^3 - 0.56851189 \ln(T_e)^4 + 6.79539123 \times 10^{-2} \ln(T_e)^5 - 5.00905610 \times 10^{-3} \ln(T_e)^6 + 2.06723616 \times 10^{-4} \ln(T_e)^7 - 3.64916141 \times 10^{-6} \ln(T_e)^8]$	b
35	$\text{He}^+ + \text{e}^- \rightarrow \text{He} + \gamma$	$k_{35} = k_{35}^d + k_{35}^e$ $k_{35}^d = 1.544 \times 10^{-9} T_e^{-1.5} \exp(-48.596/T_e) \times [0.3 + \exp(-8.1/T_e)]$ $k_{35}^e = 3.925 \times 10^{-13} T_e^{-0.6353}$	c
36	$\text{He}^+ + \text{e}^- \rightarrow \text{He}^{++} + 2\text{e}^-$	$k_{36} = \exp[-68.71040990 + 43.93347633 \ln(T_e) - 18.4806699 \ln(T_e)^2 + 4.70162649 \ln(T_e)^3 - 0.76924663 \ln(T_e)^4 + 8.113042 \times 10^{-2} \ln(T_e)^5 - 5.32402063 \times 10^{-3} \ln(T_e)^6 + 1.97570531 \times 10^{-4} \ln(T_e)^7 - 3.16558106 \times 10^{-6} \ln(T_e)^8]$	b
37	$\text{He}^{++} + \text{e}^- \rightarrow \text{He}^+ + \gamma$	$k_{37} = 3.36 \times 10^{-10} T^{-0.5} (T/1000)^{-0.2} [1 + (T/10^6)^{0.7}]^{-1}$	b

The various types of reaction are described below in some detail. Tables 2, 3, 4, and 5 also sample the chemical reactions according to the same four groups in which we separated the elemental species depending on the type of reaction and the information at our disposal to derive the reactions rates. In doing this, the notation in use may appear rather complex. This is done on purpose, and the main motivation is that we want to keep the same formalism as adopted in the literature for easy comparison. First of all, there are 64 reactions among particles, whether atoms or molecules or free electrons. Each reaction is identified by a progressive number from 1 to 64 and so the associated reaction rate named k_n (with n from 1 to 64). Each reaction will be a term in the righthand of the 28 differential equations of system (1) and the associated reaction rate will coincide with one of the R_{ij} terms of eqns. 1). To establish the exact correspondence is a matter of patient work, which is not the prime interest here. Most of these reaction rates are taken from Abel et al. (1997) and Glover & Savin (2009) to whom we refer for all the details. Some of these reactions are discussed in this section, and we pay attention to those rates that are vividly debated in literature.

Minimal reaction network for CO. Since this molecule plays a key role in determining the properties of the ISM, we have included a simplified network of reactions taken from Nelson & Langer (1997) to follow the creation/destruction of the CO in detail. These reactions

are listed in Table 5 together with the corresponding rates of Woodall et al. (2007). Note that $T_{300} = T/300$, with T in K.

Table 5 shows how CO is formed from C^+ and O species, using CH, CH_2 , CH_2^+ and CH_3^+ as intermediate products/reactants, and H_2 and e^- as main reactants.

Our model includes UV radiation, so we must consider the photodissociation of the molecules included in the network. The main effects of the radiation are on the carbon-based molecules and the corresponding photo-ionization rates are

$$\Gamma_{\text{CO}} = G_0 10^{-10} \text{ s}^{-1}, \quad (4)$$

and

$$\Gamma_{\text{CH}_x^{(+)}} = G_0 5 \times 10^{-10} \text{ s}^{-1}, \quad (5)$$

where G_0 is the ratio of the adopted UV flux to that by Habing; i.e. $I_{\text{Hab}} = 1.2 \times 10^{-4} \text{ erg/cm}^2/\text{s/sr}$. The expression $\text{CH}_x^{(+)}$ indicates that we are dealing with the CH, CH_2 , CH_2^+ , and CH_3^+ molecules.

Basing on eqn. (5), we must consider the products of all the $\text{CH}_x^{(+)}$ molecules present in reactions like $\text{CH}_x^{(+)} + \gamma \rightarrow \text{products}$. However, at this stage we are not interested in the detail of these reactions; therefore, following Nelson & Langer (1997), we introduce the parameter $\beta(G_0)$ controlling the efficiency of the reactions from 58 through 61 in Table 5. This parameter is defined as $\beta(G_0) = \exp[G_0 \cdot \ln(\xi)]$ with $\xi = 5 \times 10^{-10}$ from eqn. (5). In the absence of the UV radiation ($G_0 = 0$), we get

Table 4. Reaction rates for the processes where the metals are involved and where cosmic rays (CR) interact with the ISM. References: (c) Verner & Ferland (1996); (d) Voronov (1997); (e) Glover & Jappsen (2007); (f) Zhao et al. (2004); (g) Walmsley et al. (2004). More details within the text.

	Reaction	Reaction Rate (cm ³ /s)	Ref
38	$C^+ + e^- \rightarrow C + \gamma$	$k_{38} = \Phi_{\text{rec}}(T, 6.85 \times 10^{-8}, 0, 11.3, 0.193, 0.25)$	c
39	$C + e^- \rightarrow C^+ + 2e^-$	$k_{39} = \Phi_{\text{col}}(T_e, 6.556 \times 10^{-10}, 65.23, 2.446 \times 10^7, 0.7567)$	d
40	$Si^+ + e^- \rightarrow Si + \gamma$	$k_{40} = \Phi_{\text{rec}}(T, 3.59 \times 10^{-8}, 0, 13.6, 0.073, 0.34)$	c
41	$Si + e^- \rightarrow Si^+ + 2e^-$	$k_{41} = \Phi_{\text{col}}(T_e, 8.616 \times 10^{-10}, 119.1, 4.352 \times 10^7, 0.7563)$	d
42	$O^+ + e^- \rightarrow O + \gamma$	$k_{42} = \Phi_{\text{rec}}(T, 1.88 \times 10^{-7}, 1, 8.2, 0.376, 0.25)$	c
43	$O + e^- \rightarrow O^+ + 2e^-$	$k_{43} = \Phi_{\text{col}}(T_e, 1.517 \times 10^{-9}, 360.1, 1.329 \times 10^8, 0.7574)$	d
44	$Fe^+ + e^- \rightarrow Fe + \gamma$	$k_{44} = \Phi_{\text{rec}}(T, 2.52 \times 10^{-7}, 0, 7.9, 0.701, 0.25)$	c
45	$Fe + e^- \rightarrow Fe^+ + 2e^-$	$k_{45} = \Phi_{\text{col}}(T_e, 2.735 \times 10^{-9}, 1314.0, 4.659 \times 10^8, 0.7568)$	d
46	$O^+ + H \rightarrow O + H^+$	$k_{46} = 4.99 \times 10^{-11} T^{0.405} + 7.54 \times 10^{-10} T^{-0.458}$	e
47	$O + H^+ \rightarrow O^+ + H$	$k_{47} = [1.08 \times 10^{-11} T^{0.517} + 4 \times 10^{-10} T^{0.00669}] \exp(-227/T)$	e
48	$O + He^+ \rightarrow O^+ + He$	if($T < 6000$ K) $k_{48} = 4.991 \times 10^{-15} \left(\frac{T}{10^4}\right)^{0.3794} \exp\left(-\frac{T}{1.121 \times 10^6}\right)$ if($T \geq 6000$ K) $k_{48} = 2.780 \times 10^{-15} \left(\frac{T}{10^4}\right)^{-0.2163} \exp\left(\frac{T}{8.158 \times 10^5}\right)$	e f
49	$C + H^+ \rightarrow C^+ + H$	$k_{49} = 3.9 \times 10^{-16} T^{0.213}$	e
50	$C^+ + H \rightarrow C + H^+$	$k_{50} = 6.08 \times 10^{-14} \left(\frac{T}{10^4}\right)^{1.96} \exp\left(-\frac{1.7 \times 10^5}{T}\right)$	e
51	$C + He^+ \rightarrow C^+ + He$	if($T \leq 200$ K) $k_{51} = 8.58 \times 10^{-17} T^{0.757}$ if($200 < T \leq 2000$ K) $k_{51} = 3.25 \times 10^{-17} T^{0.968}$ if($T > 2000$ K) $k_{51} = 2.77 \times 10^{-19} T^{1.597}$	e
52	$Si + H^+ \rightarrow Si^+ + H$	if($T \leq 10^4$ K) $k_{52} = 5.88 \times 10^{-13} T^{0.848}$ if($T > 10^4$ K) $k_{52} = 1.45 \times 10^{-13} T$	e
53	$Si + He^+ \rightarrow Si^+ + He$	$k_{53} = 3.3 \times 10^{-9}$	e
54	$Si + C^+ \rightarrow Si^+ + C$	$k_{54} = 2.1 \times 10^{-9}$	e
55	$H + CR \rightarrow H^+ + e^-$	$k_{56} = 0.46 \zeta_{\text{CR}}$	g
56	$H_2 + CR \rightarrow 2H$	$k_{57} = 1.50 \zeta_{\text{CR}}$	g
57	$H_2 + CR \rightarrow H_2^+ + e^-$	$k_{58} = 0.96 \zeta_{\text{CR}}$	g

Table 5. Reaction rates for the reactions belonging to the CO network. References: (a) Woodall et al. (2007); (b) Petuchowski et al. (1989). See the text for more details.

	Reaction	Reaction Rate (cm ³ /s)	Ref
58.	$C^+ + H_2 \rightarrow CH_2^+ + \gamma$	$k_{58} = 4 \times 10^{-16} T_{300}^{-0.2}$	a
59.	$CH_2^+ + e^- \rightarrow CH + H$	$k_{59} = 1.6 \times 10^{-7} T_{300}^{-0.6}$	a
60.	$CH_2^+ + H_2 \rightarrow CH_3^+ + H$	$k_{60} = 1.6 \times 10^{-4}$	a
61.	$CH_3^+ + e^- \rightarrow CH_2 + H$	$k_{61} = 6.6 \times 10^{-11}$	a
62.	$CH + O \rightarrow CO + H$	$k_{62} = 7.58 \times 10^{-8} T_{300}^{-0.5}$	a
63.	$CH_2 + O \rightarrow CO + 2H$	$k_{63} = 1.33 \times 10^{-10}$	a
64.	$CO + e^- \rightarrow C + O + e^-$	$k_{64} = 2.86 \times 10^{-3} T_{300}^{-3.52} \exp(-112700/T)$	b

$\beta = 1$: this means that the existing photons cannot affect the formation of the various $CH_x^{(+)}$. Instead, when $G_0 = 1$, it follows $\beta = \xi$, which means that the formation reactions have reduced their efficiency. Finally, when $G_0 \rightarrow \infty$, (unphysical) then $\beta \rightarrow 0$, which means that the $CH_x^{(+)}$ molecules are completely destroyed before they can interact. The coefficient of the i -th reaction now is $k'_i = \beta \cdot k_i$, with $i \in [58, 61]$ and eqn. (5) is not included in the code.

Photochemistry. In addition to this, we consider the group of photochemical processes listed in Table 6 for which we provide the cross section of the reaction and

the analytical expression to derive the reaction rate as a function of the existing radiation field.

In the present model of the ISM we do not include the chemical reactions with lithium and its compounds (e.g. LiH and LiH⁺) because according to Prieto et al. (2008) and Mizusawa et al. (2005), they are not important coolants. After we neglecting lithium and compounds, the number of species to follow, including metals and deuterium compounds, amounts to the 28 we have considered.

There is some interest in the CO molecule because it is an important coolant at very low temperatures. Among others, recipes for the formation of the interstellar CO are

presented by Ruffle et al. (2002) and Glover et al. (2010). However, explicitly following the formation of the CO molecule would be too complicated for our aims. See below for the cooling rate by the CO molecule.

In our chemical network we also include the effect of the cosmic ray radiation field. For convenience, we list in Table 4 the ionization rates for cosmic rays given by Walmsley et al. (2004). Cosmic rays are important for the cooling because they can destroy molecules that contribute to it, thus influencing the temperature of the interstellar medium (see the reactions shown in Table 4). In particular, cosmic rays are able to destroy H_2 and HD and to ionize atoms. The role of the cosmic rays is still largely unknown because the strength of their radiation field in various regions and epochs of the Universe is not firmly determined. While their effect can be explored in the vicinity of a strong source, for a random sample of the Universe there are no exact measurements of the cosmic ray radiation field. Furthermore, most NB-TSPH simulators like EVoL do not contain the full description of cosmic rays, but simply consider their presence as a parameter. Therefore, even if these reactions have been considered and their rates presented in Table 4, they have not been included in practice in the system of equations (1).

PHOTOCHEMICAL REACTIONS. For the photochemical reactions we adopt the model and rates of Glover & Jappsen (2007) and Verner & Ferland (1996). In general the reaction rate is given by

$$R_{\text{photo}} = 4\pi \int_{E_{\text{th}}}^{\infty} \frac{\sigma(E)J(E)}{E} e^{-\tau(E)} [1 + f(E)] dE, \quad (6)$$

where $J(E) = J(h\nu)$ is the energy flux per unit frequency and solid angle of the impinging radiation field, $\sigma(E)$ is the cross-section, $\tau(E)$ the gas opacity at the energy E , and $f(E)$ a numerical factor accounting for the effects of the secondary ionization, which are negligible if the UV radiation is not dominated by X-rays (Glover & Jappsen 2007). The rate is in s^{-1} .

The cross sections $\sigma(E)$ are expressed in two different ways according to the reaction under consideration. The reactions and associated rates are listed in Table 6. For the reactions from 1 through 8 we note that (i) the quantity $\epsilon = \sqrt{(E/E_{\text{th}}) - 1}$; (ii) for H_2 and HD photodissociation, we use the model proposed by Glover & Jappsen (2007). We have

$$\sigma_{\text{H}_2} = 1.38 \times 10^9 J(h\nu) \quad (7)$$

without considering self-shielding. For the HD we have

$$\sigma_{\text{HD}} = 1.5 \times 10^9 J(h\nu), \quad (8)$$

with $h\nu = 12.87 \text{ eV}$.

For the remaining reactions involving metals (i.e. from 9 through 12), the rates contain the fits given by Verner & Ferland (1996) using the expression

$$\Phi_{\text{ph}}(a_f, y_w, x_f, y_a, P) = [(x_f - 1)^2 + y_w^2] \times y^{0.5P-5.5} \left[1 + \sqrt{\frac{y}{y_a}} \right]^{-P}, \quad (9)$$

where $y = \sqrt{x_f^2 + a_f^2}$. The meaning of the various symbols is given in Table 6. More details on the reactions and companion quantities and symbols are given by Glover & Jappsen (2007) and Verner & Ferland (1996). See also the references therein.

The UV radiation field is calculated as in Efstathiou (1992), Vedel et al. (1994), and Navarro & Steinmetz (1997). In particular we have

$$J(\nu) = 10^{-21} J_{21}(z) \left(\frac{\nu_{\text{H}}}{\nu} \right)^{\alpha_{\text{UV}}}, \quad (10)$$

where z is the redshift, $\nu_{\text{H}} = E_{\text{H}}/h$ is the frequency corresponding to the hydrogen first-level energy threshold, $\alpha_{\text{UV}} = 1$ and $J_{21}(z)$ is

$$J_{21}(z) = \frac{J_{21}}{1 + [5/(1+z)]^4}, \quad (11)$$

where in our case $J_{21} = 1$. Finally, to derive the reaction rates, we integrate eqn. (6) from $E = E_{\text{th}}$ to $E = \infty$ over the energy range of the radiation field photons.

The integrals must be calculated at each time step if the radiation field changes with time or once for all at the beginning of the simulation if the radiation field remains constant. The integrals are calculated using Romberg's integration method.

HEATING BY PHOTODISSOCIATION. Heating by photodissociation of molecular hydrogen and UV pumping, H and He photo-ionization, H_2 formation in the gas and dust phase, and finally ionization from cosmic rays are modeled as in Glover & Jappsen (2007). All these processes are listed in Table 7 where n_{cr} is the critical density and R_{d} is the photodissociation rate. To describe the heating by the photoelectric effect we adopt the model proposed by Bakes & Tielens (1994) and discuss it separately in the Section 2.4.1 below.

The H and He photo-ionizations increase the gas energy at the rate

$$\Gamma = 4\pi \int_{E_0}^{\infty} \frac{\sigma(E)J(E)}{E} e^{-\tau(E)} (E - E_0) \eta(E - E_0) dE, \quad (12)$$

where $\sigma(E)$ is the reaction cross-section, $J(E)$ is the photon flux, $e^{-\tau(E)}$ is the optical depth for a photon of energy E , η is the efficiency of the process (i.e. the fraction of energy converted to heat), and $(E - E_{\text{th}})$ is the difference between the energy of the photon and the energy of the atomic ground level. The rates Γ are given in Table 7.

2.2. Reaction by reaction: notes on a few cases

In this section we examine in some detail a few chemical reactions that are widely debated and are affected by

Table 6. Cross sections of photochemical processes in cm^2 . The energy E is in eV. The various quantities in use are described within the text. References: (a) Glover & Jappsen (2007); (b) Verner & Ferland (1996). See these papers for further references.

	Reaction (cm^3/s)	Cross Section	Note	Ref
1	$\text{H} + \gamma \rightarrow \text{H}^+ + \text{e}^-$	$\sigma_1 = 6.3 \times 10^{-18} \xi^4 \exp(4 - 4\epsilon^{-1} - \arctan \epsilon) [1 - \exp(-\frac{2\pi}{\epsilon})]^{-1}$	$E_{th} = 13.6$	a
2	$\text{D} + \gamma \rightarrow \text{D}^+ + \text{e}^-$	$\sigma_2 = \sigma_1$	$E_{th} = 13.6$	a
3	$\text{He} + \gamma \rightarrow \text{He}^+ + \text{e}^-$	$\sigma_3 = 3.1451 \times 10^{-16} \xi^{7/2} [1.0 - 4.7416 \xi^{1/2} + 14.82 \xi - 30.8678 \xi^{3/2} + 37.3584 \xi^2 - 23.4585 \xi^{5/2} + 5.9133 \xi^3]$	$E_{th} = 24.6$	a
4	$\text{H}^- + \gamma \rightarrow \text{H} + \text{e}^-$	$\sigma_4 = 2.11 \times 10^{-16} (E - E_{th})^{3/2} E^{-3}$	$E_{th} = 0.755$	a
5	$\text{H}_2 + \gamma \rightarrow \text{H} + \text{H}$	see text		
6	$\text{H}_2^+ + \gamma \rightarrow \text{H} + \text{H}^+$	if $(2.65 < E < 11.27)$ $\sigma_5 = \text{dex}[-40.97 + 15.9795 \xi - 3.53934 \xi^2 + 0.2581155 \xi^3]$ if $(11.27 < E < 21)$ $\sigma_5 = \text{dex}[-30.26 + 7.3935 \xi - 1.29214 \xi^2 + 0.065785 \xi^3]$	$E_{th} = 2.65$	a
7	$\text{H}_2 + \gamma \rightarrow \text{H}_2^+ + \text{e}^-$	if $(15.4 < E < 16.5)$ $\sigma_6 = 9.560 \times 10^{-17} \xi - 9.4 \times 10^{-17}$ if $(16.5 < E < 17.7)$ $\sigma_6 = 2.16 \times 10^{-17} \xi - 1.48 \times 10^{-17}$ if $(17.7 < E < 30.0)$ $\sigma_6 = 1.51 \times 10^{-17} \xi - 2.71$	$E_{th} = 15.4$	a
8	$\text{HD} + \gamma \rightarrow \text{H} + \text{D}$	see text		
9	$\text{C} + \gamma \rightarrow \text{C}^+ + \text{e}^-$	$\sigma_7 = 5.027 \times 10^{-16} \Phi_{\text{ph}}(1.607, 0.09157, x_f, 62.16, 5.101)$	$E_{th} = 11.26$ $x_f(2.144, 1.133)$	a;b
10	$\text{O} + \gamma \rightarrow \text{O}^+ + \text{e}^-$	$\sigma_8 = 1.745 \times 10^{-15} \Phi_{\text{ph}}(0.1271, 0.07589, x_f, 3.784, 17.64)$	$E_{th} = 13.62$ $x_f(1.240, 8.698)$	a;b
11	$\text{Si} + \gamma \rightarrow \text{Si}^+ + \text{e}^-$	$\sigma_9 = 2.506 \times 10^{-17} \Phi_{\text{ph}}(0.4207, 0.2837, x_f, 20.57, 3.546)$	$E_{th} = 8.152$ $x_f(23.17, x_{Si})$	a;b
12	$\text{Fe} + \gamma \rightarrow \text{Fe}^+ + \text{e}^-$	$\sigma_{10} = 3.062 \times 10^{-19} \Phi_{\text{ph}}(0.2481, 20.69, x_f, 2.671 \times 10^7, 7.923)$	$E_{th} = 7.902$ $x_f(0.05461, 138.2)$	b

Table 7. Heating processes. References: (a) Glover & Jappsen (2007); (b) Verner & Ferland (1996). Details within the text.

Process	Heating	Ref
H_2 photodissociation	$\Gamma = 6.4 \times 10^{-13} R_d n_{\text{H}_2}$	a
UV pumping of H_2	$\Gamma = 2.7 \times 10^{-11} R_d n_{\text{H}_2} \left(\frac{n}{n+n_{\text{cr}}} \right)$	a
H photo-ionization	see text	a
He photo-ionization	see text	a
Gas-phase H_2 formation	$\Gamma = \left(2.93 \times 10^{-12} k_{21} n_{\text{H}^-} + 5.65 \times 10^{-12} k_{11} n_{\text{H}_2^+} \right) \left(\frac{n}{n+n_{\text{cr}}} \right)$	a
Dust-phase H_2 formation	$\Gamma = 7.16 \times 10^{-12} k_{\text{dust}} n n_{\text{H}} \left(\frac{n}{n+n_{\text{cr}}} \right)$	a
Cosmic-ray ionization	$\Gamma = 3.2 \times 10^{-11} \zeta_{\text{tot}} n$	a
Photoelectric effect	see Sect. 2.4.1	b

large uncertainties. The reaction numbers are the same as those used in Tables 2, 3, 4, and 6 for the sake of an easy identification.

$\text{H}_2 + \text{H}^+ \rightarrow \text{H}_2^+ + \text{H}$ (REACTION 9). Each author has his own favored rate for this reaction so that the overall uncertainty is large. Our prescription is as follows. Up to 3×10^4 K we adopt the data of Savin et al. (2004), based on the latest and most accurate quantum-mechanical calculations of the vibrationally resolved cross sections for the charge transfer $\text{H}_2 + \text{H}^+ \rightarrow \text{H}_2^+ + \text{H}$ at the center-of-mass collision energies from the threshold (~ 1.8 eV) up to 10 eV. For higher collision energies, up to approximately 10^4 eV, we derive the cross section following the suggestions by Barnett et al. (1990) and Janev et al. (1987), which

stand on the best known experimental (see for instance Gealy & van Zyl 1987) and theoretical data, as well as on the recent measurements by Kusakabe et al. (2003). These data were smoothly matched to those of Krstić (2002) at low energies, thus yielding the most updated cross section for the charge transfer from H to H_2^+ in the vibrationally ground state. This cross section is shown in the top panel of Fig. 1. Following Savin et al. (2004), from this cross section we calculate the rate coefficients for temperatures from the threshold up to 10^8 K, as shown in the bottom panel of Fig. 1.

We fit the reaction rate (in units of $\text{cm}^3 \text{s}^{-1}$) with the analytical expression

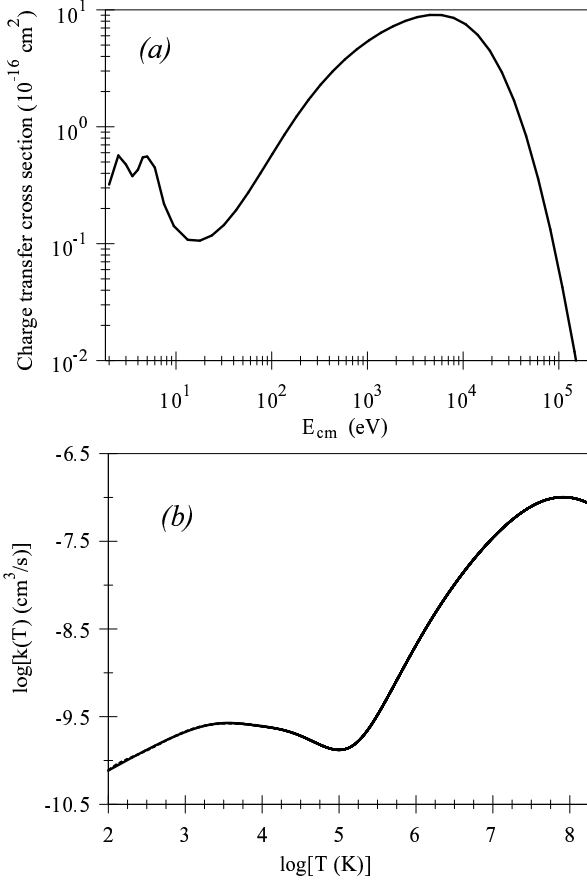
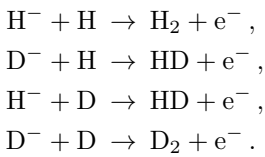


Fig. 1. Top panel: charge transfer cross section for the reaction $\text{H}_2 + \text{H}^+ \rightarrow \text{H}_2^+ + \text{H}$. Bottom panel: Rate coefficient $k(T)$ derived from the cross section in the top panel. The fits of $k(T)$ (dashed line) given by eqn. (9) and Table 8 cannot be visually distinguished from $k(T)$.

$$\log[k_9(T)] = \sum_{i=0}^7 a_i \log(T)^i, \quad (13)$$

where T is the temperature in Kelvin. Two intervals are considered for the temperature, i.e. $T = [10^2, 10^5]$ K and $T = [10^5, 10^8]$ K and two different fits are derived. The coefficients a_i for the two fits are given in Table 8. The rates obtained from this analytical fit are compared to those derived from the numerical data. No difference can be noticed as shown in the bottom panel of Fig. 1.

ASSOCIATIVE DETACHMENTS OF H^- AND D^- (REACTIONS FROM 21 TO 24). These processes include the following four reactions



Following Glover et al. (2006) and using their notation, the reaction rate is expressed as $k = \text{const} \times \xi$ where

Table 8. Fit coefficients for the charge transfer reaction $\text{H}_2 + \text{H}^+ \rightarrow \text{H}_2^+ + \text{H}$ as described in the eqn. (13). Coefficients are in the form $a(b) = a \times 10^b$. All the temperature ranges are shown.

a_i	$10^2 \leq T < 10^5$ K	$10^5 \leq T \leq 10^8$ K
a_0	-1.9153214(+2)	-8.8755774(+3)
a_1	4.0129114(+2)	1.0081246(+4)
a_2	-3.7446991(+2)	-4.8606622(+3)
a_3	1.9078410(+2)	1.2889659(+3)
a_4	-5.7263467(+1)	-2.0319575(+2)
a_5	1.0133210(+1)	1.9057493(+1)
a_6	-9.8012853(-1)	-9.8530668(-1)
a_7	4.0023414(-2)	2.1675387(-2)

ξ is a parameter taking values in the interval $[0.65, 5.0]$. In our case we obtain $k_{21} = k_{24} = 10^{-9} \xi$ and $k_{22} = k_{23} = 10^{-9} \xi/2$. In ROBO the parameter ξ can be varied in the above interval to investigate its effects on the overall results. In the present study we adopt $\xi = 0.65$.

$\text{H}^+ + \text{H}^- \rightarrow \text{H} + \text{H}$ (REACTION 29). For the mutual neutralization of H^- and H^+ we adopt the cross section given by Croft et al. (1999) and Glover & Abel (2008). The mutual neutralization rate is related to the rate of the associative detachment $\text{H}^- + \text{H} \rightarrow \text{H}_2 + \text{e}^-$, in competition for the available H^- ions. Both rates are important for the formation of H_2 (see Glover & Abel 2008, for more details). It is worth recalling here that other estimates for this reaction have been reported by Glover & Abel (2008). Moseley et al. (1970) proposed the rate $k = 5.7 \times 10^{-6} T^{-0.5} + 6.3 \times 10^{-8} - 9.2 \times 10^{-11} T^{0.5} + 4.4 \times 10^{-13} T \text{ cm}^3 \text{ s}^{-1}$, whereas Dalgarno & Lepp (1987) gave $k = 7.0 \times 10^{-7} T^{-0.5} \text{ cm}^3 \text{ s}^{-1}$.

$\text{He}^+ + \text{e}^- \rightarrow \text{He} + \gamma$ (REACTION 35). This process can occur either via direct radiative recombination or di-electronic recombination, followed by radiative relaxation. Therefore, the reaction rate is the sum of two terms

$$k_{35} = k_{35}^d + k_{35}^e, \quad (14)$$

in units of $\text{cm}^3 \text{ s}^{-1}$. The first term is the di-electronic recombination rate, described by

$$k_{35}^d = 1.544 \times 10^{-9} T_e^{-1.5} \exp(-48.596/T_e) \times [0.3 + \exp(8.1/T_e)], \quad (15)$$

where T_e is the temperature expressed in eV. The second term is the radiative recombination rate whose temperature dependence is

$$k_{35}^e = 3.925 \times 10^{-13} T_e^{-0.6353}. \quad (16)$$

Both terms are taken from Abel et al. (1997).

$\text{Z}^+ + \text{e}^- \rightarrow \text{Z} + \gamma$ (REACTIONS 38, 40, 42, 44). An important process to include is the metal recombination. The metals considered by our ISM model and ROBO are C, Si, O, and Fe. The recombination rates of the metals are calculated according to the formalism proposed by Verner & Ferland (1996)

$$\Phi_{\text{rec}}(T, A, \tau_0, \tau_1, b) = \frac{A}{T_0 (1 + T_0)^{1-b} (1 + T_1)^{1+b}}, \quad (17)$$

in units of cm^3s^{-1} , where T is the temperature, $T_0 = \sqrt{T/\tau_0}$ and $T_1 = \sqrt{T/\tau_1}$. In Table 4 we give the fit coefficients A , b , τ_0 and τ_1 for each metal.

$Z + e^- \rightarrow Z^+ + 2e^-$ (REACTIONS 39, 41, 43, 45). For the collisions between electrons and metals we use the fit proposed by Voronov (1997) expressed by

$$\Phi_{\text{col}}(T, A, P, \Delta E, X, K) = A \frac{1 + P\sqrt{U}}{X + U} U^K e^{-U}, \quad (18)$$

in units of cm^3s^{-1} , where $U = \Delta E/T$ in which ΔE is the energy difference between the two atomic levels involved in the process, and T is the temperature. Both ΔE and T are in eV. The parameters A , P , ΔE , X , and K are given in Table 4.

2.3. The dust

Dust grains take part to the process of molecule formation, e.g. HD and H_2 form on the surface of dust grains; therefore, all the physical processes involving the dust must be described and treated with the highest accuracy. To understand how grains take part in the process of molecule formation, we need to know the mechanisms of grain formation and destruction, along with the distribution of the grain temperature and size.

Given an initial set of dust composition and abundances, our dust model follows the evolution of the dusty components during the history of the ISM due to the creation of new grains and the destruction of the existing ones. Creation of new dust grains is governed by several processes that have different efficiencies depending on the size of the dust particles. The same applies for the destruction that is mainly due to the thermal motion of the gas particles and shocks from supernovae. Thermal destruction is quite easy to model, because the only parameter at work is the gas temperature. Shock disruption is more difficult to evaluate. The main uncertainty comes from discrete numerical hydrodynamical simulations only being able to follow shocks up to a given (often too coarse) resolution, yet insufficient for the microscopic description required here. To cope with this, we have followed a “statistical” approach. In brief, once identified the gas particles of the NB-TSPH simulations with their turbulent velocities (suggested by their velocity dispersion), we assume that all the shocks inside them follow the Kolmogorov law. See below for more details.

Furthermore, grain destruction may depend on their size. This is the case of the destruction by thermal motions, where the small grains are destroyed before the large ones. As a consequence of this, the size distribution function and the abundances of dust grains of different type are continuously changing with time. The ever-changing

size distribution function plays an important role in the formation mechanism of HD and H_2 , which are among the most efficient coolants. In brief, changing the distribution function of the dust grains means changing the quantity of key-role molecules produced by dust, as shown in Cazaux & Spaans (2009).

We analyze here the different aspects of the grain evolution and their role in the formation of coolant molecules. First, we focus on the distribution function of the dust grains, then we describe the formation of coolants on grains. Finally, we analyze the destruction and the formation of dust, as well as the effects of the grain temperature.

2.3.1. Size distribution function of dust grains

We adopt a simple power-law, MRN-like, grain size distribution function (Mathis et al. 1977; Draine & Lee 1984). This is given by $dn(a)/da \propto a^{-\lambda}$, with a the grain dimension, $n(a)$ the corresponding number density of grains with dimension a , and $\lambda = 3.5$; this distribution covers the range $5 \text{ \AA} < a < 2500 \text{ \AA}$ and is extended to the smallest grain dimensions in such a way as to include the typical polycyclic aromatic hydrocarbons (PAHs) region (Li & Draine 2001b). Even if more complicated distributions have been proposed (Weingartner & Draine 2001a), our simple choice is suitable as an initial condition for the purposes of this work.

Indeed, it is worth pointing out here that the initial size distribution function changes with time owing to dust destruction and formation, which in turn depend on the grain size. Consequently, the power law we started with does no longer apply. This will soon affect the formation of H_2 or HD. For this reason, the knowledge of the current size distribution function is paramount.

First of all, we split the interstellar dust in two main components: the carbon-based (thereinafter simply carbonaceous grains) and the silicon-based grains (thereinafter the silicates). These play different roles in the formation of molecular hydrogen, as well as in different formation and destruction rates. In principle, there should be an additional group to consider, namely the PAHs, because the three types of dust have different efficiencies in the H_2 and HD formation. However, there is the lucky circumstance that PAHs and graphite grains have similar efficiencies to those shown by Fig. 2 of Cazaux & Spaans (2009). Basing on this, we lump together graphite grains and PAHs and treat separately the silicates. Therefore, thereinafter we refer to the mixture graphite grains plus PAHs as “carbonaceous grains”.

2.3.2. Dust-driven H_2 and HD formation

The model we adopt to describe the so-called dust phase is the one proposed by Cazaux & Spaans (2009). In brief, the dust phase has its own network of reactions that establish the relative abundances of the various types of dust and govern the formation of H_2 and HD. We can neglect

the presence of other molecules like CO. The newly formed H_2 and HD are fed to the gas phase and enter the system of equations governing the abundances of the ISM and vice versa. In reality, the two phases should be treated simultaneously, governed by a unique system of differential equations determining the number densities of the elemental species, dusty compounds, and free electrons at once. In practice this is hard to do because in general the time scales of the various processes creating and destroying the elemental species of the gas phase are much different from those governing the formation/destruction of dust grains. To cope with this difficulty, the Cazaux & Spaans (2009) model separates the two phases, provides the results for the dust phase, and more important provides a link between the two phases that allows determining the formation rate of H_2 and HD by dust as a function of the gas temperature T_g and dust temperature T_d . This link is named the “Sticking Function $S(T_g, T_d)$ ”. The sticking function somehow quantifies the probability that an hydrogen atom striking a grain sticks to the surface rather than simply bouncing off. The sticking function characterizes the probability for an atom to remain attached to a grain. Here we strictly follow this way of proceeding.

According to the Cazaux & Spaans (2009) model, the rate at which H_2 molecule forms over the grain surface is

$$R_d(\text{H}_2) = \frac{1}{2} n(\text{H}) v_H n_d \sigma \epsilon_{\text{H}_2} S(T_g, T_d) \text{ cm}^{-3} \text{ s}^{-1}. \quad (19)$$

where $n(\text{H})$ is the density of hydrogen in the gas phase, $v_H = \sqrt{8kT_g/(\pi m_{\text{H}})}$ is the gas thermal speed (k is the Boltzmann constant and m_{H} is the hydrogen mass), n_d the dust number density, σ the grain cross section (i.e. πa^2), ϵ_{H_2} the intrinsic efficiency of the process, and $S(T_g, T_d)$ the sticking function. This latter is in turn a function of the dust and grain temperatures

$$S(T_g, T_d) = \left[1 + 0.4 \left(\frac{T_g + T_d}{100} \right)^{0.5} + 0.2 \left(\frac{T_g}{100} \right) + 0.08 \left(\frac{T_g}{100} \right)^2 \right]^{-1}, \quad (20)$$

where T_g and T_d are in Kelvin. With this function the probability for a gas molecule to remain stuck on a dust grain is higher in a cold gas than in a hot one. This probability is even higher for the cold grains. For HD the expressions are similar.

Equation (19) contains the intrinsic efficiency ϵ_{H_2} (or as ϵ_{HD} for HD), which is the probability for the process to occur. In our case it is the probability that atoms, which are stuck to a grain, react to form H_2 or HD.

For the carbonaceous grains, the efficiencies ϵ_{H_2} and ϵ_{HD} coincide and are equal to

$$\epsilon_{\text{C}} = \frac{1 - T_{\text{H}}}{1 + 0.25 \left(1 + \sqrt{\frac{E_{\text{ch}} - E_{\text{S}}}{E_{\text{phy}} - E_{\text{S}}}} \right)^2 e^{-\frac{E_{\text{S}}}{T_d}}}. \quad (21)$$

Table 9. The values used to compute the formation efficiency of H_2 and HD, with E_{phy} , E_{ch} , and E_{S} in K, a_{pc} is in Å. From Cazaux & Spaans (2009).

Surface	E_{phy}	E_{ch}	E_{S}	a_{pc}
Carbons	800 K	7000 K	200 K	3 Å
Silicates	700 K	15000 K	−1000 K	1.7 Å

The efficiency for the silicates ϵ_{Si} is

$$\epsilon_{\text{Si}} = \frac{1}{1 + \frac{16T_d}{E_{\text{ch}} - E_{\text{S}}} e^{-\frac{E_{\text{ph}}}{T_d}} e^{\beta_d a_{\text{pc}} \sqrt{E_{\text{phy}} - E_{\text{S}}}}} + \mathcal{F}, \quad (22)$$

where $\beta_d = 4 \times 10^9$ for H_2 and $\beta_d = 5.6 \times 10^9$ for HD. The term $\mathcal{F}$ is a function of the gas temperature and can be written as

$$\mathcal{F}(T) = 2 \frac{e^{-\frac{E_{\text{phy}} - E_{\text{S}}}{E_{\text{phy}} + T_g}}}{\left(1 + \sqrt{\frac{E_{\text{ch}} - E_{\text{S}}}{E_{\text{phy}} - E_{\text{S}}}} \right)^2}, \quad (23)$$

where T_{H} is given by the expression

$$T_{\text{H}} = 4 \left(1 + \sqrt{\frac{E_{\text{ch}} - E_{\text{S}}}{E_{\text{phy}} - E_{\text{S}}}} \right)^{-2} \exp \left(-\frac{E_{\text{ch}} - E_{\text{S}}}{E_{\text{ch}} + T_g} \right). \quad (24)$$

The various quantities appearing in the above relationships are listed in Table 9. For more details on these equations see Cazaux & Spaans (2009). Comparing eqns. (21) and (22), we see that the efficiency is high when the dust temperature is low. For the silicates the efficiency window is shorter than for the carbonaceous grains. For the silicates the efficiency is high for temperatures up to 20 K and then falls by two orders of magnitude. The carbonaceous grains are efficient for temperatures up to 100 K, where the efficiency is still 0.1 (instead of 0.01 as for the silicates). Finally, it is worth noticing that the efficiency profile is smoother for the carbonaceous grains.

From all these considerations it follows that cold dust and warm carbon-dominated dust in a cold gaseous environment are the most efficient drivers for the formation of coolant molecules like H_2 and HD.

2.3.3. Grain formation

Here we briefly examine the formation of dust grains. We start with an initial number density with the size distribution given in Section 2.3.1 and with a given ratio between the silicates and the carbonaceous grains. The latter is a free parameter varying in the range $[0, 1]$, where zero stands for dust made of sole carbonaceous grains; one is for dust made of sole silicates.

According to Dwek (1998), the temporal variation in the size distribution function of grains caused by accretion is given by

$$\frac{dn(a)}{dt} = c_d \alpha(T_g, T_d) \pi a^2 n_g n_d(a) v_g \quad (25)$$

where all the quantities have the same meaning as in eqn. (19), but for $\alpha(T_g, T_d)$ and c_d .

The quantity $\alpha(T_g, T_d)$ is a sort of sticking coefficient depending on the gas and dust temperature and the type of dust. This coefficient is therefore forced to change in the course of the evolution. Furthermore, n_d (and consequently n) are functions of a . Equation (25) is similar to eqn. (19) because the process is similar, except that now this mathematical description is applied to the carbon atoms that remain stuck to the carbon lattice of the grain. Our expression differs slightly from the original one of Dwek (1998) because the parameter c_d is introduced in eqn. (25) to take some considerations made by Dwek (1998) himself into account. Inserting $c_d = 1$ we can obtain the time scale of the process. In a standard cold gas this time scale is $\tau \approx 2 \times 10^4$ yrs, which is significantly smaller than the observational estimates. Normal evaporation, caused by cosmic rays or UV heating and grain-grain collisions can halt the growing of the dust grains. The factor c_d somehow takes this phenomena into account. It is estimated to be on the order of $c_d = 10^{-3}$. We name c_d the delay factor.

For the sticking coefficient $\alpha(T_g, T_d)$, we make use of the data by Leitch-Devlin & Williams (1985) and consider a carbon atom of the gas phase as impinging on a carbon lattice. The equation fitting the data has the form

$$\alpha(T_g, T_d) = 0.0190 T_g (0.0017 T_d + 0.4000) \times \exp(-0.0070 T_g), \quad (26)$$

where T_g and T_d are the gas and dust temperatures, respectively (see Fig. 2). The sticking coefficient $\alpha(T_g, T_d)$ has no dimensions. The data of Leitch-Devlin & Williams (1985) provide a good dependence on the gas temperature T_g , but a poor one on the dust temperature T_d . Therefore an accurate fit is not possible. The above relationship can be used in the temperature intervals $10 \text{ K} \leq T_g \leq 1000 \text{ K}$ and $3 \text{ K} \leq T_d \leq 300 \text{ K}$. At present we also use the same model for silicate grains even though this approximation might not be accurate. We plan to improve upon this point in the future.

With this model the formation efficiency is higher when the interstellar medium has a temperature $T \approx 100 \text{ K}$ and when the dust grains have a temperature of about 300 K . For dust grains with temperature higher than 300 K or lower than 3 K , the formation efficiency is unknown. Though the fits from Leitch-Devlin & Williams (1985) are not very accurate, they are accurate enough for our purposes because we are only interested in the shape and maxima of the curves in Fig. 2.

Finally, we note that we have already described the formation of dust by means of a general process of accretion in the ISM, leaving aside other sites of dust formation like the envelopes of obscured AGB stars, Wolf-Rayet

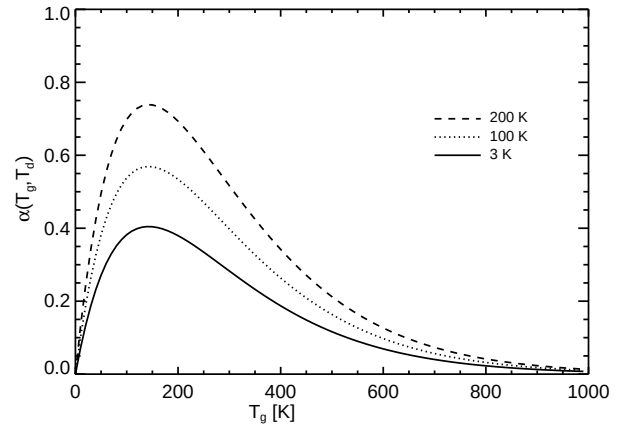


Fig. 2. The sticking coefficient for the carbon molecules over a carbon lattice derived from Leitch-Devlin & Williams (1985). The different lines indicate a different dust temperature: 3 K (solid), 100 K (dotted), and 200 K (dashed). T_g is the gas temperature in Kelvin.

stars, and remnants of supernovae (Dwek 1998). The reason behind this is that all of these are external sources of dust that eventually enrich the ISM in dust content and therefore determine a different initial dust content as input for ROBO. Indeed, for our purposes, only the internal sources of dust in the unit volume are important. The information about the initial conditions of the dust mixture should be provided to ROBO by the companion NB-TSPH code EvOL, of properly taking the dusty yields coming from the stars into account.

2.3.4. Grain destruction

In our model we assume that there are three processes destroying the grains of the ISM, i.e. shocks, in particular supernovae induced shocks, vaporization by high-velocity shocks, and thermal sputtering, in which dust is destroyed by the thermal motions of the gas. We need to describe the three processes in a way suited to the NB-TSPH formalism.

Destruction by shocks. This phenomenon is difficult to model. First of all shocks may easily induce turbulence in the ISM and to be suitably described one needs some assumptions about the velocity fields of the fluid elements. The main difficulty arises from the fact that within an SPH gas particle, because of the typical mass resolution, unresolved shocks may take place at different velocities. To avoid this difficulty, we treat the gas as a turbulent fluid, assuming that even inside a gas particle the turbulent nature of the fluid is preserved, and use the Gaussian velocity distribution $\phi(v)$ derived from the Kolmogorov law for the power spectrum $E(k) \propto k^{-\alpha} dk$ with $\alpha=5/3$. Second, grains of different masses and sizes move at different velocities: the large, more massive grains moving slower than the smaller less massive ones. This

suggests adopting the point of view in which the small fast grains are considered as projectiles impinging on the massive slow grains considered as targets. To summarize, the ISM we are dealing with is turbulent because of the shocks crossing it and rich in dust grains of all possible dimensions and masses interacting with each other and with the shock fronts.

Given these premises, we model the destruction of dust grains by shocks according to the picture by Hirashita & Yan (2009). The dust grains are grouped in $N = 100$ bins of number density according to their mass. The mass of a grain is in turn expressed as the product of a reference density, here assumed to be the density of the graphite grains $\rho_{\text{gr}} = 2.3 \text{ g/cm}^3$ times the volume of the grain $V_{\text{gr}} = 4/3\pi a^3$ where a is the radius of the grain (assumed to be spherical for simplicity). Therefore each density bin contains grains whose mass goes from m_{low} to m_{up} , where both mass limits vary with the bin. The corresponding radii are given by $m_{\text{low}} = 4/3\pi a_{\text{low}}^3$ and $m_{\text{up}} = 4/3\pi a_{\text{up}}^3$. The grain radii follow the distribution law given by $dn(a) = C_{\text{nor}} \times a^{-3.5} da$, where C_{nor} is a suitable normalization factor to be determined. Therefore, the number density of the grains in each mass interval is

$$n_i = \int_{a_{\text{low}}}^{a_{\text{up}}} n(a) da = C_{\text{nor}} \int_{a_{\text{low}}}^{a_{\text{up}}} a^{-3.5} da \quad (27)$$

The total number density of dust grains of any size and hence mass is

$$n_{\text{dust}} = \sum_{i=0}^{N-1} n_i. \quad (28)$$

When a projectile of mass m_j hits a target of mass m_i , the target loses a fraction of mass $f_{\text{sh}} m_i$ if $m_j < m_i/2$, or it is totally destroyed if $m_j \geq m_i/2$ (i.e. $f_{\text{sh}} = 1$). In both cases the projectile is destroyed. The lost mass, i.e. a fragment of lower mass, so the dimension is assigned to the mass (size) bin according to the adopted power law. The remaining part of the target (remnant) of mass $(1 - f_{\text{sh}})m_i$ is added to the appropriate mass, hence size bin, so the number density variation of the i -th bin is

$$\frac{dn_i}{dt} = \left(\frac{dn_i}{dt}\right)_{\text{L}} + \left(\frac{dn_i}{dt}\right)_{\text{F}} + \left(\frac{dn_i}{dt}\right)_{\text{R}}, \quad (29)$$

where the subscript L indicates the term representing the mass lost in fragments, F the mass gained from the fragmentation, and finally R the mass of the remnants moving from other bins to the i -th one. The first term must be negative.

To estimate the various contributions to the total number density variation, we must consider the probability of an impact. This will be proportional to the number density of the targets n_i , the density of the projectiles n_j , their relative speed v_{ij} , and their sizes (a_i and a_j). The assumptions on the speed will be discussed later in this section. For the targets belonging to the i -th bin, we find

that the mass lost in fragments corresponds to the density change of

$$\left(\frac{dn_i}{dt}\right)_{\text{L}} = - \sum_{j \leq i} \epsilon_{ij} \alpha_{ij} n_i n_j, \quad (30)$$

where

$$\epsilon_{ij} = \begin{cases} \frac{f_{\text{sh}} n_i + n_j}{n_i + n_j} & \text{if } m_j \leq m_i/2, \\ 1 & \text{if } m_j > m_i/2 \end{cases} \quad (31)$$

and the impact coefficient α_{ij} is

$$\alpha_{ij} = v_{ij} \pi (a_i + a_j)^2. \quad (32)$$

Consequently, the total density variation of newly created fragments is

$$\frac{dn_{\text{frag}}}{dt} = \sum_i \frac{dn_i}{dt}. \quad (33)$$

They will be distributed among the bins with a power law to obtain the $(dn_i/dt)_{\text{F}}$ of each bin.

The variation in the remnant number density is given by

$$\left(\frac{dn_k}{dt}\right)_{\text{R}} = (1 - \epsilon_{ij}) \frac{dn_i}{dt}, \quad (34)$$

where k is the index of the bin receiving the remnant given by

$$k = \text{int} \left[\frac{m_i(1 - f_{\text{sh}}) - m_{\text{low}}}{\Delta m} \right], \quad (35)$$

with i the index of the initial grain that suffered fragmentation and Δm the mass range of each bin, $(m_{\text{up}} - m_{\text{low}})/N$.

The shock velocities obey a Gaussian distribution $\phi(v)$ normalized to

$$\int_{v_{\text{low}}}^{v_{\text{up}}} \phi(v) dv = 1, \quad (36)$$

where v_{low} and v_{up} are the limits of the shock velocities over which the normalization of $\phi(v)$ applies, thus fixing the normalization constant. The relative speed v_{ij} of the grains is obtained from the velocity distribution of eqn. (36) with $v_{\text{low}} = 1 \text{ km/s}$ and $v_{\text{up}} = 200 \text{ km/s}$ and a total of 30 velocity bins. The velocity $\phi(v)$ is a Gaussian centered at $v = 100 \text{ km/s}$ with a dispersion of 30 km/s . The integral

$$\xi(v) = \int_{v_i}^{v_{i+1}} \phi(v) dv, \quad (37)$$

yields the relative weight of each velocity bin. The relative velocities of the projectiles are distributed according to these weights and the total density of targets in the i -th bin changes as described by eqns. (30) and (32).

Vaporization. Since we also deal with high-velocity shocks, vaporization of dust grains into the gas phase may become important. We use the formalism of Tielens et al. (1994a) for the impact of carbon grains. The fraction of vaporized material is

$$F_v = \frac{f_{\text{vap}}}{1 + f_{\text{vap}}} \quad (38)$$

with f_{vap} given by

$$f_{\text{vap}} = \left[f_{v1} + f_{v2} \sqrt{1 - \frac{v_t}{v}} \right] \frac{m_j}{m_i}, \quad (39)$$

where $v_t = 23$ km/s is the shock threshold velocity, m_i and m_j are the masses of target and projectile grains, respectively. Equation (39) must satisfy the condition $v \geq v_t$. The quantities f_{v1} and f_{v2} are taken from Tielens et al. (1994a) and are given by the relations

$$f_{v1} = 2.59 \frac{a_i}{a_i + a_j}, \quad (40)$$

and

$$f_{v2} = \frac{2.11}{\sigma_1^{8/9}}, \quad (41)$$

where

$$\sigma_1 = \frac{0.3 (s + \mathcal{M}^{-1} - 0.11)^{1.3}}{(s + \mathcal{M}^{-1} - 1)}, \quad (42)$$

with $\mathcal{M}$ the Mach's number and $s = 1.9$.

Destruction by thermal sputtering. To evaluate the fraction of grains destroyed by thermal sputtering, we adopt the approximation by Draine & Salpeter (1979). A grain of dust in a medium with temperature $T \leq 10^6$ K has a destruction time (i.e. its lifetime)

$$\tau_{\text{dist}} \approx 10^3 a / n_{\text{H}} \text{ yr.}$$

with a in nm. From this we can obtain the destruction rate per second. The above relation is only valid for high temperatures.

To include the temperature dependence of the destruction rate, we refer to Tielens et al. (1994b), with the aid of which we model the dependence on the gas temperature of the lifetime of the dust. As expected, dust grains with lower temperature have a longer lifetime. Finally, according to Draine & Salpeter (1979), the small grains have a shorter lifetime than the large ones in the same environment.

The grain temperature depends on the radiation field generated by all the stellar sources. However, for the sake of simplicity, in this study we used a fixed value for the grain temperature. Therefore, a big improvement would be given by including a description of the photon diffusion (e.g. Mathis et al. 1983). Our choice is partially justified by the fact that the companion NB-TSPH code EVoL does not yet include photon diffusion. This leads us to postpone the implementation of different grain temperatures to when photon diffusion is included in EVoL.

2.4. Heating and cooling

At this stage, it is worth discussing in some detail the role played by the heating and cooling during the history of star formation in a galaxy or a cosmological simulation. We have already emphasized that coolants are the key elements for the formation scenario. As the gas cools down, more and more spatial structures and stars (with

a certain mass function, hence mass-luminosity law) are born. Without coolants neither structures nor stars would form. In this context, the grains and their temperature in turn play the key role in the formation of efficient coolant molecules like H_2 and HD. Cold grains form more molecules than the warm ones. Since the dust temperature depends on the surrounding photon flux, it means that stars in the neighborhood are crucial for heating the dust particles. All this forms a closed loop of interwoven physical processes: dust forms coolants - coolants induce star formation - stars heat the dust. Understanding the details of this mutual interaction will allow us to get clues on the star formation history in general.

In this scene a starring actor is the gas cooling. Chemical reactions are sensitive to the gas temperature, hence to the gas cooling; indeed, the formation of coolants depends on the gas temperature, which in turn depends on the cooling process. We split the cooling of the gas in several sources characterized by the physics of the dominant process. Above 10^4 K, there are two very popular descriptions of the cooling, namely Cen (1992) and Sutherland & Dopita (1993). At lower temperatures we have the *metal* cooling (Maio et al. 2007), the *molecular hydrogen* cooling (Glover & Abel 2008; Galli & Palla 1998), and finally, the HD cooling (Lipovka et al. 2005).

2.4.1. Heating by photoelectric ejection of electrons from dust grains

The photoelectric ejection of electrons by dust grains is an important source of heating. The model proposed here is based on Bakes & Tielens (1994) and Weingartner & Draine (2001b). The photoelectric heating is given by

$$H(N_C, Z) = W \pi \int_{\nu_Z}^{\nu_H} \sigma_{\text{abs}}(N_C) Y_{\text{ion}}(N_C, IP_Z) \times F(\nu) g(N_C, IP_Z) d\nu, \quad (43)$$

where W is the FUV dilution factor, σ_{abs} the photon absorption cross section, Y_{ion} the photoelectric ionization yield, $F(\nu)$ the UV radiation flux, and $g(N_C, IP_Z)$ the kinetic energy partition function. Here, N_C is the number of carbon atoms that form the dust ($N_C \sim 0.5 a^3$ with a the radius of the grain in Å, according to Li & Draine (2001b)) and IP_Z the ionization potential of a grain of charge Z , and ν_H is the Lyman frequency and ν_Z the frequency corresponding to IP_Z . In our model the range of grain sizes goes from 5 Å to 100 Å, since the total heating is mostly due to small grains (see Bakes & Tielens 1994, for details).

The total photoelectric heating is

$$\Gamma = \int_{N_-}^{N^+} \sum_Z H(N_C, Z) f(N_C, Z) n(N_C) dN_C, \quad (44)$$

where $f(N_C, Z)$ is the probability to find a grain composed of N_C carbon atoms at a certain charge Z . This can be

computed by considering the collisions with electrons and ions. The detailed balance yields

$$f(Z) [J_{\text{pe}}(Z) + J_{\text{ion}}(Z)] = f(Z+1)J_e(Z+1), \quad (45)$$

with J_{pe} the rate of photoelectron emission, J_{ion} , and J_e the accretion rates of ions and electrons. For the detailed calculation of $f(Z)$ see Bakes & Tielens (1994), while for all the details about J_{pe} , J_{ion} and J_e see Weingartner & Draine (2001b).

As the FUV absorption cross section σ_{abs} depends on the absorption efficiency of FUV photons by the grains, we use the data of Draine & Lee (1984) and Laor & Draine (1993) for the graphite grains and Li & Draine (2001a,b,c) for the PAHs. We also use the model proposed by Li & Draine (2001a,b,c) to create a mixed population of PAHs and graphite grains. Following their paper the total absorption efficiency is

$$Q_{\text{abs}}(a, \lambda) = \xi(a)Q_{\text{abs}}^{\text{PAH}}(a, \lambda) + [1 - \xi(a)]Q_{\text{abs}}^{\text{gra}}(a, \lambda), \quad (46)$$

defining

$$\xi(a) = (1 - q) \min \left[1, \left(\frac{a_0}{a} \right)^3 \right] \quad (47)$$

by $q = 0.01$ and $a_0 = 50 \text{ \AA}$ (see Li & Draine 2001a,b,c, for more details).

In the same way we can also obtain the cooling associated with the electron recombination. This is given by

$$C(N_C, Z) = n_i s_i \sqrt{\frac{8kT}{\pi m_i}} \pi a^2 \hat{\Lambda}(\tau, \nu), \quad (48)$$

where n_i is the density of the charged particle with sticking coefficient $s_i = 1$ and mass m_i and a is the grain size. The term $\hat{\Lambda}(\tau, \nu)$ is described in detail in Bakes & Tielens (1994). The total cooling is obtained as in eqn. (44). In Fig. 3 we show the results from a simple model with a grain distribution with $n_{\text{dust}} = 10^{-5} \text{ cm}^{-3}$ and a gas with a temperature of 10^2 K .

2.4.2. Which are the sources of cooling to consider?

Cooling at high temperatures ($T \geq 10^4$). Two main sources of cooling can be found in the literature: (i) The formulation proposed by Cen (1992) which includes the following mechanisms:

- collisional ionization - H, He, He^+ , $\text{He}(2^3\text{S})$,
- recombination - H^+ , He^+ , He^{++} ,
- dielectronic recombination - He,
- collisional excitation - H(all n), $\text{He}^+(n=2)$, $\text{He}(n=2, 3, 4)$,
- bremsstrahlung - all ions,

which is particularly suited in presence of non equilibrium reactions for the H and He chemistry. (ii) The description of Sutherland & Dopita (1993, thereafter SD93), which is particularly suited to treat cooling in presence of metals and to describe it as function of the gas metallicity. Their case for $Z = 0$ can be left aside. Both cooling models

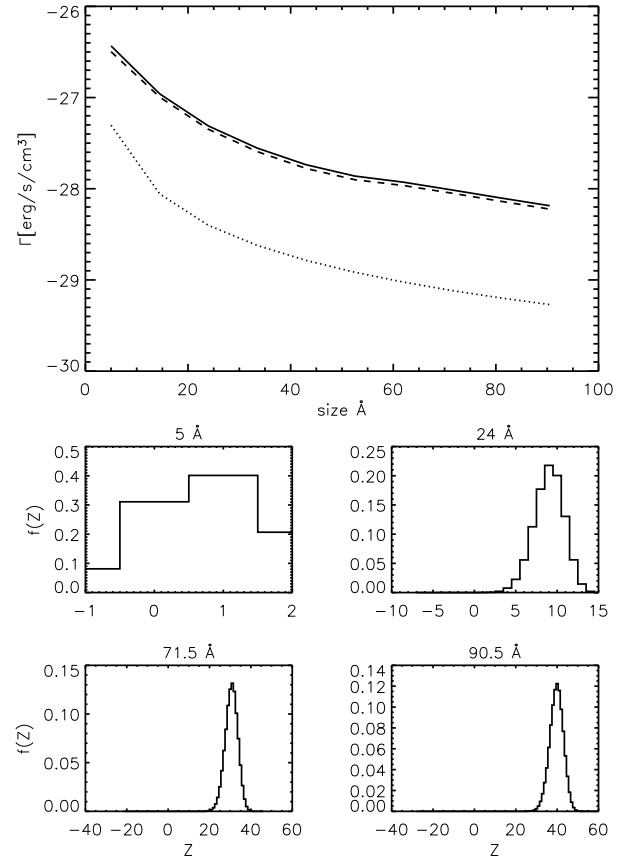


Fig. 3. Top panel: heating by dust (solid line), cooling (dotted line) and the difference between heating and cooling (dashed line) for different diameters of the dust particles. Bottom panel: the probability of finding a grain with a certain charge Z as a function of the grain dimensions.

are used for temperature higher than 10^4 K . We refer to them by the acronyms C92 or SD93. In our models of the ISM we adopt both sources as appropriated to the current physical conditions.

A difference between C92 and SD93 is that the cooling rate is described in the former by numerical fits (no further interpolations are needed), whereas in the latter one has to interpolate huge tabulations of data in temperature and metallicity. To this aim we used a surface fit (routine SFIT in IDL) where the plane surface passes through four points that correspond to two discrete values in temperature and two discrete values in metallicity. The analytical expression of this plane yields the cooling rate. Given the two pairs of interpolating points, (T_0, Z_0) , (T_0, Z_1) , (T_1, Z_0) , and (T_1, Z_1) , the analytical function that describes the surface can be written in the form

$$\Lambda_{\text{SD93}}(T, Z) = a_0 + a_1 Z + a_2 T + a_3 T Z, \quad (49)$$

with a_i the fit coefficients. So, for a generic point (T, Z) , for which $T_0 \leq T \leq T_1$ and $Z_0 \leq Z \leq Z_1$, the rate cooling is given by eqn. (49).

H₂ cooling. The H₂ cooling requires a different description. Hollenbach & McKee (1979) first evaluated the molecular hydrogen cooling. A modern, widely used source of H₂ cooling is by Galli & Palla (1998). Although this cooling function is quite accurate, we prefer to add some more details by including the function found by Glover & Abel (2008), which takes not only the H₂ – H interaction into account, but also the collisions with He, H⁺, H₂, and free electrons. It is described by

$$\Lambda_{\text{H}_2} = \sum_k \sum_i \text{dex} [a_{ik} \log(T_3)^i] n_{\text{H}_2} n_k \text{ erg cm}^{-3} \text{ s}^{-1}, \quad (50)$$

where $T_3 = T/10^3$ K with T the gas temperature, a_{ik} is the i -th fit coefficient of the k -th species ($k = \{\text{H}, \text{H}^+, \text{H}_2, \text{e}^-, \text{He}\}$), and n are the number densities. The orto-para ratio is assumed to be 3 : 1. Outside the temperature range of the fits we use the molecular hydrogen cooling by Galli & Palla (1998).

HD cooling. To describe the cooling by the deuterated molecular hydrogen we use the model proposed by Lipovka et al. (2005). It includes HD roto-vibrational structures, radiative and collisional transitions for $J \leq 8$ rotational levels, and the vibrational levels $v = 0, 1, 2, 3$. It has been found that including the roto-vibrational transitions increases the cooling efficiency of the HD. The fit provided by the authors depends on the gas density and temperature. It can be parameterized as

$$\Lambda(T, n)_{\text{HD}} = \sum_{i=0}^4 \sum_{j=0}^4 c_{ij} \log_{10}(n)^i \log_{10}(T)^j, \quad (51)$$

where c_{ij} is the matrix whose elements are given in Table 10. Using eqn. (51) is particularly convenient from a numerical point of view as it provides fast evaluations of the cooling by the HD molecule.

Cooling by the CO molecule. Owing to founding hypotheses of our model of the ISM presented in the introduction, the ISM is optically thin so that it is worth considering the straight cooling of the CO molecules by radiative transitions among rotational and vibrational energy levels.

To this aim, we may adopt the steady-state equations of McKee et al. (1982) and the rates calculated by Schinke et al. (1985) by applying the $L \rightarrow 0$ coefficients to the Goldflam et al. (1977) method and including the Depristo et al. (1979) correction (see these studies for more details).

To calculate the total rotational cooling, we use the equation of statistical equilibrium from McKee et al. (1982):

$$n_j [A_j + n_{\text{H}_2} \sum_i \gamma_{ji}] - \left[n_{\text{H}_2} \sum_i \gamma_{ij} n_i + n_{j+1} A_{j+1} \right] = 0, \quad (52)$$

where n_j is the number density of the CO molecules in the state $j = J$, n_{H_2} the number density of the molecular

hydrogen, γ_{ij} the rate coefficient of the transition, and A_j the A -coefficient from the state j to $j - 1$. The system of equations is completed by the condition $\sum_j n_j = n_{\text{CO}}$. The solution of this linear system is straightforward. The emission due to the transition from J to $J - 1$ is

$$I_J = \frac{h\nu}{4\pi} \frac{A_J n_J}{n_{\text{CO}}} \text{ erg/s/molecule/sr}, \quad (53)$$

from which we get the total cooling rate $\Lambda_{\text{rot}} = n_{\text{CO}} \sum_J I_J$.

The vibrational component is taken from Hollenbach & McKee (1979), considering collisions with H₂ whose rates, are

$$\begin{aligned} \gamma_{01} &= 4.3 \times 10^{-14} T \exp(-68/T_3) \exp(-E_{10}/kT) \\ \gamma_{02} &= 2.7 \times 10^{-13} T \exp(-172/T_3) \exp(-E_{20}/kT), \end{aligned} \quad (54)$$

where $E_{10}/k = 0.5E_{20}/k = 3080$ K, $T_3 = T^{1/3}$, and k is the Boltzmann's constant.

Cooling by the metals. The cooling process by the metals is included using the list of the metals adopted by Maio et al. (2007), namely CII, SiII, OI and FeII (see the Appendix B in Maio et al. 2007). This source of cooling is particularly important for temperatures lower than 10^2 K. The cooling is due to the fine structure level transitions of the ionized carbon $2\text{P}(3/2) \rightarrow 2\text{P}(1/2)$, and similarly for the ionized silicates. There are two other species that take part in this process: neutral oxygen (nine transitions between levels $^1\text{S}_0$, $^1\text{D}_2$, $^3\text{P}_0$, $^3\text{P}_1$, and $^3\text{P}_2$) and ionized iron (five transitions between levels $^6\text{D}_i$ with i odd in the range $[1, 9] \in \mathbb{N}$). The cooling from each transition is additive and is given in $\text{erg/cm}^3/\text{s}$ by

$$\Lambda_i = \frac{\gamma_{ji}^{\text{H}} + \gamma_{ji}^{\text{e}} n_{\text{e}}/n_{\text{H}}}{\gamma_{ji}^{\text{H}} + \gamma_{ij}^{\text{H}} + (\gamma_{ji}^{\text{e}} + \gamma_{ij}^{\text{e}}) n_{\text{e}}/n_{\text{H}} + A_{ij}/n_{\text{H}}} n_{\text{tot}} A_{ij} \Delta E_{ij} \quad (55)$$

where γ_{ij}^{H} is the reaction rate for the hydrogen, and γ_{ij}^{e} is the same but for the electrons, A_{ij} is the Einstein's coefficient between the i -th and j -th levels, ΔE_{ij} is the energy difference between the levels, n_{tot} is the total number density of the species considered, and finally, n_{H} and n_{e} are the hydrogen and electron number density, respectively. To complete the equation we need to know

$$\gamma_{ji} = \frac{g_i}{g_j} \gamma_{ij} \exp[-\Delta E_{ij}/(k_B T_{\text{gas}})], \quad (56)$$

both for hydrogen and electrons; g_i and g_j are the level statistical weights, k_B is the Boltzmann's constant, and T_{gas} is the gas temperature.

If the model is calculated at high redshift, the CMB pumping must be considered. This process is described by including the stimulated emission coefficient in the treatment of the cooling. The black-body radiation field of the CMB has a temperature that depends on the redshift as $T_{\text{CMB}}(z) = (1+z)T_{\text{CMB}}(0)$ K. However, as the models of the ISM refer to the current age, the temperature is kept

Table 10. Coefficients c_{ij} used in the eqn. (51) and taken from Lipovka et al. (2005).

	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$j = 0$	-42.56788	0.92433	0.54962	-0.07676	0.00275
$j = 1$	21.93385	0.77952	-1.06447	0.11864	-0.00366
$j = 2$	-10.19097	-0.54263	0.62343	-0.07366	0.002514
$j = 3$	2.19906	0.11711	-0.13768	0.01759	-0.00066631
$j = 4$	-0.17334	-0.00835	0.0106	-0.001482	0.000061926

constant and equal to the present day value (see below). In any case, owing to the low value of the present-day CMB temperature, this effect is neglected. To obtain the total cooling by the metals we use $\Lambda_{\text{metals}} = \sum_i \Lambda_i$ where i indicates a generic level transition for any of the four metals we have included.

The cooling proposed by Maio et al. (2007) is improved implementing the cooling rates by Glover & Jappsen (2007); in particular, we use the de-excitation rates calculated for the collisions with ionized and molecular hydrogen when available. We also include the cooling from neutral C, Si (as in Glover & Jappsen (2007)), Fe, and ionized O (as in Hollenbach & McKee (1989)).

Total cooling. The cooling rate by all the above processes is additive and can be described by

$$\Lambda_{\text{tot}} = \Lambda_X + \Lambda_{\text{H}_2} + \Lambda_{\text{HD}} + \Lambda_{\text{CO}} + (\Lambda_{\text{C}^+} + \Lambda_{\text{O}} + \Lambda_{\text{Si}^+} + \Lambda_{\text{Fe}^+}). \quad (57)$$

where Λ_X is either Λ_{C92} or Λ_{SD93} as appropriate, and all the Λ s are functions of the temperature. Both Λ_{H_2} and Λ_{HD} are the cooling functions of the two molecular species, and the terms in brackets are the cooling of the metals.

The overall rate of temperature change due to the heating and cooling is given by the following equation

$$\frac{dT}{dt} = \frac{\gamma - 1}{k_B \sum_i n_i} (\Gamma - \Lambda) \text{ K/s}, \quad (58)$$

where Γ is the heating source (if present), Λ the cooling source, n_i the number density (the sum is over all the elements), k_B the Boltzmann constant, and γ the adiabatic index defined in Merlin et al. (2011) as

$$\gamma = \frac{5x_{\text{H}} + 5x_{\text{He}} + 5x_{\text{e}^-} + 7x_{\text{H}_2}}{3x_{\text{H}} + 3x_{\text{He}} + 3x_{\text{e}^-} + 5x_{\text{H}_2}}, \quad (59)$$

where x is the fraction of the element indicated by the subscript. For an ideal gas of pure hydrogen this value is 5/3. If the gas is made only of molecular hydrogen we have $\gamma = 7/5$. In eqn. (59) we use a linear fit of the data proposed by Boley et al. (2007) for the adiabatic index of H_2 under $\log(T) \leq 2.6$ (see also erratum), considering a 3:1 ortho/para ratio mix.

The generic heating term Γ can be used to introduce heating phenomena like SNæ feedback, cosmic rays, and others. In the current version of ROBO, Γ is used as a free

parameter that is kept constant during a single run. In any case, the Γ term can be specified by the user according to his scientific aims.

3. Code description

The code has been developed with IDL². Its user friendly features help the development of applications that are otherwise difficult to build in FORTRAN. The code is divided in self-explanatory procedures (routines) that are grouped in four classes (gas chemistry, dust, cooling, and general code behavior). The main routine calls all other subroutines that are needed to calculate the gas evolution. The first group of routines calculates the reaction rates and updates the density of each species.

MASS CONSERVATION. The total mass per unit volume of the ISM at time t is $M(t) = \sum_i n_i(t) m_i$, where $n_i(t)$ is the current number density of the i -th species, m_i its molecular or atomic mass, and the sum is over all the species. While the number densities can vary with time, the total mass must be conserved. In the numerical integration of the system of differential eqns. (1), the conservation of the total mass is not always guaranteed, because it depends on the physical processes, the choice of initial parameters and the time step. Particular care is paid in our test computations to securing and checking run-by-run the conservation of the total mass, i.e. $|M(t_0) - M(t)|/M(t_0) \ll \mathcal{O}(10^{-10})$ where $M(t_0)$ is the initial total mass of the system per unit volume.

TIME STEPS. Even with short time steps, it may happen that some species reach negative values in a single time step. This is because the differential of a species could be negative, and its absolute value higher than the relevant number density. This problem could be solved by forcing the species to be positive. This would cause difficulties with the mass conservation, and then in the subsequent time step the solver would again change the sign of the species abundance, and finally, the cooling would produce some nonphysical effects. In brief, a negative value of the abundance of the species would artificially turn the cooling into heating, which is clearly impossible. The obvious way out is to suitably choose the time step.

NUMERICAL SOLVERS. In our system of differential equations (1) there are 76 reactions to deal with (64 reactions plus 12 photochemical processes), so a fast and

² IDL is a product of ITT Visual Information Solutions, <http://ittvis.com/>

accurate solver is needed. We use the routine LSODE of IDL. LSODE uses adaptive numerical methods to advance the solution of a system of ordinary differential equations by one time step (Hindmarsh 1983; Petzold 1983), optimizing the process for user-defined time steps. In our case LSODE absolute tolerance is 10^{-40} , and the relative one is 10^{-12} .

INTEGRATORS. The number densities of all the particles (atomic species, molecules, dusty components, and free electrons) are governed by the balance between creation and destruction processes and, even more important, span very wide ranges of values.

RANGES OF APPLICABILITY. ROBO can be safely used in the following intervals for temperature, density, and metallicity: $10 \leq T \leq 10^7$ K, $10^{-12} \leq n \leq 10^3$, and $10^{-12} \leq Z \leq 10$. The density range is somewhat limited towards the high value end: observational estimates of the density in molecular clouds can indeed reach higher values. Work is in progress to extend the density range. Furthermore, let us remind the reader that $Z = 1$ means $[\text{Fe}/\text{H}] = 0$, the solar case, and $Z = 10$ means $[\text{Fe}/\text{H}] = 1$; i.e., the ratio of the number densities (Fe/H) is equal to ten times solar. Finally, as not all the reaction rates cover the whole range of values, some extrapolations are required.

The aforementioned ranges of applicability are chosen in such a way as to guarantee the numerical stability of the system. Indeed, the core of the model is the system of differential equations that describe the chemical evolution of the gas according to the reactions listed in Tables 2 through 6. The stability of the system is measured by the conservation of the mass of the ISM elements. Therefore, we carefully checked whether ROBO satisfied this conditions for values of the parameters spanning over large volumes of the parameter space. The results show that this is always the case.

3.1. Free parameters

ROBO contains 47 parameters governing the physics, the mathematics, and the numerical procedure. Here we briefly comment on the most important ones.

- *Ionized fraction of metals*: it fixes the fraction of C, O, Si, and Fe at the first ionization level.
- *Metallicity*: the metallicity of the gas is defined as $Z = 10^{[\text{Fe}/\text{H}]}$. It is worth recalling here that $Z = 1$ is the solar case corresponding to $[\text{Fe}/\text{H}] = 0$ (see below for more details).
- *Number densities*: the number densities of the 28 elemental species and/or molecules, free electrons, and dusty components are all in cm^{-3} .
- *Fraction of carbon dust*: the percentage of carbonaceous grains. If it is equal to 1, all the dust is made by graphite grains and PAHs; if this parameter is equal to 0, all the dust is made by silicates. Intermediate values are also possible.

- *Gas temperature*: the temperature of the gas depends on the cooling and heating processes and changes during the gas evolution.
- *Dust temperature*: the temperature of the dust is kept constant during each run. It controls the formation of the molecules on the surface of the grains and also the accretion of the dust grains themselves.
- *CMB temperature*: the gas temperature cannot go below the temperature of the cosmic microwave background (CMB). At present the CMB temperature is kept constant during each run.
- *Cosmic ray field*: the field formed by the cosmic rays that populate the gas. It destroys or ionizes molecules like H_2 . This field is expressed in s^{-1} , thus corresponding to the rate of ionization of the H_2 by the cosmic rays.
- *Total integration time of a model*: the total time of each run in s.
- *Time step*: the time step used in the models. A typical value is 10^3 years. However, it can be changed by the routine LSODE.
- Finally, there is a number of flags activating or switching off the different sources of heating and cooling and the mechanisms of dust formation and destruction we have described in the previous sections.

4. Models and discussion

To validate ROBO we calculated two groups of models of gas evolution: the first one consists of 600 dust-free cases. Each case corresponds to a different combination of the parameters. The second group consists of 32 models of gas evolution, but the presence of dust grains is taken into account in them.

The aim here is to investigate the evolution of the same elemental species in the presence or absence of the dust. In addition to this, the large number of cases per group allow us to investigate the model response at varying the key parameters. Indeed, owing to the large number of parameters at our disposal, it is not possible to explore the whole parameter space.

Finally, it is worth noticing that the choice of the parameters for the two groups of models is somehow guided by ROBO mainly being designed to become a sort of ancillary tool for NB-TSPH codes. Therefore, in view of this, the two groups of models use the same parameter space adopted by EVOL.

4.1. Dust-free models

Each model is a simulation of a unitary volume of gas in the absence of dust. The results consist of 600 (see below) evolutionary models of the gas temperature and number densities of the 28 species under consideration plus the free electrons. As already emphasized, in each model special care is paid to secure the mass conservation and to avoid unphysical negative number densities. The models of this set are calculated neglecting the presence of any type of

dust, and they are meant to check the internal consistency of ROBO. and

$$n_X = Z n_H \left(\frac{n_X}{n_H} \right)_\odot. \quad (62)$$

4.1.1. Set up of the parameters

Each model gives the thermal and density history of the gas for an assigned set of parameters. The number of parameters and the values assigned to each of them determine the total number of models to calculate. This is given by $N_m = \prod_{i=1}^N p_i$, where N_m is the total number of models, p_i the number of different values for the i -th parameter, and N the total number of the parameters, which in our case amounts to 47 (see Sect. 3.1).

For all the models, we must specify the initial value of the metallicity, the number densities n_{H_2} , n_{H^+} , n_{e^-} , and the gas temperature. We adopt four values for the metallicity $Z = 10^{[Fe/H]}$: $\{0, 10^{-6}, 10^{-3}, 1\}$; five values for the H_2 number density: $\{10^{-10}, 10^{-6}, 10^{-2}, 10^{-1}, 1\} \text{ cm}^{-3}$; three values for the H^+ number density: $\{10^{-10}, 10^{-1}, 1\} \text{ cm}^{-3}$; two values for the electron number density: $\{10^{-10}, 10^{-1}\} \text{ cm}^{-3}$; five values for the gas temperature: $\{10, 10^3, 10^4, 10^6, 10^8\} \text{ K}$. More details on the metallicity are given below when we discuss the number densities of the metals. All the other parameters of the models have a constant value.

In all the models, the following quantities have a fixed initial value. They are $n_H = 1.0$ for the neutral hydrogen, $n_{H^-} = 10^{-9}$ for the hydride, $n_{H_2^+} = 10^{-11}$ for the molecular hydrogen and $n_{He} = 0.08$ for the helium. No helium ions are present at the beginning, so we have $n_{He^+} = n_{He^{++}} = 0$. The initial values of all the deuteroids are set to 10^{-25} : these are the deuterium atom, its ions (D^+ and D^-), the molecules HD and D_2 , and the ion HD^+ .

Metals (i.e. C, O, Si, Fe and their ions C^+ , O^+ , Si^+ , and Fe^+) are computed from the metallicity as follows³. We start from the general

$$[Fe/H] = \log_{10} \left(\frac{n_{Fe}}{n_H} \right) - \log_{10} \left(\frac{n_{Fe}}{n_H} \right)_\odot, \quad (60)$$

where n_{Fe} and n_H are the number densities of iron and hydrogen, and the definition of metallicity we have adopted, namely $Z = 10^{[Fe/H]}$. It must be emphasized that this notation for Z is different from the commonly used definition of metallicity, that is, $Z = 1 - X - Y$ with X and Y indicating the abundances by mass of hydrogen and helium. In the usual meaning, Z is therefore the mass fraction of all the species heavier than helium. In our definition Z is simply related to the iron content $[Fe/H]$ by the expression $Z = 10^{[Fe/H]}$. With this notation, Z can be higher than one: $Z = 1$ corresponds to the solar iron abundance. Considering now a generic metal indicated by X we can write

$$\frac{n_{Fe}}{n_X} = \left(\frac{n_{Fe}}{n_X} \right)_\odot, \quad (61)$$

³ ROBO also lets us insert the value for single metallic species without using the total metallicity.

Therefore if we know the ratio $(n_X/n_H)_\odot$, the total metallicity Z , and the number density n_H of hydrogen in the model, we can derive the number density of the generic metal X . By doing this we assume that for any metallicity the relative abundances of the elements follow the solar partition. In other words we do not consider here the possibility that the gas may have a different distribution of the heavy elements from that in the Sun with respect to the hydrogen (α -enhancement problem).

We use the same method to derive the number density of metal ions given by

$$n_{X^+} = \kappa Z n_H \left(\frac{n_X}{n_H} \right)_\odot, \quad (63)$$

where κ represents the ionized fraction n_{X^+}/n_X of the metal X . In the present models we set $\kappa = 0$, so that all the metals start as neutral species. If the initial temperature is high enough, the hot gas can ionize the metals very early on in the course of the ISM evolution.

The temperature of the gas is a free parameter, but cannot be lower than the temperature of the CMB. For the present models we set $T_{CMB} = 2.73 \text{ K}$, the present day measured value (Boggess et al. 1992; Fixsen 2009).

Each model of the ISM is followed during a total time of $3.15 \times 10^{14} \text{ s}$, which approximately corresponds to 10^7 yrs . This choice stems from the following considerations. Each model of the ISM is meant to represent the thermal-chemical history of a unit volume of ISM whose initial conditions have been established at a given arbitrary time and whose thermal-chemical evolution is followed over a time scale long enough for secular effects to develop but short enough to closely correspond to a sort of instantaneous picture of large-scale evolution of the whole system hosting the ISM unit volume. The initial physical conditions are fixed by a given set of parameters each of which can vary over wide ranges. One has to solve the network of equations for a time scale that is long enough to reveal the variations due to important phenomena such as star formation, cooling, and heating, but not too long to let the system depart from the instantaneous situation one is looking at. The value of 10^7 yr resulted in a good compromise.

The initial value of the time step is $3.15 \times 10^{10} \text{ s} \approx 10^3 \text{ yrs}$. This time step determines the minimum number of steps required to cover the time spanned by a model. It means that each simulation needs at least 10^4 iterations to be completed. The LSODE integrator may introduce shorter time steps depending on the complexity of the problem, so 10^4 is the minimum number of required steps. This value for the time step seems to keep the system stable during the numerical integration.

While integrating, all the sources of cooling are kept active: metal cooling, H_2 cooling according to the prescriptions by Galli & Palla (1998) and Glover & Abel (2008), and finally, the cooling from deuterated hydrogen.

In general, the initial values of the number densities fall into three groups: (i) the elemental species with constant initial values, the same for all the models (namely H^- , H_2^+ , He^+ and He^{++}); (ii) the elemental species whose initial values are derived from other parameters (namely He , all the deuteroids and the metals, such as C and O , which depend on the choice the total metallicity Z), and finally, (iii) the elemental species with free initial conditions (namely H , H_2 , H^+ , and e^-).

Hydrogen group: H , H^+ , H^- , H_2 , and H_2^+ . The initial values of the species H^- and H_2^+ are $n_{\text{H}^-} = 10^{-9}\text{cm}^{-3}$, $n_{\text{H}_2^+} = 10^{-11}\text{cm}^{-3}$ according to Prieto et al. (2008), while the three other hydrogen species have free initial values.

Deuterium group: D , D^+ , D^- , D_2 , HD , and HD^+ . The number densities of the deuteroids are calculated from their hydrogenoid counterparts. For the single atom species we have $n_{\text{D}} = f_{\text{D}} n_{\text{H}}$, $n_{\text{D}^+} = f_{\text{D}} n_{\text{H}^+}$, and $n_{\text{D}^-} = f_{\text{D}} n_{\text{H}^-}$, where $f_{\text{D}} = n_{\text{D}}/n_{\text{H}}$. For the molecules, we can consider the ratio f_{D} as the probability of finding an atom of deuterium in a population of hydrogen-deuterium atoms. This assumption allows us to calculate the HD , D_2 , and HD^+ number densities as a joint probability. For HD and HD^+ we have $n_{\text{HD}} = f_{\text{D}} n_{\text{H}_2}$ and $n_{\text{HD}^+} = f_{\text{D}} n_{\text{H}_2^+}$, but for D_2 is $n_{\text{D}_2} = f_{\text{D}}^2 n_{\text{H}_2}$ as the probability of finding two deuterium atoms is f_{D}^2 . This is valid as long as $f_{\text{D}} \ll 1$.

Helium group: He , He^+ , He^{++} . The ratio $n_{\text{He}}/n_{\text{H}}$ is set to 0.08, thus allowing the initial value of n_{He} to vary according to the initial value for n_{H} . The initial number densities of the species He^+ , He^{++} are both set equal to zero in all the models.

Metals group: C , C^+ , O , O^+ , Si , Si^+ , and Fe , Fe^+ . The Fe number density of the ISM is

$$n_{\text{Fe}} = n_{\text{H}} \cdot \text{dex} \left\{ [\text{Fe}/\text{H}] + \log \left(\frac{n_{\text{Fe}}}{n_{\text{H}}} \right)_{\odot} \right\}, \quad (64)$$

where $(n_{\text{Fe}}/n_{\text{H}})_{\odot}$ is the iron-hydrogen ratio for the Sun. To retrieve the number density of a given metal X we use $n_{\text{X}} = n_{\text{Fe}} \cdot f_{\text{X}}$, where f_{X} is the metal-iron number density ratio in the Sun.

The list of the species whose initial number densities are kept constant in all the models of the ISM is given in Table 11. The values listed here are either fixed to a constant value or based upon the number density of one of the *free* hydrogen species n_{H} , n_{H_2} and n_{H^+} via the f_{D} factor. Since H^- and H_2^+ are constant, then also D^- and HD^+ are fixed. Values are indicated as $a(b) = a \times 10^b$.

The chemical composition of the ISM is typically primordial with a mass abundances of hydrogen $X = 0.76$, and helium $Y = 0.24$ and all metals $Z \simeq 0$. The helium-to-hydrogen number density ratio corresponding to this primordial by mass abundances is $n_{\text{He}}/n_{\text{H}} \simeq 0.08$. The adopted cosmological ratio for the deuterium is $f_{\text{D}} = n_{\text{D}}/n_{\text{H}} \simeq 10^{-5}$. With the aid of these numbers and the above prescriptions we get the number density ratios listed in Table 4.1.1 and the initial values of the number density in turn.

Table 11. Initial values for the number densities of the hydrogen and helium elemental species and the deuteroids. See the text for details.

H^-	1.0(−9)	D	$f_{\text{D}} n_{\text{H}}$
H_2^+	1.0(−11)	D^+	$f_{\text{D}} n_{\text{H}^+}$
He	0.8(−1) n_{H}	D^-	$f_{\text{D}} n_{\text{H}^-}$
He^+	0.0(+0)	HD	$f_{\text{D}} n_{\text{H}_2}$
He^{++}	0.0(+0)	HD^+	$f_{\text{D}} n_{\text{H}_2^+}$
		D_2	$f_{\text{D}}^2 n_{\text{H}_2}$

4.1.2. Results for dust-free models

In this section we discuss the results we have obtained for the dust-free models. Together with this we also consider a side group of models calculated with some specific assumptions (e.g. without metal cooling and others to be explained in the course of the presentation) to underline some effects and to better understand the physical processes taking place in the ISM.

Special attention is paid to check whether there are some unexpected effects or drawbacks in the models and in ROBO. We also describe how different physical quantities affect the overall behaviour of the gas. This is achieved by varying a single parameter at a time and keeping constant all the others.

Temperature and H_2 density. In Fig. 4 we compare the temperature evolution for different values of the initial molecular hydrogen densities in two cases with very different metallicity, i.e. $Z = 10^{-10}$ and $Z = 1$. The important role played by the metal cooling is soon visible comparing the two panels. In the bottom panel (high-metallicity $Z = 1$), cooling is so strong that all the models simultaneously reach in a short time interval nearly the same final temperature of about 10 K. In the top panel (low-metallicity case) the cooling time scale is much longer than in the bottom panel because the contribution to cooling by the metals is negligible. Comparing each curve with the others in both panels, the effect of the molecular hydrogen is evident: higher H_2 densities imply steeper cooling curves and a shift toward lower temperatures. In the metal-free models, the temperature systematically shifts to lower values at increasing the density of the molecular hydrogen (top panel of Fig. 4). The effect is similar but enhanced in the case of metal-rich models (bottom panel of Fig. 4).

Metallicity. Figure 5 shows the evolution of the temperature (top panel) and H_2 number density (bottom panel) of models with different metallicity Z . Each curve is characterized by a different initial metallicity. The sources of UV radiation are at work in these models. As in the previous figure, the cooling strongly depends on the metallicity. The marked knee in the temperature-age relationship is caused by the cooling via the metallicity. Looking at the H_2 -age relationship, we note that the formation of H_2

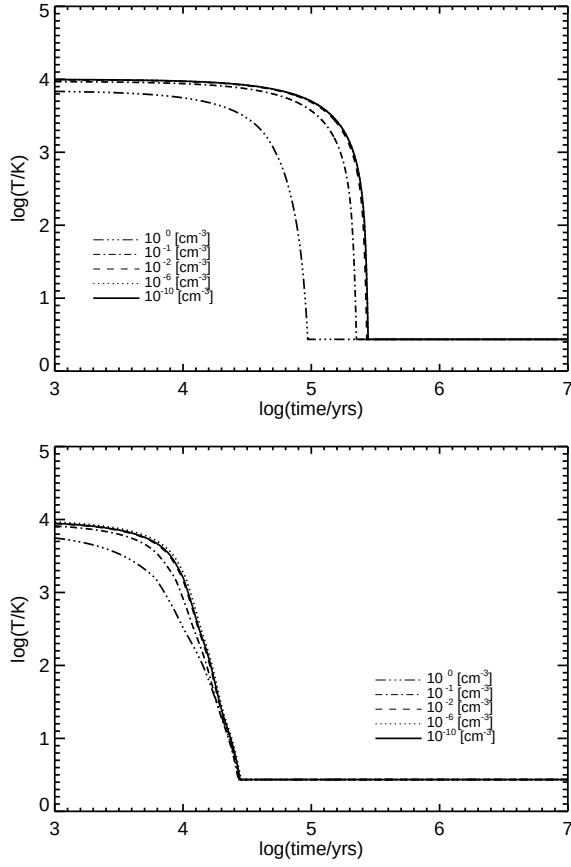


Fig. 4. Temperature evolution for different values of the initial H_2 densities. The lines represent the time evolution in the cases $H_2 = 10^{-10}$ (solid line), $H_2 = 10^{-6}$ (dotted line), $H_2 = 10^{-2}$ (dashed line), $H_2 = 10^{-1}$ (dashed-dotted line), and finally, $H_2 = 1$ (three dots-dashed line). All the densities are in cm^{-3} . Top panel: the models with $Z = 10^{-10}$. Bottom panel: the same as in the top panel but for $Z = 1$. See the text for more details.

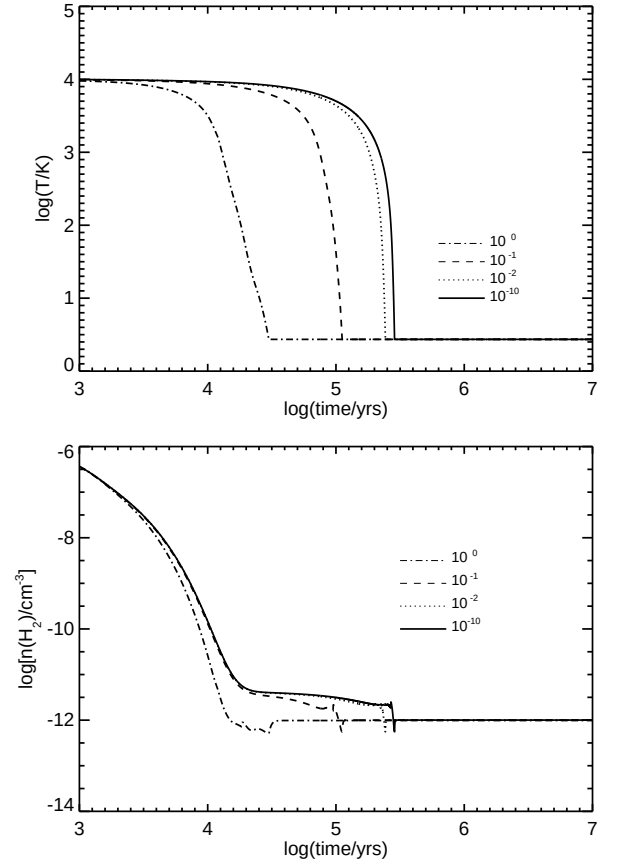


Fig. 5. Top panel: Temperature evolution for different metallicities Z and in presence of the UV radiation field. Bottom panel: evolution of the number density of molecular hydrogen for the same set of parameters. We represent the cases $Z = 1$ (dot-dashed line), $Z = 10^{-1}$ (dashed line), $Z = 10^{-2}$ (dotted line), and finally $Z = 10^{-10}$ (solid line). See the text for more details.

is favored when more free electrons are present (an effect that becomes clear when considering Figs. 5, 6, and 7).

UV radiation. The UV radiation plays a key role in the H_2 density evolution. To cast light on the issue, we calculated models with no UV sources. They are displayed in Fig. 6. The top panel shows the temperature-age relationship, whereas the bottom panel shows the H_2 number density - age relationship. Comparing the bottom panel of Fig. 5 to that of Fig. 6, it is clear the effect of the UV radiation. When the UV radiation is present (Fig. 5), H_2 is destroyed very early on, whereas when the UV radiation is absent (Fig. 6) the number density of H_2 remains nearly constant during the whole evolution. We also note that the effect of the cooling is stronger in the first case. This happens because the UV flux increases the density of free electrons, and since they act as colliders, the eventual higher number of collisions determines a marked improvement in the cooling process. This effect of the UV flux on the number density of free electrons is displayed in Fig.

7, where we show the time evolution of this quantity at varying the metallicity and switching on (top panel) and off (bottom panel) the UV radiation.

Ionized versus neutral metals. The difference in the cooling between Fig. 5 (top panel) and Fig. 6 (top panel) could be attributed to the different efficiencies of the cooling by ionized and neutral metals. Indeed, it is worth noticing that, in order to make the effects of the metals evident, the initial number density of H_2 in Figs. 5 and 6 is set to 10^{-6} . The UV radiation affects the ionization state of atoms and molecules, thus varying the partition between ionized and neutral metals. To clarify this issue, we first compare models in the presence of UV flux in Fig. 8, but the metal cooling by ionized species (top panel) or neutral species (bottom panel) are alternatively switched off. The initial number density of H_2 is always set to 10^{-6} . For each metallicity, the values shown in Fig. 5 (top panel), where both neutral and ionized metals are included, are roughly the sum of the values presented in Fig. 8 (both panels). In the top panel of Fig. 8, the cooling

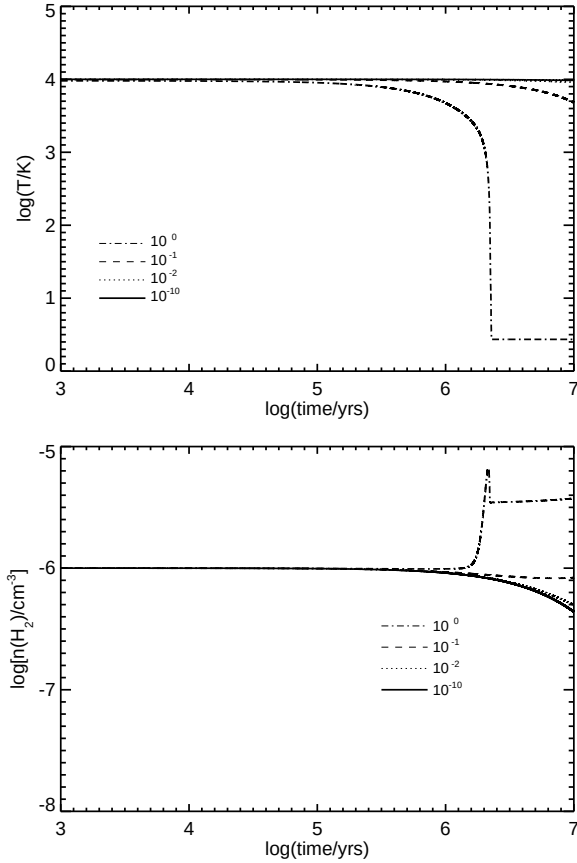


Fig. 6. Top panel: Temperature evolution for different metallicities Z but in the absence of the UV radiation field. Bottom panel: evolution of the molecular hydrogen for the same set of parameters. We represent the following cases: $Z = 1$ (dot-dashed line), $Z = 10^{-1}$ (dashed line), $Z = 10^{-2}$ (dotted line), and finally, $Z = 10^{-10}$ (solid line). See the text for more details.

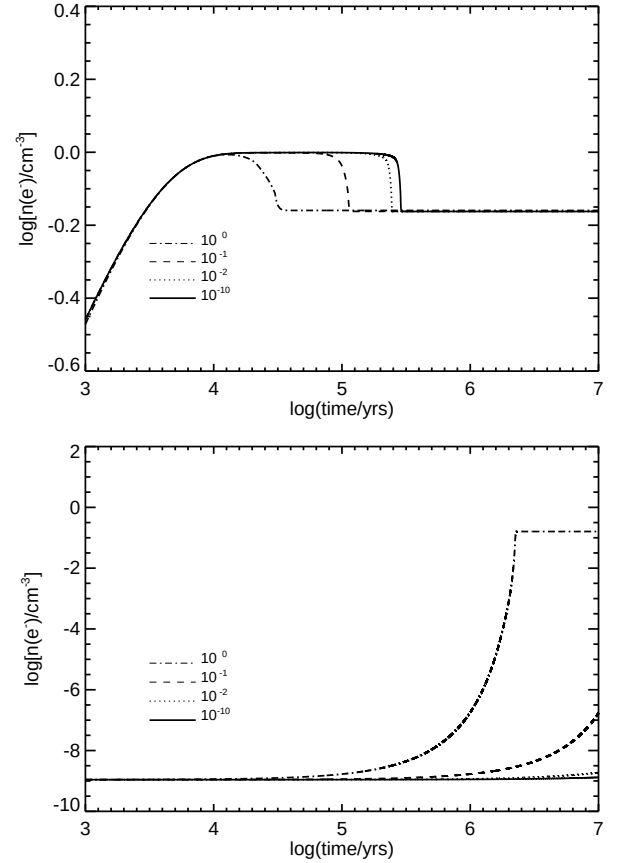


Fig. 7. Temporal evolution of the electrons number density for different metallicities Z and in presence of the UV radiation field (top panel, related to Fig. 5) or without the UV field (bottom panel, same model of Fig. 6). The metallicities are $Z = 1$ (dot-dashed line), $Z = 10^{-1}$ (dashed line), $Z = 10^{-2}$ (dotted line), and finally $Z = 10^{-10}$ (solid line). See the text for more details.

by ionized metals is switched off so that we would expect a behavior similar to that of Fig. 6 (top panel), where the UV radiation is switched off and the neutral metals are favored. It must be pointed out, however, that when the UV is switched off (Fig. 6) most metals are neutral. Even if the ionized metals are switched off (Fig. 8), the fraction of neutral metals will be much lower. Indeed, the shape of the curves displayed in Fig. 8 (top panel) remains similar to those shown in Fig. 5 (top panel), and the long cooling time scales of Fig. 6 cannot be reproduced.

For the same effect of the UV flux on metals, in the top panel of Fig. 8 (where the UV flux is on and the ionized metals are off) cooling does not depend on the metallicity. The UV radiation easily ionizes metals like silicon, carbon, and iron: UV radiation below 13.6 eV from various astrophysical sources generates a UV background that ionizes atoms with a first ionization potential lower than 13.6 eV (Maio et al. 2007). Consequently, with the UV flux switched on, the total amount of neutral metals becomes negligible. If the cooling by ionized metals

is switched off and the contribution to the total cooling by neutral metals is negligible, it follows that the metals almost do not contribute to cooling. When the ionized metals are put back and the negligible amount of neutral metals is dropped (Fig. 8 - bottom panel), we recover the situation of Fig. 5 as expected.

Now the UV flux is switched off as in Fig. 6 and the metal cooling by ionized species (top panel) or neutral species (bottom panel) is alternatively removed. As we can see in Fig. 9, with no cooling by ionized metals and only with the contribution by neutral metals, we are able to reproduce the behavior observed in Fig. 6 (top panel). The case without neutral metals and only with cooling by ionized metals is not shown because it would lead to an almost constant temperature in all the cases, consistent with the picture we have just described. In brief, in this case, both H_2 (the input number density of H_2 is set to a lower value, thus contributing less to the cooling) and the ionized metals (without UV flux, as expected, most of the metals are neutral) little contribute to the cooling process.

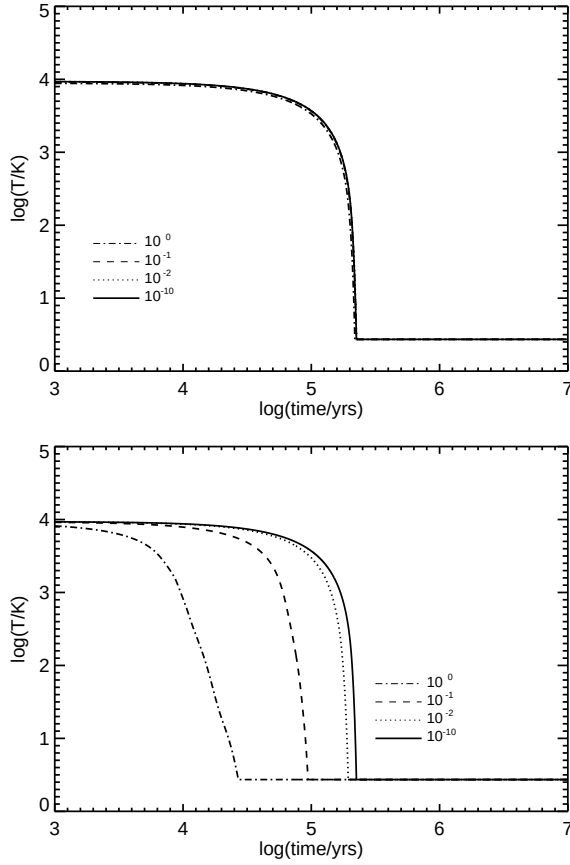


Fig. 8. Temperature evolution for different metallicities Z and with the cooling contribution by ionized or neutral metals alternatively switched off. The *UV flux is switched on*. Top panel: no cooling by ionized metals. Bottom panel: no cooling by neutral metals. In both panels the meaning of the lines is as follows: the dot-dashed line is for $Z = 1$, the dashed line for $Z = 10^{-1}$, the dotted line for $Z = 10^{-2}$, and the solid line for $Z = 10^{-10}$. See the text for more details.

We can finally conclude that the partition between ionized and neutral metals fully explains the difference between Figs. 5 and 6.

Metallicity and H_2 vs. Cooling. In Fig. 10 are plotted four different models with different combinations of the metallicity (very high or very low) and cooling by metals (included or switched off)⁴. Furthermore, to examine the effect of the H_2 cooling, we set the input number density of H_2 equal to 10^{-1} cm^{-3} (see Sect. 4.1.1 for more details on the selected input values for H_2). This plot highlights how the cooling by metals and H_2 affects the temperature and thus the physical status in the gas. The steep decrease of the temperature from the beginning of the simulations and common to all four models is due to the strong H_2 cooling. As expected, the model with high-metallicity and cooling by metals (top panel of

⁴ In these models we switch off the C92 cooling to better highlight the sole effects due to different metallicities.

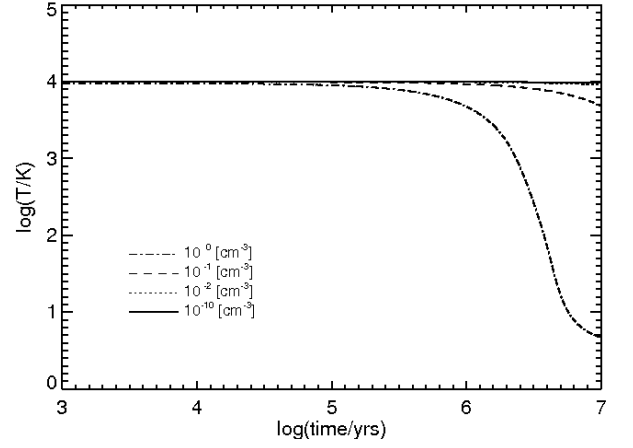


Fig. 9. Temperature evolution for different metallicities Z and with the cooling contribution by ionized metals switched off. Also, the *UV flux is switched off*. The meaning of the lines is as follows: the dot-dashed line is for $Z = 1$, the dashed line for $Z = 10^{-1}$, the dotted line for $Z = 10^{-2}$, and the solid line for $Z = 10^{-10}$. See the text for more details.

Fig. 10, dashed line) is the one experiencing the strongest cooling. The two models with very low-metallicity have a similar behavior independent of the presence or absence of the cooling by metals. The last case, with high-metallicity and no cooling by metals, is the one with the lowest temperature decrease. The reason for it is explained in Fig. 11, where the evolution of H_2 for the four cases under consideration is shown.

The model with high-metallicity and no cooling by metals has the lowest H_2 density compared to the other three and consequently has low cooling. From Fig. 11 we also see that the model with high-metallicity and cooling by metals (dashed line) has the highest H_2 density. This can explain the results in the top panel of Fig. 10, in the sense that the cooling process here could be mainly due to the contribution coming from the molecular hydrogen. The bottom panel of Fig. 10 shows the same models with no cooling by H_2 . In this case only the model with cooling by metals included and high-metallicity undergo a significant cooling. Indeed, if the effect of H_2 is neglected, we need metals in significant amounts to have strong cooling.

4.2. Models with dust: parameter set up

In this section we consider models in which the effect of the dust grains is taken into account. We adopt here the same values for the parameters as in the dust-free models.

The minimum initial metallicity is $Z = 10^{-6}$, and the initial number density of the molecular hydrogen is $n_{H_2} = 10^{-6} \text{ cm}^{-3}$. The number densities of electrons and H^+ are free parameters; they are $n_{H^+} = \{10^{-10}, 10^{-1}\}$ and $n_{e^-} = \{10^{-10}, 10^{-1}\}$, both in units of cm^{-3} as usual. The initial temperature is set to $T = 10^4 \text{ K}$.

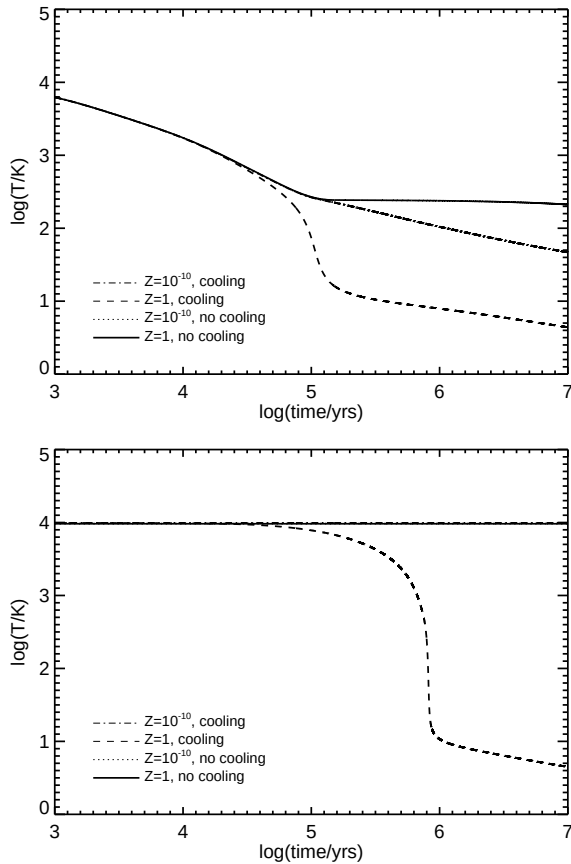


Fig. 10. Temperature evolution with different metallicities Z and cooling options by metals. The meaning of the lines is as follows: high Z and no cooling by metals (solid line), low Z and no cooling by metals (dotted line), high Z and cooling by metals included (dashed line), low Z , and cooling by metals included (dashed-dotted). The dashed-dotted and dotted lines overlap. Top panel: the H_2 cooling is enabled. Bottom panel: the H_2 cooling is switched off. The cooling by Cen (1992) is switched off in both panels.

The only difference with respect to the previous models is the presence of dust. We adopt four values for the number density of dust grains, namely $n_{\text{dust}} = \{0, 10^{-3}, 10^{-2}, 10^{-1}\} \text{ cm}^{-3}$. In these models the composition of the dust mixture is 50% carbonaceous grains and 50% silicates.

We also calculate models with or without dust sputtering by shocks, in order to describe the behavior of a turbulent gas particle with and without the grain depletion due to the shocks.

All the other parameters remain the same as in the dust-free models, such as the number densities of different elements and the cooling processes. In all these models thermal sputtering and dust formation are active, so the dust properties are let change during the evolution of the interstellar medium.

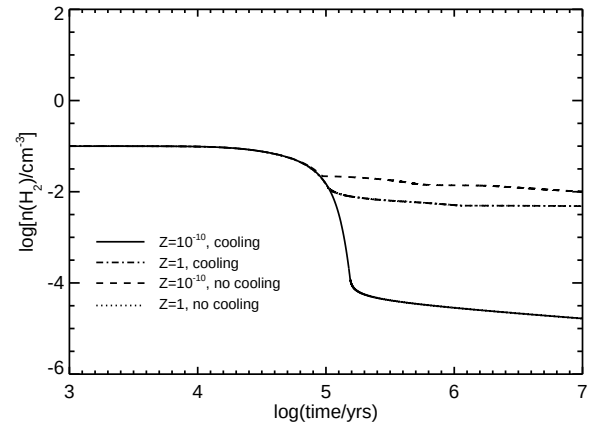


Fig. 11. Evolution of the number density of molecular hydrogen H_2 for different initial metallicity Z and options for the cooling by metals. The meaning of the lines is as follows: high Z and no cooling by metals (solid line), low Z and no cooling by metals (dotted line), high Z and cooling by metals enabled (dashed line), low Z and cooling by metals switched off (dashed-dotted line). The dashed-dotted and dotted lines overlap. The cooling by Cen (1992) is switched off in both panels.

4.2.1. Results for models with dust

The series of plots going from Fig. 12 through Fig. 14 show the temporal evolution of three important quantities of the models, namely the gas temperature, the number densities of dust, and molecular hydrogen for different amounts of initial dust and different initial metallicities. Clearly there is a tight relationship between the initial amount of dust, the temporal behavior of temperature, and the number density of H_2 . First, at a given low-metallicity and by increasing the dust fraction, the temperature decreases earlier and faster (top panel of Fig. 12), whereas if the metallicity is high there is no remarkable effect of the increased dust content (bottom panel of Fig. 12). Also, for the low-metallicity, the trend is anticipated as the dust content increases.

Looking at the temporal evolution of the number density of H_2 shown in the panels of Fig. 13 we note that, in coincidence with the temperature fall off and subsequent gentle decrease, the H_2 number density first decreases and then increases, forming a local minimum. This happens because the dust drives the formation of H_2 : models with more dust suffer more cooling and hence form more H_2 . Only if the metallicity is very high, does the cooling effect of dust grains lose importance as shown by the bottom panel of Fig. 12. To conclude, the curves displayed in the top and bottom panels of Fig. 13 nearly have the same shape, but a different vertical offset. The position and depth of the minima depends on the temperature variation, and the vertical offset clearly depends on the amount of dust present in the gas. This means that amount of dust and H_2 are closely related.

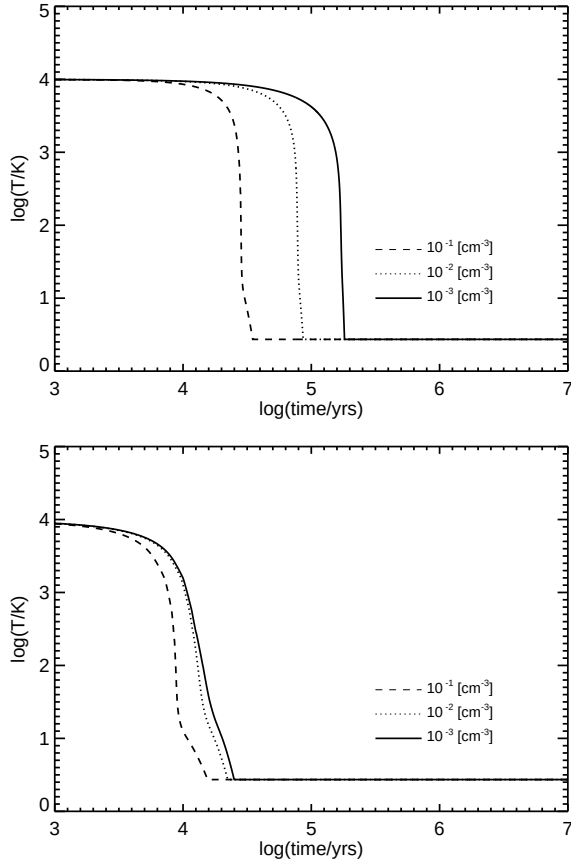


Fig. 12. Temperature evolution for different amounts of initial dust. Top panel: low-metallicity cases. Bottom panel: high metallicity cases. In both panels the meaning of the lines is as follows: the solid line is for $n_{\text{dust}} = 10^{-3}$, dotted line is for $n_{\text{dust}} = 10^{-2}$, and finally, the dashed line is for $n_{\text{dust}} = 10^{-1}$. See the text for more details.

Finally, we calculate and present six models (three for each initial amount of the dust density) that include now the dust destruction by shock sputtering and vaporization. Their temporal evolution is limited to the first 10^5 years only. The results are shown in Fig. 15 (the gas temperature) and Fig. 16 (the dust grains number density as a function of the size of dust grains). Looking at Fig. 15, the temporal evolution of the gas temperature is the same as in the previous case with the sole thermal sputtering at work. In contrast, the dust number density undergoes big changes as shown in Fig. 16, because the shock sputtering is very efficient. Figure 16 shows how the dust distribution changes with the time for the $n_{\text{dust}} = 10^{-3} \text{ cm}^{-3}$ case. After plotting the data of Fig. 16, the dust grains have been grouped in bins according to their size⁵. The lines represent different distributions at different ages, namely

⁵ It worth noticing that even if the initial distribution of dust density per size bin is a power law, assuming that the dimension of the bin size is chosen in such a way that the density per bin is constant, the initial distribution shown in Fig. 16 appears to be flat.

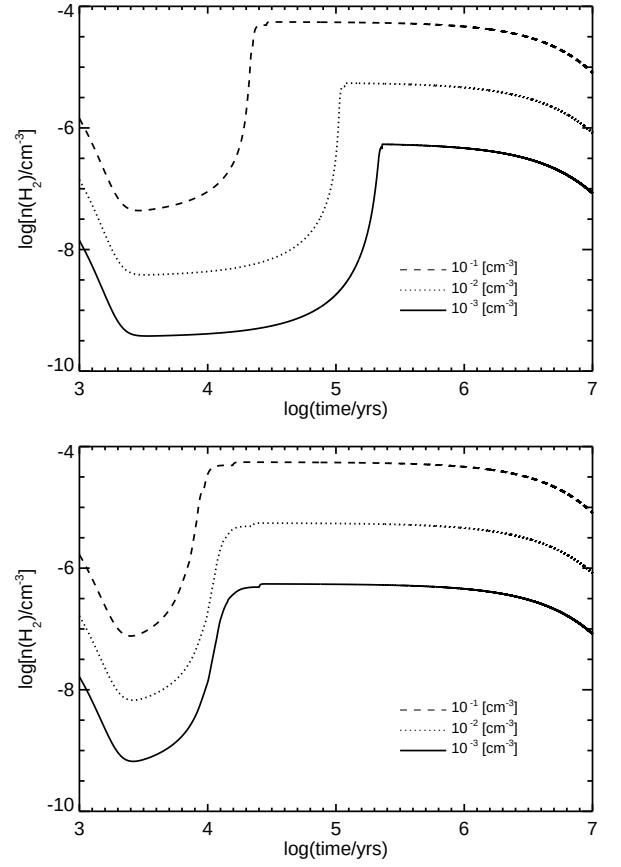


Fig. 13. Evolution of molecular hydrogen for different amounts of initial dust. Top panel: low-metallicity cases. Bottom panel: high metallicity cases. In both panels the meaning of the lines is the same as in Fig. 12. See the text for more details.

$10, 10^2, 10^3$, and 10^4 years from top to bottom. As expected, the shock first destroys the large size grains and then the small ones.

5. Including ROBO results in an NB-TSPH code

In this section we briefly describe how ROBO is planned to be used as an ancillary tool for the NB-TSPH code EVOL to calculate the thermal and chemical properties of the gas particles. In the following we present some preliminary results. The method and its results will be widely discussed in a forthcoming paper (Grassi et al. 2011).

We already mentioned that two different techniques can be used. The first one is a real-time method, in which the elemental abundances are calculated solving the Cauchy problem of the chemical network. It means that for every time step and for every particle, EVOL needs to perform this computation using ROBO as a routine dedicated to this purpose. The advantage is that elemental abundances, thermal properties of the gas particles, and cooling can be calculated with high precision. However, this would require large computational resources, and unfortunately, the more elements it tracks, the more CPU

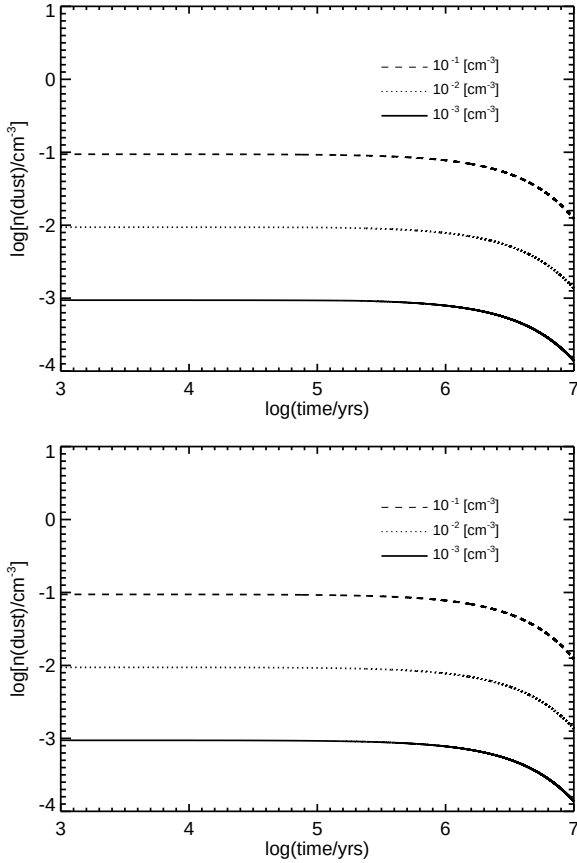


Fig. 14. Dust density evolution for different initial amounts of dust. Top panel: low-metallicity cases. Bottom panel: high metallicity cases. The meaning of the lines is the same as in Fig. 12. See the text for more details.

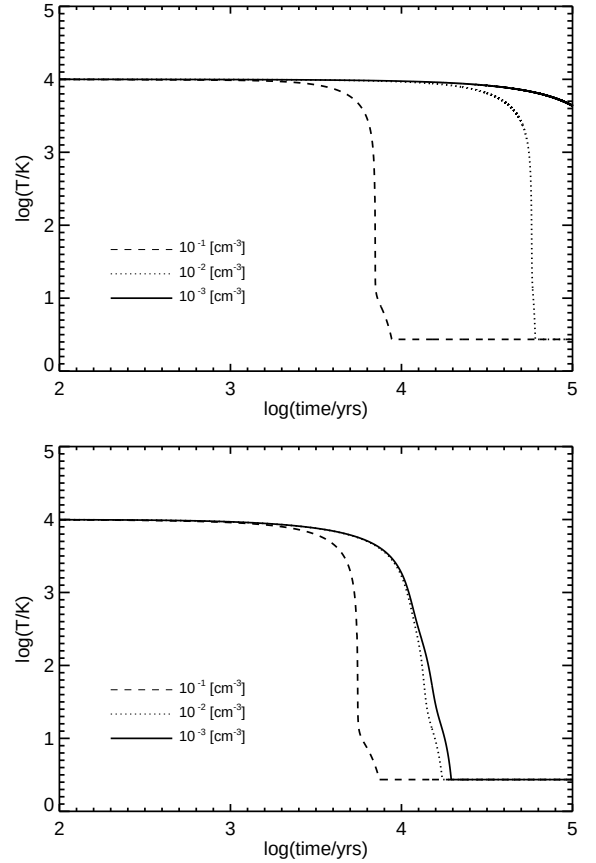


Fig. 15. Temperature evolution for different amounts of dust. Shock disruption is enabled. Top panel: low-metallicity cases. Bottom panel: high-metallicity cases. The meaning of the lines is the same as in Fig. 12. See the text for more details.

time is needed. Furthermore, the network complexity and additional physical processes (e.g. detailed cooling) would increase the computational cost of it.

To avoid these problems an NB-TSPH code can use the s- called “grid” method. It consists of calculating a large number of models exploring different scenarios produced with different sets of the input parameters. The results are then tabulated. When a particle in the cosmological or galaxy simulation needs to know its evolution after a time step, the particle looks for the closest parameters set in the grid, and interpolate the evolved parameters. This a lightweight method, but the main drawback is that the number of free parameters cannot be too large, mainly because interpolating multidimensional grid can be difficult and inaccurate (depending on the grid coarseness).

Our method is similar to the latter one. First of all, we select a set of parameters. Second, we run ROBO for each parameter combination over the total integration time we have chosen (3.15×10^{14} s). The total number of simulations depends on the number of parameters and on the multiplicity of each parameter (see Sect. 4.1.1). This part can easily be achieved using for example a cluster of computers.

Now we have a huge number of models characterized by a large set of parameters. With the standard grid method, interpolating the requested data would be complicated. To solve this problem we use a powerful tool: the ANNs. There are different kinds of ANN architectures depending on the task it needs to perform. We choose the so called back-propagation algorithm (Rumelhart et al. 1986). It is one of the most used and versatile methods for solving a wide range of problems. This mathematical method consists of an algorithm that behaves like the gas model itself. Our ANN learns how the gas evolves using the models produced with ROBO. We first perform a training stage, in which a subset of randomly chosen models is shown to the ANN. After some iterations the ANN converges to a stable state. At this point we look at the complementary of the subset of models: its aim is to calculate the ANN error on unknown data. If the error is small enough, we can assert that the ANN behaves like the gas model. When this happens, in EvOL ROBO can be safely replaced by the ANN.

The great advantage is that the ANN can give the requested solution with ease. It is very light, since it consists

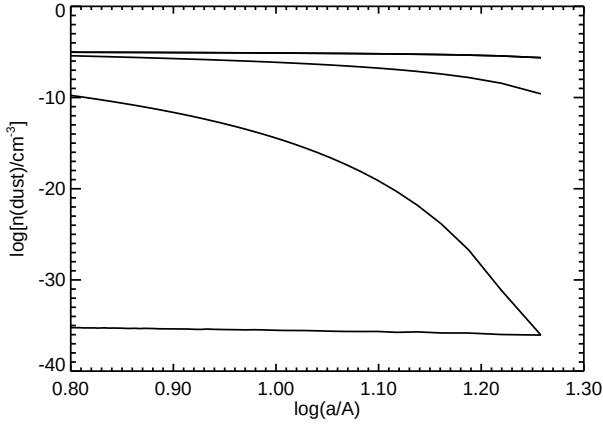


Fig. 16. Dust density evolution for different dimension bins. Shock disruption is enabled. Lines represent the distribution of dust among the bins for different ages (from top to bottom, 10, 10^2 , 10^3 , and 10^4 years). See the text for more details.

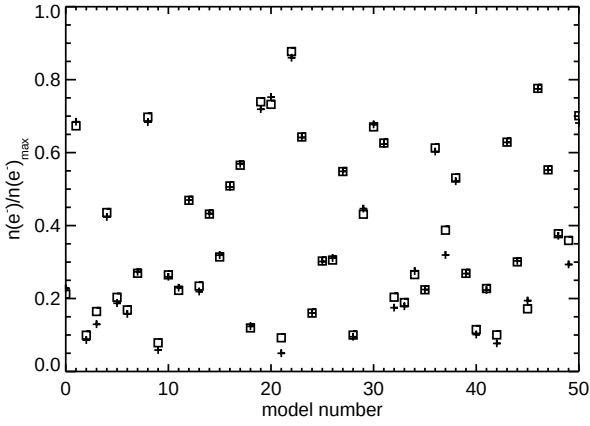


Fig. 17. Comparison between the theoretical data calculated by ROBO for the free electron number density and fed to the ANNs (squares) and the same data retrieved by the ANNs (crosses). Each input data corresponds to a model calculated by ROBO for a set of input parameters, hence to a network data point of the hyperspace of parameters handled by the ANNs. The input and predicted data refer to the free electron number density after 10^4 years of evolution. The data are normalized to the maximum value of the distribution. Similar plots are easily obtained for all the other physical quantities relative to the ISM.

of three small matrices of so-called synaptic weights. Each particle needs only to perform a matrix product, which is (in this scenario) a fast algebraic operation. In this way retrieving chemical evolution for each particle is fast. To illustrate the ability of the ANN to provide good representations of the physical state of the ISM in Fig. 17, we show the comparison between the theoretical data calculated by ROBO for the free electron number density and fed to the ANN (squares) and the same data retrieved by

the ANN (crosses). The agreement is remarkably good. The same technique can be applied to any other variable of interest here.

6. Conclusions

We have presented a model of the ISM that provides a detailed description of the gas chemistry and evolution, the formation and destruction of dust grains of different types, and finally, a thorough description of the cooling process over a wide range of physical parameters and initial conditions. The way the model is conceived corresponds to an instantaneous picture of the physical state of an elementary volume of the ISM characterized by a set of physical parameters assumed here as the initial conditions of a given volume element. Under the action of the ISM models, the initial physical state evolves on a secular time scale. This provides us a sort of vector field telling how a given physical state will evolve (how much and in which direction in the multidimensional space of the physical conditions). The integral of the elementary volumes over the underlying evolutionary path of the grand physical quantities like density and temperature (all of these functions of space and time) of the host system (a galaxy or a cosmological simulation) will give us the detailed evolution of the ISM. This is the big advantage offered by the model, securing it a wide range of applicability.

We have presented here the results for dust-free and dust-rich ISM at varying the key parameters. The first group of models for a dust-free medium is meant to understand how the ISM behaves in the absence of dust grains. These models highlight the importance of the different kinds of cooling that are dominant in different kinds of environments. In particular, we call attention to the role of metals and free electrons in driving the physical behavior of the ISM via their effect on the gas cooling during its evolution. The dust-rich ISM allows us to understand how the ISM responds to the presence of the dust. In particular, we analyzed the temperature variations caused by the presence of dust in different amounts. We have also explored how the creation and destruction of the dust grains (the latter induced by shock and thermal sputtering) affects the evolution of the ISM.

The ISM model and companion code were created as auxiliary tools for NB-TSPH simulations in the context of galaxy cosmological simulations of the Universe and models of galaxy formation, structure, and evolution. Our specific aim is to give a more accurate description of the gas component in EvoL or in similar codes in literature. Finally, ROBO is also designed to run in small and middle-size computers. Thanks to ROBO, detailed gas physics can be inserted in NB-TSPH simulations at low computational costs.

To include the results of ROBO in our NB-TSPH code, we plan to use the ANNs that are more accurate, faster, and easier to implement than the standard fits on multidimensional grids. A complete account of this will be made public soon (Grassi et al. 2011).

Future implementations of ROBO are planned, among which we mention the inclusion of the photo-ionization by single stellar populations of different age and chemical composition in the chemical network and a better determination of the grains temperature that is tightly related to the local stellar radiation field.

Acknowledgements. The authors would like to thank the referee, Dr. Simon Glover, for useful and constructive remarks that greatly helped improve the first version of the manuscript.

T. Grassi is grateful to Dr. F. Combes for the kind hospitality at the Observatoire de Paris - LERMA under EARA grants, where part of the work was developed and for the many stimulating discussions.

P. Krstic acknowledges support from the US DOE Office of Fusion Sciences through ORNL, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

References

- Abel, T., Anninos, P., Zhang, Y., & Norman, M. L. 1997, *New Astronomy*, 2, 181
- Anninos, P., Zhang, Y., Abel, T., & Norman, M. L. 1997, *New Astronomy*, 2, 209
- Bakes, E. L. O. & Tielens, A. G. G. M. 1994, *ApJ*, 427, 822
- Barnett, C. F., Hunter, H. T., Fitzpatrick, M. I., Alvarez, I., Cisneros, C., & Phaneuf, R. A. 1990, NASA STI/Recon Technical Report N, 91, 13238
- Boggess, N. W., Mather, J. C., Weiss, R., Bennett, C. L., Cheng, E. S., Dwek, E., Gulkis, S., Hauser, M. G., Janssen, M. A., Kelsall, T., Meyer, S. S., Moseley, S. H., Murdock, T. L., Shafer, R. A., Silverberg, R. F., Smoot, G. F., Wilkinson, D. T., & Wright, E. L. 1992, *ApJ*, 397, 420
- Boley, A. C., Hartquist, T. W., Durisen, R. H., & Michael, S. 2007, *ApJ*, 660, L175
- Cazaux, S., Caselli, P., Cobut, V., & Le Bourlot, J. 2008, *A&A*, 483, 495
- Cazaux, S. & Spaans, M. 2009, *A&A*, 496, 365
- Cen, R. 1992, *ApJS*, 78, 341
- Croft, H., Dickinson, A. S., & Gadea, F. X. 1999, *MNRAS*, 304, 327
- Dalgarno, A. & Lepp, S. 1987, in *IAU Symposium*, Vol. 120, *Astrochemistry*, ed. M. S. Vardya & S. P. Tarafdar, 109–118
- Depristo, A. E., Augustin, S. D., Ramaswamy, R., & Rabitz, H. 1979, *J. Chem. Phys.*, 71, 850
- Draine, B. T. & Lee, H. M. 1984, *ApJ*, 285, 89
- Draine, B. T. & Salpeter, E. E. 1979, *ApJ*, 231, 77
- Dwek, E. 1998, *ApJ*, 501, 643
- Efstathiou, G. 1992, *MNRAS*, 256, 43P
- Ferland, G. J., Korista, K. T., Verner, D. A., Ferguson, J. W., Kingdon, J. B., & Verner, E. M. 1998, *PASP*, 110, 761
- Fixsen, D. J. 2009, *ApJ*, 707, 916
- Galli, D. & Palla, F. 1998, *A&A*, 335, 403
- Gealy, M. W. & van Zyl, B. 1987, *Phys. Rev. A*, 36, 3091
- Glover, S. C., Savin, D. W., & Jappsen, A. 2006, *ApJ*, 640, 553
- Glover, S. C. O. & Abel, T. 2008, *MNRAS*, 388, 1627
- Glover, S. C. O., Federrath, C., Mac Low, M., & Klessen, R. S. 2010, *MNRAS*, 404, 2
- Glover, S. C. O. & Jappsen, A. 2007, *ApJ*, 666, 1
- Glover, S. C. O. & Savin, D. W. 2009, *MNRAS*, 393, 911
- Goldflam, R., Kouri, D. J., & Green, S. 1977, *J. Chem. Phys.*, 67, 4149
- Grassi, T., Merlin, E., Piovan, L., Buonomo, U., & Chiosi, C. 2011, *ArXiv/1103.0509*
- Hindmarsh, A. C. 1983, *IMACS Transactions on Scientific Computation*, 1, 55
- Hirashita, H. & Yan, H. 2009, *MNRAS*, 394, 1061
- Hocuk, S. & Spaans, M. 2010, *A&A*, 510, A110+
- Hollenbach, D. & McKee, C. F. 1979, *ApJS*, 41, 555
- . 1989, *ApJ*, 342, 306
- Janev, R. K., Langer, W. D., Post, Jr., D. E., & Evans, K. J. 1987, *Shock and Vibration*, 4
- Katz, N., Weinberg, D. H., & Hernquist, L. 1996, *ApJS*, 105, 19
- Krstić, P. S. 2002, *Phys. Rev. A*, 66, 042717
- Kusakabe, T., Kimura, M., Pichl, L., Buenker, R. J., & Tawara, H. 2003, *Phys. Rev. A*, 68, 050701
- Laor, A. & Draine, B. T. 1993, *ApJ*, 402, 441
- Leitch-Devlin, M. A. & Williams, D. A. 1985, *MNRAS*, 213, 295
- Li, A. & Draine, B. T. 2001a, in *Bulletin of the American Astronomical Society*, Vol. 33, *Bulletin of the American Astronomical Society*, 1451–+
- Li, A. & Draine, B. T. 2001b, *ApJ*, 554, 778
- . 2001c, *ApJ*, 550, L213
- Lipovka, A., Núñez-López, R., & Avila-Reese, V. 2005, *MNRAS*, 361, 850
- Maio, U., Dolag, K., Ciardi, B., & Tornatore, L. 2007, *MNRAS*, 379, 963
- Mathis, J. S., Mezger, P. G., & Panagia, N. 1983, *A&A*, 128, 212
- Mathis, J. S., Rumpl, W., & Nordsieck, K. H. 1977, *ApJ*, 217, 425
- McGreer, I. D. & Bryan, G. L. 2008, *ApJ*, 685, 8
- McKee, C. F., Storey, J. W. V., Watson, D. M., & Green, S. 1982, *ApJ*, 259, 647
- Meijerink, R. & Spaans, M. 2005, *A&A*, 436, 397
- Merlin, E., Buonomo, U., Grassi, T., Piovan, L., & Chiosi, C. 2010, *A&A*, 513, A36+
- . 2011, in preparation
- Merlin, E. & Chiosi, C. 2006, *A&A*, 457, 437
- . 2007, *A&A*, 473, 733
- Mizusawa, H., Omukai, K., & Nishi, R. 2005, *PASJ*, 57, 951
- Moseley, J., Aberth, W., & Peterson, J. R. 1970, *Physical Review Letters*, 24, 435
- Navarro, J. F. & Steinmetz, M. 1997, *ApJ*, 478, 13
- Nelson, R. P. & Langer, W. D. 1997, *ApJ*, 482, 796
- Petuchowski, S. J., Dwek, E., Allen, Jr., J. E., & Nuth, III, J. A. 1989, *ApJ*, 342, 406

- Petzold, L. 1983, *SIAM Journal on Scientific and Statistical Computing*, 4, 136
- Prieto, J. P., Infante, L., & Jimenez, R. 2008, *astroph/0809.2786*
- Ruffle, D. P., Rae, J. G. L., Pilling, M. J., Hartquist, T. W., & Herbst, E. 2002, *A&A*, 381, L13
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986, *Nature*, 323, 533
- Santoro, F. & Shull, J. M. 2006, *ApJ*, 643, 26
- Savin, D. W., Krstić, P. S., Haiman, Z., & Stancil, P. C. 2004, *ApJ*, 606, L167
- Schinke, R., Engel, V., Buck, U., Meyer, H., & Diercksen, G. H. F. 1985, *ApJ*, 299, 939
- Smith, B., Sigurdsson, S., & Abel, T. 2008, *MNRAS*, 385, 1443
- Sutherland, R. S. & Dopita, M. A. 1993, *ApJS*, 88, 253
- Tielens, A. G. G. M., McKee, C. F., Seab, C. G., & Hollenbach, D. J. 1994a, *ApJ*, 431, 321
- . 1994b, *ApJ*, 431, 321
- Vedel, H., Hellsten, U., & Sommer-Larsen, J. 1994, *MNRAS*, 271, 743
- Verner, D. A. & Ferland, G. J. 1996, *ApJS*, 103, 467
- Voronov, G. S. 1997, *Atomic Data and Nuclear Data Tables*, 65, 1
- Walmsley, C. M., Flower, D. R., & Pineau des Forêts, G. 2004, *A&A*, 418, 1035
- Weingartner, J. C. & Draine, B. T. 2001a, *ApJ*, 548, 296
- . 2001b, *ApJS*, 134, 263
- Woodall, J., Agúndez, M., Markwick-Kemper, A. J., & Millar, T. J. 2007, *A&A*, 466, 1197
- Zhao, L. B., Stancil, P. C., Gu, J. P., Liebermann, H., Li, Y., Funke, P., Buenker, R. J., Zygelman, B., Kimura, M., & Dalgarno, A. 2004, *ApJ*, 615, 1063

MaNN: Multiple Artificial Neural Networks for modelling the Interstellar Medium

Tommaso Grassi¹, Emiliano Merlin¹, Lorenzo Piovan¹, Umberto Buonomo¹, Cesare Chiosi¹

¹ Department of Astronomy, Padova University, Vicolo dell'Osservatorio 3, I-35122, Padova, Italy
e-mail: tommaso.grassi@unipd.it, cesare.chiosi@unipd.it

Received: March 2011; Revised: *** ***, Accepted: *** ***

ABSTRACT

Aims. Modelling the complex physics of the Interstellar Medium (ISM) in the context of large-scale numerical simulations is a challenging task. A number of methods have been proposed to embed a description of the ISM into different codes. We propose a new way to achieve this task: Artificial Neural Networks (ANNs).

Methods. The ANN has been trained on a pre-compiled model database, and its predictions have been compared to the expected theoretical ones, finding good agreement both in static and in dynamical tests run using the Padova Tree-SPH code EvOL.

Results. A neural network can reproduce the details of the interstellar gas evolution, requiring limited computational resources. We suggest that such an algorithm can replace a real-time calculation of mass elements chemical evolution in hydrodynamical codes.

Key words. ISM: evolution - methods: numerical - galaxies: evolution

1. Introduction

In the framework of modern large-scale numerical hydrodynamical simulations, the correct modelling of the Interstellar Medium (ISM) plays an increasingly crucial role as many physical phenomena such as star formation, energy feedback, diffusion of heavy elements and others, strongly depend on it and simultaneously affect the properties of the ISM. The task is, however, very demanding. This is ultimately due to the complexity of the hydrodynamical interactions of the Baryonic Matter in large scale systems. Contrary to the simple behaviour of the Dark Matter dynamically governed only by gravity, the behavior of the Baryonic Matter is the result of many complex and interwoven physical phenomena. Therefore, to obtain a realistic model of the ISM, many physical processes must be taken into account going from chemical reactions among atoms and molecules occurring in complicated networks, dust formation and destruction, chemical reactions taking place on dust grains, to heating and cooling by various mechanisms, photo-ionization and radiative transfer. These processes span wide ranges of temperature and density, thus making their accurate, simultaneous description a cumbersome affair.

Moreover, even when the physics is described with the desired accuracy, the number of mass elements (particles)

customarily used in large-scale simulations is large (typically 10^4 to 10^7), so that large amounts of computational workload are requested. Clearly, computing the internal evolution of each mass element in detail is extremely time-consuming, whatever the method chosen to follow it might be.

The aim of this study is to present a method able to alleviate part of the above difficulties by approximating the exact solution of a detailed physical model and yet ensuring both a robust and sufficiently precise description of the gas properties, while maintaining the computational workload to an acceptable level.

The method stands on the model of the ISM and companion code ROBO developed by Grassi et al. (2010). In brief, ROBO follows the evolution of a gas element during a given time interval Δt . In other words, given the initial physical and chemical state of a gas element as functions of its hydrodynamical properties, the relative abundances of its chemical constituents and the kind of interactions with the surrounding medium, the model calculates the *final* state after some time interval Δt .

The straight implementation of ROBO into a generic hydrodynamical simulation (and companion code) to perform the on-the-fly computation of the ISM properties, although desirable, would be far too expensive in terms

of computational costs. The easiest way of proceeding would be to store the results of ROBO as look-up tables (grids) where the values of interest (e.g. the physical parameters of the gas element: mass fractions of molecular, neutral, ionized hydrogen; temperature, density, pressure; the abundance of dust, and so on) are memorized for a given (large) set of initial configurations. However, each parameter to be stored corresponds to one dimension of the look-up table (grids). Thus, tracking N parameters requires a N -dimensional matrix. Such matrix is discrete, so an interpolation in a N -dimensional space is required to know the final state corresponding to a given initial state of a generic gas element. The all procedure when applied many times to a high number of gas particle would be very time consuming. A different strategy must be conceived.

To cope with this, we develop an algorithm based on the Artificial Neural Networks (ANN) technique to establish the correspondence between the *initial* + *final* state. The ANN technique can handle a large number of parameters without requiring a real-time interpolation. The ANNs require few floating point operations and “learn” to behave like the original physical models on which they are trained. The computational cost of it is easily affordable.

The paper is organized as follows. In Section 2 we briefly describe the ISM model and companion code ROBO. In Section 3 we briefly review the concept and algorithm of the ANN in general, whereas in Section 4 we present the ANN we have set up for our purposes. This is named MaNN. In the same section, we also examine the ability of MaNN in reproducing the data calculated by ROBO. In Section 5 we describe how MaNN is implemented into the NB-TSPH code EVOL of Merlin et al. (2010) and present a suitable hydrodynamical test aimed at checking the robustness of the method. Finally, in Section 7 we summarize our results and draw some general considerations.

2. Robo: the ISM model and companion code

ROBO is a numerical code specifically designed to study the evolution of the ISM. It includes several atomic and molecular species linked together by a large network of reactions. In the following we summarize the main features of ROBO and emphasize the fact that ROBO is specifically tailored to follow the evolution of the coolants owing to their important role in the wider context of the ISM evolution. To this purpose, we follow the temporal variation of molecules like H_2 , HD and metals like C, O, Si, Fe and their ions. For all the details the reader should refer to Grassi et al. (2010).

LAYOUT OF THE ISM MODEL. The model deals with an ideal ISM element of unit volume, containing gas and dust in arbitrary initial proportions, whose initial physical conditions are specified by a set of parameters, which is let evolve for a given time interval. The history leading the element to that particular initial physical state is not of interest here. The ISM element is mechanically isolated from the host environment, i.e. it does not expand

or contract under the action of large scale forces, however it can be interested by the passage of shock waves originated by physical phenomena taking place elsewhere (e.g. supernova explosions). Furthermore, it does not acquire nor lose material, so that the conservation of the total mass applies, even if its chemical composition can change with time. It is immersed in a bath of UV radiation generated either by nearby or internal stellar sources and in a field of cosmic ray radiation. It can generate its own radiation field by internal processes and so it has its own temperature, density and pressure, each other related by an Equation of State (EoS). If observed from outside, it would radiate with a certain spectral energy distribution. For the aims of this study, we do not need to know the whole spectral energy distribution of the radiation field pervading the element, but only the UV component of it. Given these hypotheses and the initial conditions, the ISM element evolves toward another physical state under the action of the internal network of chemical reactions changing the relative abundances of the elemental species and molecules, the internal heating and cooling processes, the UV radiation field, the field of cosmic rays, and the passage of shock waves. The model is like an operator determining in the space of the physical parameters conditions the vector field of the local transformations of the ISM from one initial state to a final one. This is the greatest merit of this approach, which secures the wide applicability of the model.

The chemical network. The following species are tracked by ROBO: H, H^+ , H^- , H_2 , D, D^+ , D^- , D_2 , HD, HD^+ , He, He^+ , He^{++} , C, C^+ , O, O^+ , Si, Si^+ , Fe, Fe^+ , and e^- . The list of reactions and their rates that are included in ROBO can be found in Grassi et al. (2010). The reactions are of different type and can be grouped as follows

- collisional ionization ($A + e^- \rightarrow A^+ + 2e^-$),
- photo-recombination ($A^+ + e^- \rightarrow A + \gamma$),
- dissociative recombination ($A_2^+ + e^- \rightarrow 2A$),
- charge transfer ($A^+ + B \rightarrow A + B^+$),
- radiative attachment ($A + e^- \rightarrow A^- + \gamma$),
- dissociative attachment ($A + B^- \rightarrow AB + e^-$),
- collisional detachment ($A^- + e^- \rightarrow A + 2e^-$),
- mutual neutralization ($A^+ + B^- \rightarrow A + B$),
- isotopic exchange ($A_2^+ + B \rightarrow AB^+ + A$),
- dissociations by cosmic rays ($AB + CR \rightarrow A + B$),
- neutral-neutral ($AB + AB \rightarrow A_2 + B_2$),
- ion-neutral ($AB^+ + AB \rightarrow AB_2^+ + A$),
- collider ($AB + C \rightarrow A + B + C$),
- ionizations by field photons ($A + \gamma \rightarrow A^+ + e^-$)

where A and B are two generic atoms, γ is a photon. Note that cosmic rays and photo-ionization effects are included.

DUST. ROBO also tracks the evolution of dust, in turn made by several components, of which it follows the formation and destruction. The formation of dust is modelled according to the prescriptions by Dwek (1998), however with some modifications that allow us to know when a species belonging to the gas phase is captured by dust

(trapped at the surface of dust grain lattice). The evolution of the dust type (both in abundance and dimensions) is followed for silicates and carbonaceous grains. Dust is also destroyed by thermal sputtering and shocks. The first process is modeled using the description by Draine & Salpeter (1979), that depends on the environment density and the grain size. We also add the dependence on temperature, considering that hot gas disrupts the dust grains more efficiently than the cold one. The dust disruption by shocks requires the velocity distribution of the gas component to be known. This is supposed to be in the turbulence regime and the velocity field to be suitably described by the Kolmogorov-law. This allows us to describe the shocks on very small scales (contrary to what usually happens in large-scale simulations that do not have the required spatial resolution). Dust is an important component of the ISM because it affects the formation of H_2 (one of the most efficient coolants of the ISM) and HD and also the gas phase catalyzing some chemical processes. The formation of H_2 and HD is modeled as in Cazaux & Spaans (2009), whose description depends on the gas and dust temperature. Finally, the photoelectric ejection of electrons from dust grains is described in detail as well. We can estimate the amount of heating (and cooling) produced by each size-bin of the grain distribution. For further details, see Grassi et al. (2010).

COOLING. Several cooling processes are considered in ROBO. In the high temperatures regime ($T \geq 10^4$ K) the metallicity dependent cooling rates of Sutherland & Dopita (1993) are adopted. For temperatures lower than 10^4 K we consider the following processes: (i) cooling by molecular hydrogen according to the formulation by Galli & Palla (1998), however, supplemented by the results of Glover & Jappsen (2007) who take in account the $H_2 - H$ interaction and the collisions with He, H^+ , H_2 and free electrons; (ii) cooling by metals is modeled including C, Si, O, Fe, and their ions as in Maio et al. (2007), Glover & Jappsen (2007) and Hollenbach & McKee (1989); (iii) cooling by deuterated molecular hydrogen according to the model by Lipovka et al. (2005); (iv) cooling by the CO molecule and (v) finally, cooling by Cen (1992).

The total cooling rate is the sum of all the contributions

$$\Lambda_{\text{tot}} = \Lambda_{\text{SD}} + \Lambda_{H_2} + \Lambda_{\text{HD}} + \Lambda_{\text{CO}} + \Lambda_{\text{metals}} + \Lambda_{\text{CEN}},$$

where the various terms Λ s are all functions of temperature and density and Λ_{SD} is the cooling rate by Sutherland & Dopita (1993), Λ_{H_2} , Λ_{HD} and Λ_{CO} are the cooling rates of the indicated molecular species, and Λ_{metals} is the cooling by metals (C, O, Si and Fe). Finally, Λ_{CEN} is the cooling by Cen (1992).

It is worth to notice that in the tests presented in this paper the cooling proposed by Cen (1992) is disabled (i.e. $\Lambda_{\text{CEN}} = 0$).

HEATING. For the photo-dissociation of the molecular hydrogen and the UV pumping, the H and He photo-ionization, the H_2 formation in the gas and dust

phase and, finally, the ionization from cosmic rays, heating is described as in Glover & Jappsen (2007). For the heating due to the photoelectric effect on dust grains, the model proposed by Bakes & Tielens (1994) and Weingartner & Draine (2001) is used.

INTEGRATION TIME OF THE MODELS. Each model of the ISM is followed during a total time of about 10^4 yr. This choice stems from the following considerations. Each model of the ISM is meant to represent the thermal-chemical history of a unit volume of ISM, whose initial conditions have been established at certain arbitrary time and whose thermal-chemical evolution is followed over a time scale comparable to the maximum time step of the TreeSPH code. The initial physical conditions are fixed by a given set of parameters each of which can vary over wide ranges. One has to solve the network of equations for a time scale long enough to reveal the variations due to important phenomena such star formation, cooling and heating, but not long to allow to the system to depart from the instantaneous situation one is looking at. The value of 10^4 yr resulted to be a good compromise. The initial value of the time-step is 1 yr. This time-step determines the minimum number of steps required to cover the time spanned by a model. It means that each simulation needs at least 10^4 iterations to be completed. The numerical integrator may introduce shorter time steps depending on the complexity of the problem. This value for the time-step seems to keep the system stable during the numerical integration¹.

DATABASE OF MODELS. Varying the initial value of the parameters describing the ISM (such as the temperature, density, abundances of elemental species etc.), ROBO is used to create a large number of evolutionary models.

3. The Artificial Neural Networks

As already mentioned, the ideal way of proceeding would be to insert ROBO into a code simulating the temporal evolution of large scale structures and to calculate the thermodynamical properties of the ISM component. In all practical cases, this would be unreasonably time consuming. To cope with it without loosing accuracy in the physical description of the ISM we construct an ANN able to replace ROBO in all cases.

In brief, ANNs are non-linear tools for statistical data modeling, used to find complex relationships between input and output data or to discover patterns in complex datasets. The first idea of an ANN is by Hebb (1949). In the early sixties Rosenblatt (1962) built the first algorithm based on the iterative penalty and reward method, and finally, after two decades of silence,

¹ In Grassi et al. (2010) the adopted total integration interval was about 10^3 longer, i.e. about 10^7 yr, because we were interested in exploring the chemical evolution of the ISM over a sizable time scale, whereas now we are interested in the evolution over a time scale comparable to that of typical time steps of NB-TSPH simulations.

new ideas were brought by Carpenter & Grossberg (1987); Carpenter & Grossberg (1987, 1990) (adaptive resonance theory), Hopfield (1982, 1984) (associative neural network) and Rumelhart et al. (1986); McClelland et al. (1986), who developed the *back-propagation algorithm*, today a common and widely used training scheme.

Demanding computational problems (e.g. multidimensional fits or classifications) can be easily solved using ANNs whereas other methods would require large computing resources. An ANN is based on a simple conceptual architecture inspired by the biological nervous systems. It looks like a network composed by *neurons* and linked together by *synapses* (numerical weights). This structure is modeled with a simple algorithm, designed to predict an *output state* starting from an original *input state*. To achieve this, the ANN must be *trained*, so that it can “learn” to predict the output state from a given set of initial configurations. The simplest way of doing it, is the *supervised learning*: input-output values are fed to the network, then the ANN tentatively computes an output set for each input set, and the difference between the predicted and the original output (i.e. the error) is used to calculate penalties and rewards to the synaptic weights. If the convergence to a solution exists, after a number of iterations, the predicted output value gets very close to the original one: *the ANN has learned*.

The network stores the original data in some dedicated neurons, the so called *input neurons*, elaborates these data and sends the results to another group of neurons that are called *output neurons*. Thus, data flow from input to output neurons. There is another group of neurons, named *hidden neurons*, that help the ANN to elaborate the final output (see Fig. 1). Each neuron (or unit) performs a simple task: it becomes *active* if its input signal is higher than a defined threshold. If one of these units becomes active, it emits a signal to the other neurons; for this reason, each unit can be considered as a *filter* that increases or decreases the intensity of the received signal. The connections between neurons simulate the biological synapses and this is the reason why they are called *synaptic weights* or, more simply, *weights*.

The whole process can be written as follows:

$$n_i = f(x) \left(\sum_j w_{ij} n_j - \theta_i \right), \quad (1)$$

where n_i is the i -th neuron value, $f(x)$ is the activation function, w_{ij} is the weight between i -th and j -th neuron, θ_i is the neuron threshold, and x is the signal. For the activation function we adopt a *sigmoid*

$$f(x) = \frac{1}{1 + e^{-\beta x}}, \quad (2)$$

where β is a parameter fixing the sigmoid slope. It is worth recalling that other expressions for the activation function can be used.

In general the ability of the network to reproduce the data increases with the number of the hidden units up to

a certain limit otherwise the ANN gets too “stiff” so that predicting the original output values gets more difficult. *The ANN stops learning*. Another reason to avoid many hidden neurons is that the procedure becomes time consuming (one must remember that we are mainly dealing with matrix products). On the other side, an ANN with too few hidden units gets “dull” in reproducing the original data. Therefore, there is an optimal number of hidden neurons for each ANN.

There are several possible ANN architectures, depending on the task the network is designed to perform. The already cited back-propagation algorithm (Rumelhart et al. 1986) is among the most versatile methods, it is one of the most suited to the supervised learning stage and can be applied to a wide range of problems. The name stems from the fact that the error produced by the network is propagated back to its connections to change the synapsis weights and consequently to reduce the error. As a detailed description of the ANN technique is beyond the aims of this paper, we limit ourselves to describe in some detail only the particular algorithm we have adopted for our purposes.

4. MaNN: aims and architecture

MaNN is the network we built to reproduce the ISM output models calculated by ROBO. We use a layer of input neurons where each input value corresponds to a free parameter of the ROBO models, i.e. the ones shortly described in Sect. 2. The definition of the model in MaNN requires some clarifications. While ROBO calculates the whole temporal evolution of a model, MaNN needs only the set of initial parameters (the input model) and the set of final results (the output model). Therefore, a MaNN model is made of two strings (vectors) of quantities: the initial conditions for the key parameters (the input vector) and the results for the quantities describing the final state, after that the assigned time interval has elapsed (the output vector). We refer to the input vector as the vector $\bar{x}$ whose dimension, i.e. the number of input units, is n_x . We then choose the shape of the output layer of neurons having in mind an “optimal” number of parameters to be used in a typical dynamical simulation. We use $\bar{y}$ to indicate these units and the number of the neurons belonging to this layer is n_y ². For example, if the code generating the input data only tracks the abundance of neutral hydrogen and the temperature, the MaNN input vector will be $\bar{x} = \{n_{\text{H}}^t, T^t\}$ and the relative output vector will be $\bar{y} = \{n_{\text{H}}^{t+\Delta t}, T^{t+\Delta t}\}$, and consequently $n_x = n_y = 2$. In this example the number of inputs is equal to the number of outputs, but this is not a strict requirement. In general,

² In practice, to chose the input and output vectors and their dimensions we pay attention to the characteristics of the code to which the MaNN is applied, namely the Padova Tree-SPH code EvOL by Merlin et al. (2010). The same reasoning can be applied to the code generating the input data, ROBO in our case. However, the whole procedure can be generalized to any other type of codes.

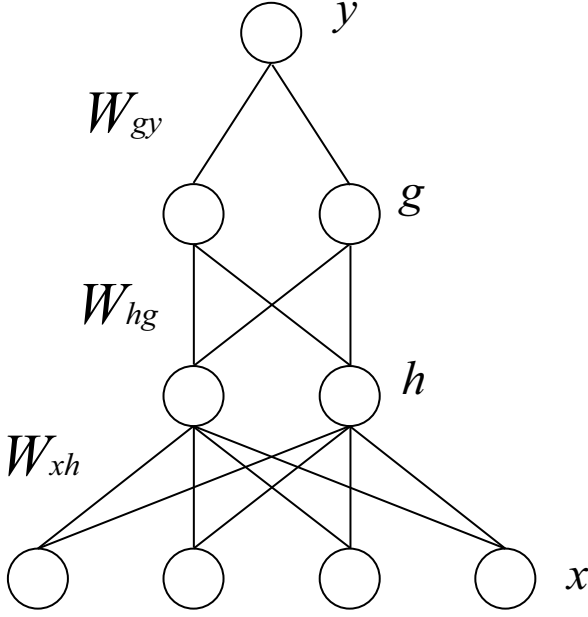


Fig. 1. The architecture of MaNN. Circles indicate neurons, lines are connections (i.e. weights). Input units are labeled with the symbol $\bar{x}$, hidden neurons with $\bar{h}$ if they belong to the first layer, and $\bar{g}$ if they belong to the second one; finally, output units are indicated with $\bar{y}$. The matrix W_{xh} represents the weights between input and the first hidden units, W_{hg} are the connections between the first hidden units and the second hidden layer and W_{gy} connects the second layer of hidden units and the output. A single output is shown. In this figure the forward propagation is from lower neurons to the upper ones. Back-propagation is from $\bar{y}$ to $\bar{x}$ neurons. Here $n_x = 4$, $n_h = n_g = 2$ and $n_y = 1$.

to avoid degeneracies one should have $n_x \geq n_y$ (considering independent database parameters). The input parameters can be also different from the output parameters, so in the previous example it could be $\bar{y} = \{n_H^{t+\Delta t}, n_{H_2}^{t+\Delta t}\}$.

There are two more layers in the MaNN, both are hidden and generally with the same size, which in turn depends on the complexity of the task (for example, retrieving a single output value can be obtained with two hidden layers of five elements, or even less). The first layer is $\bar{h}$ with n_h neurons and $\bar{g}$ is the second one with n_g units. Each layer is connected with a matrix of weights. A $n_x \times n_h$ matrix named W_{xh} is situated between the input units and the first hidden layer. Similarly, there is a $n_h \times n_g$ matrix named W_{hg} and, finally, W_{gy} , which is a $n_g \times n_y$ matrix. The architecture of MaNN that we have described here is sketched in Fig. 1.

4.1. The learning stage

The whole learning process is divided into two different steps; the first one is the so called *training stage*, where MaNN is trained with a randomly chosen set of data. In

the subsequent *test stage* new, blank (i.e. never used before) data are presented to MaNN to verify its behavior and the quality of the training. If MaNN retrieves the new data with a small (user defined) error, than the training stage is terminated. In case the error is still large, a new training stage is required. The training stage can also be stopped if the absolute error starts to increase again after an initial decrease (*early-stopping*). The aim is to avoid the so called *over-training*: a network is over-trained when it adapts too strictly to the data fed during the supervised learning, so that it cannot generalize the prediction to reproduce new cases of the test set of data, which contains data patterns that were not present in the previous stage.

At the beginning of the training process, each weight is initialized to a small, randomly chosen, value in the range $[-0.1, 0.1]$. The weights change under the action of MaNN. The rate at which MaNN changes its weights, otherwise known as the learning rate η , must be chosen carefully: for too high a value MaNN could oscillate without reaching a stable configuration, whereas for too small a value MaNN could take a long time to reach the desired solution or, even worse, be trapped in a false minimum of the error hyper-surface. We usually choose $0.1 \leq \eta \leq 0.9$.

The matrix of initial parameters contains N rows (the initial configurations of the input models); we randomly pick from this set $N_L < N$ rows for the training stage, which we refer to as L , and $N_T = N - N_L$ rows for the test stage, T .

A set $\bar{s}$ of initial parameters randomly picked up from N_L is fed to the network as input vector $\bar{x} = \bar{s}$. The activation of each node belonging to the first hidden layer is then computed as

$$\bar{h} = f(\bar{x} \otimes W_{xh}) , \quad (3)$$

where $\bar{h}$ is the vector of hidden neurons, f is the sigmoid function, W_{xh} is the weight matrix between the input and the first hidden neurons, and $\otimes$ indicates the matrix product. If $\bar{c} = \bar{a} \otimes B$, where $\bar{a}$ is a vector of n elements and B is a matrix $n \times m$, the size of the vector $\bar{c}$ is m . The sigmoid function $f(\bar{v})$ has $\beta = 1$. Then the following relation holds

$$\bar{g} = f(\bar{h} \otimes W_{hg}) , \quad (4)$$

with obvious meaning of the symbols. Finally, the output vector of MaNN is given by

$$\bar{y} = f(\bar{g} \otimes W_{gy}) . \quad (5)$$

This first step is named the *forward-propagation*, because the algorithm calculates the output of the network for a given input vector $\bar{x} = \bar{s}$.

During the very first forward-propagation the output vector will contain nearly random values, because the weights are randomly generated. To improve upon it, we compute the error and back-propagate it to change the weights accordingly. The error at the output layer is

$$\bar{\delta}_y = (\bar{t} - \bar{y}) \dot{f}(\bar{A}_y) = (\bar{t} - \bar{y}) \bar{y}(1 - \bar{y}) , \quad (6)$$

where $\bar{A}_y$ is the vector of the activation functions for the output layer and the last equality is valid only with the activation function $f(\bar{v})$ with $\beta = 1$ that we have adopted. The error $\bar{\delta}_y$ is propagated back to the second hidden layer to become

$$\bar{\delta}_g = \dot{f}(\bar{A}_g) [\bar{\delta}_y \otimes W_{gy}] = \bar{g}(1 - \bar{g}) [\bar{\delta}_y \otimes W_{gy}], \quad (7)$$

with obvious meaning of all the symbols and expressions. Finally the error is propagated back for the last time giving

$$\bar{\delta}_h = \dot{f}(\bar{A}_h) [\bar{\delta}_g \otimes W_{hg}] = \bar{h}(1 - \bar{h}) [\bar{\delta}_g \otimes W_{hg}]. \quad (8)$$

Now that the error for each step between the various layers has been computed, the changes of the synaptic weights are

$$\Delta W_{xh} = \eta [\bar{x} \otimes \bar{\delta}_h] + \alpha \Delta W_{xh}^o,$$

$$\Delta W_{hg} = \eta [\bar{h} \otimes \bar{\delta}_g] + \alpha \Delta W_{hg}^o,$$

$$\Delta W_{gy} = \eta [\bar{g} \otimes \bar{\delta}_y] + \alpha \Delta W_{gy}^o,$$

(9)

The last terms at the r.h.s. of the above equations is called the *momentum* made by the product of a parameter α (to be tuned) and the variation of the corresponding weights at the previous step. ΔW_{xh}^o and the other similar terms indicate the values of the weights at the previous training step. The *momentum* is introduced to force the procedure to avoid false minima in the error hyper-surface. This loop must be repeated for each combination of the parameters in the set L .

To test the efficiency of the MaNN learning, the forward propagation is then applied to all the variables belonging to the set L . The RMS of the test data set is

$$\text{RMS} = \sqrt{\frac{\sum_{i=1}^{N_T} (y_i - t_i)^2}{N_T - 1}}, \quad (10)$$

where y_i is the output for the i -th test, t_i is the i -th correct test pattern and consequentially $y_i - t_i$ is the MaNN's error.

The training test loop will be repeated until the MaNN's learning reaches one of the following cases: (i) the RMS is lower than a certain limit chosen by the user, (ii) the RMS does not change for a sufficient number of loop cycles or, finally, (iii) the early-stopping condition is verified.

5. Training and testing MaNN

5.1. Input data from Robo

As already stated, with ROBO we have calculated approximately 55000 models. For a given set of initial conditions and physical parameters, each model describes the state of an ISM element after $\Delta t = 10^4$ years of evolution. This time interval is short enough to secure that only small changes in gas abundances have occurred (a sort of picture of the physical state at given time), and long enough

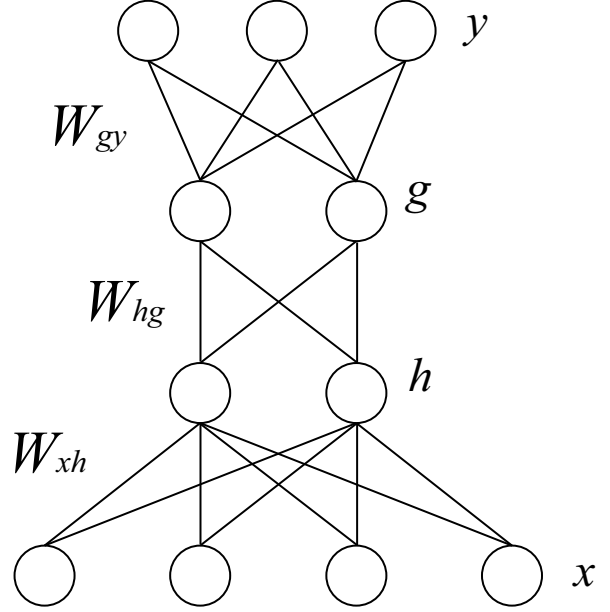


Fig. 2. The architecture of a double hidden layer ANN. Circles indicate neurons, while lines represent the connections between the layers (i.e. weights). The input units are labeled with the symbol $\bar{x}$, the hidden neurons with $\bar{h}$ if they belong to the first layer, and with $\bar{g}$ if they belong to the second one; finally, the output units are indicated with $\bar{y}$. The matrix W_{xh} represents the weights between the output and the first hidden units, W_{hg} are the connections between the first hidden units and the second one and W_{gy} connects the second layer of hidden units and the output. Multiple outputs are shown. In this figure the forward propagation is from the lower neurons to the upper ones. The back-propagation is from $\bar{y}$ to $\bar{x}$ neurons. Here $n_x = 4$, $n_h = n_g = 2$ and $n_y = 3$.

to be compatible with the typical time step in large-scale dynamical simulations.

For the purposes of the present analysis, we limit ourselves to track the following parameters: temperature T , metallicity $\bar{Z} = 10^{[\text{Fe}/\text{H}]}$ ³, and number densities of H, H₂, H⁺ and e⁻. This means that in addition to the temperature, MaNN is designed here to follow just these four species, whereas ROBO can actually follow twenty-three species including atoms, molecules and free electrons. The main motivation for this limitation is that in a large numerical dynamical simulation it is more convenient in terms of computing time and workload to follow in detail only those species (and their abundances) that are known to play a key role in the real evolution of the ISM

³ We call the attention of the reader on the definition of metallicity that we have adopted. It is related only to the iron content in the standard spectroscopic notation and not to the classical sum of the by mass abundances of all the species heavier than He, shortly indicated with Z and satisfying the relation $X + Y + Z = 1$, where X and Y are the by mass abundances of hydrogen and helium respectively.

(with particular attention to star formation and energy feedback) and to consider all the other species to remain nearly constant and small. In brief, the heavy elements are important coolants and the same holds for the molecular hydrogen, at least below $\sim 10^4$ K. The ionization fraction of hydrogen is also important, because free electrons are involved in a number of chemical reactions, and are the colliders that excite the coolants: this is the reason why n_{H^+} and n_{e^-} are considered. Moreover, the neutral n_{H} is in the list to ensure the mass conservation while changing the fractional abundances. Finally, the temperature determines the kind of physical processes that are important.

The initial values of the number densities. The initial values of the number densities fall in three groups: (i) the elemental species with constant initial values, the same for all the models (namely H^- , H_2^+ , He^+ and He^{++}); (ii) the elemental species whose initial values are derived from other parameters (namely He, all the deuteroids and the metals, like for instance C and O, that depend on the choice of the total metallicity $\tilde{Z}$) and, finally, (iii) the elemental species with free initial conditions (namely H, H_2 , H^+ , and e^-).

HYDROGEN GROUP: H, H^+ , H^- , H_2 and H_2^+ . The initial values of the species H^- and H_2^+ are $n_{\text{H}^-} = n_{\text{H}_2^+} = 10^{-10} \text{cm}^{-3}$ Prieto et al. (2008), while the other three hydrogen species have free initial values.

DEUTERIUM GROUP: D, D^+ , D^- , D_2 , HD and HD^+ . The numerical densities of the deuteroids are calculated from their hydrogenoid counterparts. For the single atom species we have $n_{\text{D}} = f_{\text{D}} n_{\text{H}}$, $n_{\text{D}^+} = f_{\text{D}} n_{\text{H}^+}$ and $n_{\text{D}^-} = f_{\text{D}} n_{\text{H}^-}$, where $f_{\text{D}} = n_{\text{D}}/n_{\text{H}}$. For the molecules, we can consider the ratio f_{D} as the probability of finding an atom of deuterium in a population of hydrogen-deuterium atoms. This assumption allows us to calculate the HD, D_2 and HD^+ number densities as a joint probability; for HD and HD^+ we have $n_{\text{HD}} = f_{\text{D}} n_{\text{H}_2}$ and $n_{\text{HD}^+} = f_{\text{D}} n_{\text{H}_2^+}$, but for D_2 is $n_{\text{D}_2} = f_{\text{D}}^2 n_{\text{H}_2}$ as the probability of finding two deuterium atoms is f_{D}^2 . This is valid as long as $f_{\text{D}} \ll 1$.

HELIUM GROUP: He, He^+ , He^{++} . The ratio $n_{\text{He}}/n_{\text{H}}$ is set to 0.08, thus allowing to the initial value of n_{He} to vary accordingly to the initial value for n_{H} . The initial number densities of the species He^+ , He^{++} are both set equal zero and kept constant in all the models.

METALS GROUP: C, C^+ , O, O^+ , Si, Si^+ and Fe, Fe^+ . The Fe number density of the ISM is

$$n_{\text{Fe}} = n_{\text{H}} \cdot \text{dex} \left\{ [\text{Fe}/\text{H}] + \log \left(\frac{n_{\text{Fe}}}{n_{\text{H}}} \right)_{\odot} \right\}, \quad (11)$$

where $(n_{\text{Fe}}/n_{\text{H}})_{\odot}$ is the iron-hydrogen ratio for the Sun. To retrieve the number density of a given metal X we use $n_{\text{X}} = n_{\text{Fe}} \cdot f_{\text{X}}$, where f_{X} is the metal-iron number density ratio in the Sun.

The list of the species whose initial number densities are kept constant in all the models of the ISM is given in Table 1. The chemical composition of the ISM is typically primordial with the by mass abundances of hydrogen $X = 0.76$, and helium $Y = 0.24$ and all metals $Z \simeq 0$. The

Table 1. Initial values for the number densities of the hydrogen and helium elemental species and the deuteroids. They are either fixed to a constant value or based upon the numerical abundance of one of the *free* hydrogen species n_{H} , n_{H_2} and n_{H^+} via the f_{D} factor. Since H^- and H_2^+ are constant, then also D^- and HD^+ are fixed. Values are indicated as $a(b) = a \times 10^b$. See also the text for details.

H^-	1.0(−12)	D	$f_{\text{D}} n_{\text{H}}$
H_2^+	1.0(−12)	D^+	$f_{\text{D}} n_{\text{H}^+}$
He	0.8(−1) n_{H}	D^-	$f_{\text{D}} n_{\text{H}^-}$
He^+	0.0(+0)	HD	$f_{\text{D}} n_{\text{H}_2}$
He^{++}	0.0(+0)	HD^+	$f_{\text{D}} n_{\text{H}_2^+}$
		D_2	$f_{\text{D}}^2 n_{\text{H}_2}$

helium to hydrogen number density ratio corresponding to this primordial by mass abundances is $n_{\text{He}}/n_{\text{H}} \simeq 0.08$. The adopted cosmological ratio for the deuterium is $f_{\text{D}} = n_{\text{D}}/n_{\text{H}} \simeq 10^{-5}$. With the aid of these numbers and the above prescriptions we get the number density ratios listed in Table 5.1 and the initial values of the number density in turn.

Domains of the physical variables. In the present version of MaNN the variables in question span the following ranges: $\tilde{Z} = [10^{-12}, 10]$ for the metallicity, $n_{\text{H}} = n_{\text{H}^+} = n_{\text{e}^-} = n_{\text{H}_2} = [10^{-12}, 10^3] \text{cm}^{-3}$ for the numerical densities, and $T = [10, 10^7] \text{K}$ for the temperature.

Role of the dust. In this study, we neglect the presence of dust: in other words, the *dust density* is kept constantly equal to zero. This is because the current version of the NB-TSPH code EvOL does not include the treatment of the dust component of the ISM. We plan to include silicates and carbonaceous dust grains both in EvOL and MaNN.

Other parameters. In addition to this, we do not consider the contribution by the *Cosmic Rays field*, the *background heating*, and the *UV ionizing field*. All of these processes, however, can be described by ROBO (see Grassi et al. 2010, for more details).

The CMB temperature is set to 2.725 K, the observed present value (Fixsen 2009) and is kept constant in all the models. In the case of a cosmological use of MaNN one should include a redshift-dependent variation of the CMB temperature for a correct description of the CMB effects.

The parameters we have chosen form a $\mathbb{R}^6$ space: the input space of the MaNN. The large number of input models secures that the space of physical parameters is smoothly mapped.

Finally, in addition to the six-dimensional input vector, another input unit is dedicated to the so called *bias* which is always set to $x_0 = -1$; it gives the threshold θ in the eqn. (1).

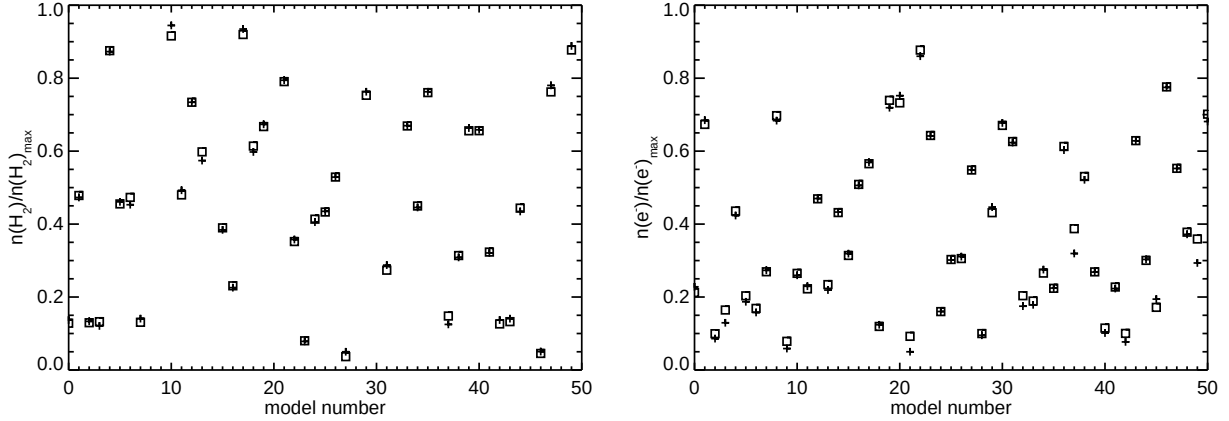


Fig. 3. Comparison between normalized data from MaNN (cross) and ROBO (square), for molecular hydrogen density (left panel) and free electrons (right panel). We show only 50 randomly picked models. Similar results for the other outputs.

5.2. Data normalization

All the input data are normalized. Given a generic input model contained in the vector $\bar{s}$ of dimensions n_s , first we take the logarithm (base 10) of its components (all of which are positive quantities) and replace it with a new vector $\bar{s}'$ so defined

$$\bar{s}' = \log(\bar{s} + \bar{\epsilon})$$

where $\bar{\epsilon}$ is a vector of the same length as $\bar{s}$ made of constant, small positive quantities here introduced to avoid infinities in the logarithms (or zeros in the original quantities). In all the cases, all the components of $\bar{\epsilon}$ are $\epsilon = 10^{-12}$. Then we introduce the matrix L containing all the $\bar{s}'$ vectors. The rows of the matrix L are input models, one row for each of them, whereas the columns of L are the n_s elements of the $\bar{s}'$. Finally we calculate the vector $\bar{x}$

$$\bar{x} = \frac{(\bar{s}' - \langle \bar{s}' \rangle) - \text{MIN}[L]}{\text{MAX}[L] - \text{MIN}[L]} (n_+ - n_-) + n_-, \quad (12)$$

where MAX and MIN are functions that retrieve the $\bar{s}$ -sized maximum and minimum value from their argument, $\langle \bar{s}' \rangle$ is a vector with size equal to the number of input parameters. The last term is introduced to have a zero-mean sample of data. the mean of $\bar{s}'$. The vector $\bar{x}$ is the so-called input pattern of MaNN. The quantities $n_+ = 0.95$ and $n_- = 0.05$ are the upper and the lower bounds of the normalization interval. In other words, for each component of the vector $\bar{s}'$ we have searched in the corresponding columns of the matrix L the minimum and maximum values with which we have normalized the $\bar{s}'$ element.

To avoid notation misunderstanding, we recall that $s_i \in \bar{s} \subset L$ and $x_i \in \bar{x}$ with x_i the i -th neuron belonging to the input layer $\bar{x}$. When a pattern is used as input, we have $\bar{x} \equiv \bar{s}'$.

Similar procedure is adopted for the output values of MaNN. Let $\bar{t}$ be the vector of the original output models,

$\bar{t}'$ the vector $\bar{t}' = \log(\bar{t} + \bar{\epsilon})$ and T the matrix containing all the $\bar{t}'$ vectors (the analog of L): the vector of normalized output values $\bar{t}_n$ is

$$\bar{t}_n = \frac{(\bar{t}' - \langle \bar{t}' \rangle) - \text{MIN}[T]}{\text{MAX}[T] - \text{MIN}[T]} (n_+ - n_-) + n_-, \quad (13)$$

with the same meaning of eqn. (12). The notation here is $t_i \in \bar{t} \subset T$.

5.3. Compression of data?

To reduce the complexity of the problem, one can try to compress the input data, i.e. to reduce the number of variables and to obtain a new initial set easy-to-handle. A popular technique to reduce the number of dimensions of a data sample is the *principal component analysis* (PCA). The PCA is a transformation between the original data space and a new space (named PCA space). The aim of this method is to build this $\mathbb{R}^n$ -dimensional subspace (with n less than the number of dimensions of the original space) so that its new PCA-variables are uncorrelated. In this way, only the most statistically significant parameters (i.e., the new variables that have the highest percentage of the total variance) are taken into consideration, instead of using the original parameters. PCA is useful when the data set is reduced to two or three modes. This is because two matrix products are needed, one to transform the input from the physical parameters space to the PCA space, and one to transform the output from the PCA space to the physical parameters space. These two operations have a non-negligible computational cost. In general, compressing the data with the PCA may improve the learning process of MaNN and its ability in reproducing the original data.

Analyzing the issue, we soon found that in our case the PCA method is not useful, and using the original data yields better results. Table 2 contains the results of the

Table 2. Results of the PCA analysis on the output data.

mode	Eigenvalue	Variance (%)
1	0.14585	36.9657
2	0.07454	18.8922
3	0.07006	17.7580
4	0.05606	14.2073
5	0.02871	7.2760
6	0.01934	4.9008

PCA analysis. It is easy to understand why reducing the number of parameters does not improve the situation: five modes cover approximately 87% of the total variance, and with the sixth mode the coverage reaches 94%. This means that the reduction of the number of parameters to obtain a $\mathbb{R}^2$ or $\mathbb{R}^3$ space is not possible. Note that only the PCA on the output data is shown, since the input data are random and they generate six modes with the same variance.

5.4. Single or multiple ANNs for output configurations

There are different ways in which the ANN algorithm can be used in practice to extract the results that we need from a given initial set of data. Once given an input pattern one may use a single ANN with multiple outputs (one for each quantity to evaluate). This situation is illustrated in Fig. 2. We will refer to it as case A or single ANN. Alternatively, given the same input pattern, one may want to set up a manifold of ANNs, each of which with a single output. This situation for each ANN is corresponding to the one shown in Fig. 1. We will refer to it as case B and the whole algorithm is named multiple ANN (from which the acronym MaNN we have already introduced). Note that this difference would be irrelevant if the MaNN does not include hidden layers, because in that case the outputs would be uncorrelated. However, when one or more layers are added, the scenario changes. As Eqn. (5) shows, the state of an output neuron depends on the weights that connect it to the previous layer. If an hidden layer is present, then all the output neurons rely on every hidden neurons (albeit with different weights). These neurons are at the same time linked with a weight matrix to the input layer (or with another hidden layer). In this way, two output neurons share some weights. This is crucial when the back-propagation error is applied, because different output neurons interfere modifying the same weight (i.e. the shared one).

There is another difference worth being pointed out. Input and output data patterns are not physically uncorrelated because the latter results from the former via the ROBO model. The architecture of case A by somehow forces an artificial correlation between input and output data patterns other than the natural one established by the underlying ISM physics. This may originate from the back-propagation procedure simultaneously changing all the weights. Alternatively, case B does not contain this

undesired effect. The input pattern separately determines each output value.

As a matter of fact, using the case A, during the learning phase the way the error decreased was not satisfactory. So we put case A aside and turned to case B. The set up of MaNN includes six different networks, one for each output parameter. It also contains a double hidden layer network because other schemes (single hidden layer or no hidden layers at all) did not give good results.

5.5. Training results

We apply MaNN to our database of ISM models obtaining different performances for different network parameters. The best cross-correlation results are obtained for $n_h = 20$ and $n_g = 200$. Using a smaller number of hidden neurons yields unsatisfactory results, as well as with $n_h > n_g$. Bad results are obtained when the RMS decrease is slow, when the error is high ($> 10^{-3}$) just before applying the early-stopping technique, or, finally, when the RMS curve appears to be noisy (huge RMS oscillations). The learning rate has been set to $\eta = 0.5$, which appears to be the best compromise between learning speed and accuracy.

Results obtained with the choice of these parameters are shown in Fig. 3, with crosses indicating the data retrieved with MaNN and squares used to show the values taken from the database produced with ROBO. The two panels show molecular hydrogen and free electrons abundances, both normalized following eqn. (13). We only show 50 randomly picked models. The other outputs (n_H and n_{H^+}) are not displayed here since they have a similar aspect.

As the figure shows, the error on the test set is relatively low. MaNN retrieves the data produced with ROBO fairly well. It is therefore possible to use it in the context of complex dynamical simulations.

6. MaNN interfaced with NB-TSPH simulations

To evaluate the performance of MaNN in real large-scale numerical simulations we implemented it in the NB-TSPH code EvOL, and run a “realistic” version of the ideal Sedov-Taylor problem, which is known to be a rather demanding test.

MaNN tracks the evolution of the relative abundances of H, H^+ , H_2 and e^- within each gas particle of the SPH simulation. The MaNN routine is called by each active gas particle during the computation of its radiative cooling (which is in turn obtained from look-up tables, see Merlin & Chiosi 2007, for details). Since the time interval $\Delta t = 10^4$ yr in between the initial and the final stage of any model in MaNN is usually smaller than the typical dynamical time step of a gas particle, the routine is called several times until the dynamical time step is covered. If the dynamical time step is not an integer multiple of the MaNN time interval, an interpolation is applied to the results of the last two calls of MaNN. During the whole

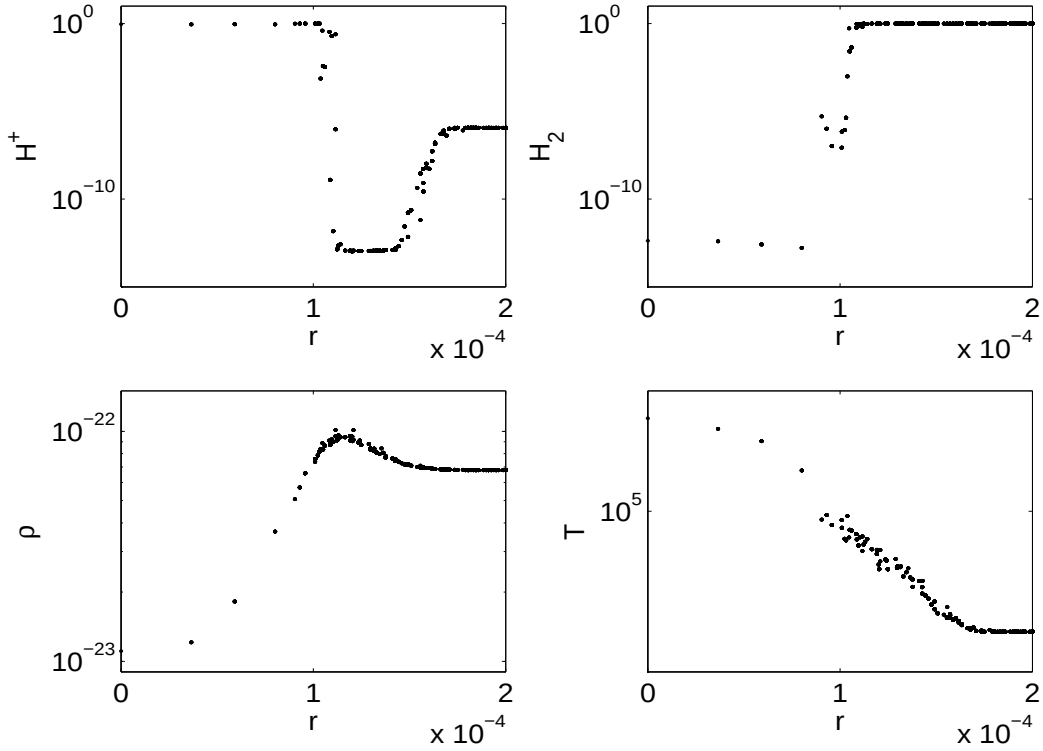


Fig. 4. Sedov-Taylor OB-wind test with chemical evolution. Left to right, top to bottom: radial profiles of H^+ and H_2 mass fractions, ρ , and T , at $t = 0.5$ Myr. Plotted are all particles within the central 200 pc.

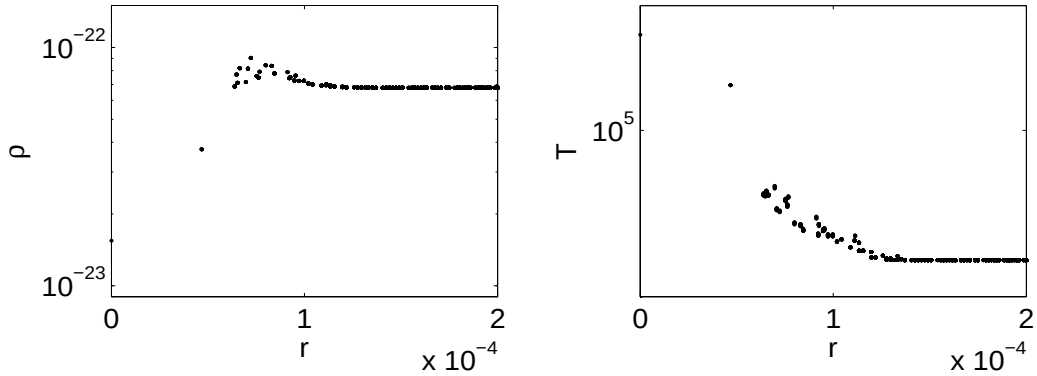


Fig. 5. Sedov-taylor OB-wind test without chemical evolution. Left to right: radial profiles of ρ and T , at $t = 0.5$ Myr. Plotted are all particles within the central 200 pc.

procedure, all the physical parameters but the afore mentioned relative chemical abundances and the temperature (namely, total density, metallicity and mechanical heating rate) are kept constant.

Due to the detailed description of the chemical properties of each particle, the adiabatic index γ in the equation of state governing the gas pressure $P = (\gamma - 1)\rho u$ varies as a function of the relative abundances (see Grassi et al. (2010)). This will immediately reflect on the relationship between the internal energy and the temperature, and on the equation of the motion itself. Therefore, we expect an important change in the dynamical evolution of the whole

system as compared to the case in which these chemical effects are ignored.

The test run is set up as follows. A uniform grid of 45^3 equal mass gas particles is displaced inside a periodic box of size $l = 1$ kpc. The mass of each mass element is $\simeq 10^4 M_\odot$, so the initial uniform density is $\rho \simeq 6.78 \times 10^{-23} \text{ g/cm}^3$. The initial temperature is 200 K. The initial chemical mass fractions of each gaseous element are $[H^+] = 10^{-20}$, $[H_2] = 0.99$, $H = 1 - ([H_2] + [H^+])$. These initial conditions resemble a dense interstellar cloud of cold molecular material.

At time $t = 0$, the temperature of the central particle is instantaneously raised up to $T = 2 \times 10^7$ K, a tem-

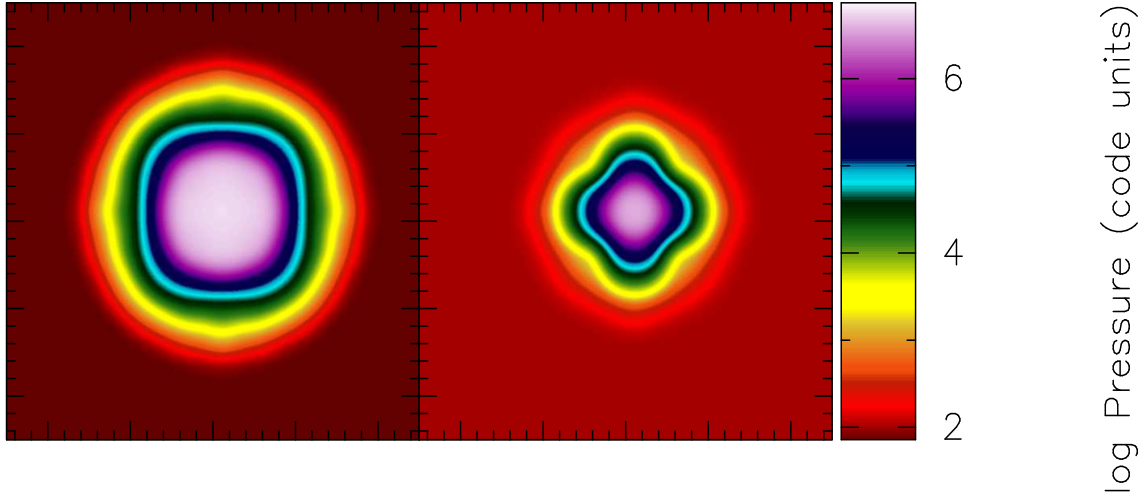


Fig. 6. Sedov-taylor OB-wind test. Pressure field in a central slice of side $l = 500$ pc, with (left) and without (right) chemical evolution.

perature similar to that of the gas shocked by the winds ejected by O-B young stars. We name this physical case as the *Sedov-Taylor OB-wind*. At the same time, the chemical composition of the particle is changed to $[H_2] = 10^{-20}$, $[H^+] = 0.99$, $H = 1 - ([H_2] + [H^+])$. The temperature of the central particle is maintained constant, to mimic the continuous energy injection by the central star, and the system is let evolve freely under the action of hydrodynamic interactions and chemical reactions. We also include thermal conduction and radiative cooling that are usually neglected in standard Sedov-Taylor explosion tests (see Merlin et al. 2010, for more details).

A region of hot material expands in the surroundings of the central particle. This is due to a spurious numerical conduction which, however, causes a numerical smoothing of the initial condition and favors the correct developing of the subsequent evolutionary stages. Then, a shock wave forms and moves outwards radially. Note that the continuous energy release from the central particle and the effect of radiative cooling, almost prevent the inward motion of shocked particles like in standard tests of the Sedov-Taylor explosion.

In Fig. 4 we show the state of the system after $\simeq 0.5$ Myr of evolution. The bottom panels display the density and temperature profiles, and the top panels the abundances of H^+ and H_2 . In the central hot and less dense region, the molecular hydrogen gets completely ionized (the abundance of H_2 falls to very low values), due to the high temperatures and low densities. The less dense cavity is surrounded by a layer of over-dense material (the shock layer) which in turn is surrounded by another layer with the initial density. The over-dense front is moving outwards. In the pre-shock region, near the shock layer, the density increase causes the H^+ to recombine into neutral hydrogen. These results fairly agree with the theoretical predictions.

To strengthen the above conclusion, we performed another simulation of the Sedov-Taylor OB-wind, in which the effects of the chemical composition and its changes on the gas particles are ignored (by the way this is exactly what is assumed in most if not all the cosmological simulations). In practice, all the gas particles are assumed to be composed of solely neutral H, so that the adiabatic index γ is fixed and equal to $5/3$. The results are shown in Fig. 5, in which the density and temperature profiles are displayed at the same age of the previous case presented in Fig. 4. Clearly, the dynamical evolution of the system is substantially different. The over-dense front moves at significantly lower speed. In particular, it is clear how the variable chemical composition ensures a higher pressure in the central region containing ionized particles, with respect to the outer region containing molecules. This causes the faster expansion of the central bubble with respect to present case. This is also supported by the pressure fields in the two cases shown in Fig. 6. The left panel shows the pressure field of the first case with evolving chemical composition of the gas particles, whereas the right panel shows the same but for constant chemical composition. The color code shows the value of the pressure in code units. Both pressure fields are measured at the same age of 0.5 Myr and on the surface of an arbitrary plane passing through the center of the computational box. There are a number of interesting features to note: (i) The pressure in the outskirts is lower in the case with chemical evolution, where the adiabatic index is $\gamma = 7/5$ for the bi-atomic molecule H_2 . At the same time, the pressure in the central region of the case with chemical evolution is higher by nearly a factor of two because the number of free particles (or equivalently the internal energy corresponding to a given temperature) is nearly doubled. (ii) The continuous energy release from the central particle, plus the action of the radiative cooling, almost prevent the inward motion of shocked particles (conversely to what happens in the

standard Sedov-Taylor explosion test). Instead, they are compressed in a thin layer, which moves outwards under the action of the central source of pressure. For this reason, we are interested in the state of the system shortly after the beginning of the energy release, before the shock layer is pushed too far away from the central region. However, at that moment only a relatively small number of particles have already felt a strong hydrodynamical interaction. This explains the geometrical features that can be seen in Figs. 6. They are due to the initial regular displacement of the particles, whose effects are still to be smeared out by hydrodynamics interactions. Because the detailed chemical treatment implies a larger pressure gradient and, consequently, a faster expansion of the central bubble, after a given Δt the simulation without the inclusion of MaNN shows a stronger geometrical spurious asymmetry with respect to the “standard” case: this is simply because it is at a more primordial stage of its evolution.

7. Summary and conclusions

We presented MaNN, an artificial neural network aimed at providing a light and versatile interface between databases of models of the ISM and numerical hydrodynamical codes. In our specific case the ISM models are the outputs of the ROBO code developed by Grassi et al. (2010) and the Padova NB-TSPH code EVOl developed by Merlin et al. (2010).

With the aid of ROBO we have calculated a large database of ISM models (55000 in total). In these models, given a set of initial conditions, the thermodynamical and chemical evolution of the ISM is followed during a given time interval to get the new physical state of the ISM. A large volume of the hyper-space of the physical parameters and their initial conditions is explored to get an ensemble of ROBO models. For the aims of this first study, the ISM models are calculated switching off the dust, the external heating, and the cosmological evolution. They can be restored at any time.

The above database is fed to MaNN with some simplifications. First of all, a MaNN model is made of two strings (vectors) of quantities: the initial condition for the key parameters (input vector) and the results for the quantities describing the final state after a certain time interval of evolution (output vector). Not necessarily they must be identical in dimension and type of physical quantities. The first step to undertake is to train MaNN to reproduce the results obtained with ROBO. The technique in use is the back-propagation algorithm. Once the training stage is completed and the architecture of MaNN is fixed, MaNN is able to mimic the results of ROBO with a great degree of accuracy. The great advantage of all of this is that in large cosmological or galactic simulations the thermodynamical and chemical properties of all the gas particles can be determined at each time step with great accuracy in practice at no computational cost.

Finally, we have implemented MaNN into EVOl and run the Sedov-Taylor OB-wind test, i.e. the evolution of

a dense and cold molecular cloud during the release of a high amount of energy in its central region. The results are fairly good, showing a nice agreement with the theoretical expectations: while a shock wave develops and moves outwards, a hot and ionized region forms in the central region. Taking into account the detailed chemical evolution greatly affects the thermo-dynamical evolution of the system. Similar effects should be expected in realistic NB-TSPH simulations of cosmological or galactic large scale structures.

In conclusion, MaNN proves to be a practical, easy-to-handle and robust method to include the thermal and chemical properties of the ISM in NB-TSPH simulations.

Acknowledgements. T. Grassi is grateful to Dr. F. Combes for the kind hospitality at the Observatoire de Paris - LERMA under EARA grants where part of the work has been developed and the many stimulating discussions.

References

- Bakes, E. L. O. & Tielens, A. G. G. M. 1994, *ApJ*, 427, 822
- Carpenter, G. A. & Grossberg, S. 1987, *Appl. Opt.*, 26, 4919
- Carpenter, G. A. & Grossberg, S. 1987, *Comput. Vision Graph. Image Process.*, 37, 54
- Carpenter, G. A. & Grossberg, S. 1990, *Neural Networks*, 3, 129
- Cazaux, S. & Spaans, M. 2009, *A&A*, 496, 365
- Cen, R. 1992, *ApJS*, 78, 341
- Draine, B. T. & Salpeter, E. E. 1979, *ApJ*, 231, 77
- Dwek, E. 1998, *ApJ*, 501, 643
- Fixsen, D. J. 2009, *ApJ*, 707, 916
- Galli, D. & Palla, F. 1998, *A&A*, 335, 403
- Glover, S. C. O. & Jappsen, A. 2007, *ApJ*, 666, 1
- Grassi, T., Krstić, P., Merlin, E., Buonomo, U., Piován, L., & Chiosi, C. 2010, *ArXiv e-prints* 1012.1142
- Hebb, D. O. 1949, *The Organization of Behaviour: a neuropsychological theory* (New York: Wiley and Sons)
- Hollenbach, D. & McKee, C. F. 1989, *ApJ*, 342, 306
- Hopfield, J. J. 1982, *Proceedings of the National Academy of Science*, 79, 2554
- . 1984, *Proceedings of the National Academy of Science*, 81, 3088
- Lipovka, A., Núñez-López, R., & Avila-Reese, V. 2005, *MNRAS*, 361, 850
- Maio, U., Dolag, K., Ciardi, B., & Tornatore, L. 2007, *MNRAS*, 379, 963
- McClelland, J. L., Rumelhart, D. E., & the PDP Research Group. 1986, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, Vol. vol. II (Cambridge, MA: MIT Press-Bradford Books)
- Merlin, E., Buonomo, U., Grassi, T., Piován, L., & Chiosi, C. 2010, *A&A*, 513, A36+
- Merlin, E. & Chiosi, C. 2007, *A&A*, 473, 733
- Prieto, J. P., Infante, L., & Jimenez, R. 2008, *ArXiv e-prints*

- Rosenblatt, F. 1962, Principles of neurodynamics; perceptrons and the theory of brain mechanisms (Washington: Spartan Books)
- Rumelhart, D., Hinton, G., & Williams, R. 1986, Nature, 323, 533
- Sutherland, R. S. & Dopita, M. A. 1993, ApJS, 88, 253
- Weingartner, J. C. & Draine, B. T. 2001, ApJS, 134, 263

Formation and Evolution of Early-Type Galaxies. III

Star formation history as a function of mass and over-density

Emiliano Merlin¹, Cesare Chiosi¹, Lorenzo Piovan¹, Tommaso Grassi¹, Umberto Buonomo¹, & Francesco La Barbera²

² INAF-Osservatorio Astronomico di Capodimonte, Salita Moiariello 16, Napoli, Italy

Received: ; Accepted:

ABSTRACT

Context. To date, a consistent and comprehensive theory of galaxy formation is still missing. Growing observational evidences claim for a population of massive, red galaxies at very high redshifts, apparently at odds with the theoretical concordance cosmology scenario, in which larger systems form later via merger of smaller subunits. How did the big, early red galaxies which we observe at high redshifts form? Did they assemble their stellar content in a strong, early burst of star formation activity? Is this scenario consistent with the theoretical cosmological background?

Aims. We investigate the influence of the initial proto-galaxies over-densities and masses on their evolution, to understand whether the internal properties of the proto-galactic halos are sufficient to account for the varied properties of the galactic populations.

Methods. By means of fully hydrodynamical N-body simulations performed with the code EVOL we produce twelve self-similar models of early-type galaxies following their evolution from the epoch of their detachment from the linear regime, i.e. $z \geq 20$, to $z \leq 1$ (we also produce some more ancillary models for further analysis). The simulations include radiative cooling, star formation, stellar energy feedback, a reionizing photoheating background, and chemical enrichment of the ISM. We do not consider the possible presence of Active Nuclei.

Results. We find a stunning correlation between the initial properties of the proto-halos and their star formation histories. Massive ($M_{tot} \simeq 10^{13} M_{\odot}$) halos experience a single, intense burst of star formation (with rates $\geq 10^3 M_{\odot}/\text{yr}$) at early epochs, consistently with observations, with a less pronounced dependence on the initial over-density; intermediate mass ($M_{tot} \simeq 10^{11} M_{\odot}$) halos histories strongly depend on their initial over-density, whereas small ($M_{tot} \simeq 10^9 M_{\odot}$) halos always have fragmented histories, resulting in multiple stellar populations, due to a “galactic breathing” phenomenon. The galaxy models have morphological, structural and photometric properties comparable to real galaxies, often closely matching the observed data; there are some incongruities which we interpret as a consequence of some numerical choices. The position of the models on diagnostic planes is generally consistent with observed galaxies.

Conclusions. We conclude that internal properties are essentially sufficient to explain many of the observed features of early type galaxies, particularly the complicated and different star formation histories shown by halos of very different mass. In this picture, *nature* seems to play a pivotal role, whereas the action of *nurture* is of secondary importance.

Key words. Galaxies, cosmology

1. Introduction

Understanding how early-type galaxies formed and evolved along the Hubble time is still a formidable challenge. According to the current view of the subject, galaxies probably began to form at $z \sim 20 - 50$, when Dark Matter (DM) gave origin to the first sufficiently deep gravitational potential wells (Tegmark et al. 1997; Gao et al. 2007a,b). However, it is not yet clear how the paradigm Λ -CDM cosmological model (which prescribes a hierar-

chical, bottom-top formation of structures, implying that massive systems assemble their mass later than the smaller ones) should be reconciled with the observational evidence for large, and red, galaxies already in place at very high redshifts (see e.g. Marchesini et al. 2010; Harrison et al. 2011a; Mortlock et al. 2011). The problem is twofold: on one hand, it must be explained how and when star formation is quenched in both massive and small haloes, which is necessary to reconcile the theoretical prediction with the observed galaxy mass function (see e.g. Bundy et al. 2006); on the other hand, it should be also clarified how such

massive systems can form at very high redshifts. Semi-analytical models of hierarchical formation (e.g., De Lucia & Blaizot 2007) have succeeded in explaining some of the observed features, but largely overestimate the number of faint objects, and fail at reproducing detailed properties such as the chemistry of the massive systems (e.g., the α -enhancement). The issue is essentially a reformulation of the long-standing dichotomy between the so-called *hierarchical* and *monolithic* models of galaxy formation, as described respectively e.g. by Eggen et al. (1962) and by White & Rees (1978), possibly including some more sophisticated features (e.g. the *dry merger* scenario, as proposed by Bell et al. 2004). After the Cold Dark Matter cosmological model has got worldwide acceptance, the hierarchical model appeared to be the most convincing scenario of galaxy formation. In recent times, however, the situation has become by far more intrigued than it happened to be in the past. Evidences claiming for a new scenario very close indeed to the classical monolithic one have long been known both observationally and theoretically (see e.g. Chiosi & Carraro 2002), but recently they have become compelling (e.g. Thomas 2011; Gobat et al. 2011).

Merlin & Chiosi (2006, 2007) suggested that a series of primordial mergers of small stellar subunits (in a sort of early hierarchical galaxy formation mechanism), taking place at $z \geq 2$, could lead to the formation of massive objects in the early Universe, resembling the observed systems as far as many different aspects (morphology, density profiles, metallicity...) are concerned. This picture finds support in numerical simulations of the very first stellar populations in the early universe, such as those by Bromm et al. (2002) and Yoshida et al. (2003), who found that the first virialized objects might form at $z \sim 30$ with a typical mass of $10^6 M_\odot$. In this scheme, a first generation of stars in clumps inside primordial haloes of Dark Matter would first enrich the medium in metals, and also act as the building blocks of larger systems with masses up to $10^{12} M_\odot$ via gravitational clustering. The mode of star formation and the efficiency of it is likely driven by the mass and mean density of the proto-galaxy as long ago pointed out by Chiosi & Carraro (2002) and more recently argued on observational basis by Harrison et al. (2011b).

A fundamental piece of the puzzle is the role played by energy feedback during the formation of the galactic systems. Although it has long been known (see e.g. McKee & Ostriker 1977) that supernova explosions can inject large amounts of kinetic and thermal energy into the galactic Interstellar Medium (ISM), only more recently the importance of this phenomenon as a source of pressure support and large scale turbulent motions is being fully recognized. The effects of the energy injection are still under investigation, since two contrasting consequences are expected to follow - i.e. the quenching of the star formation activity due to the increasing of the local gas pressure (*negative feedback*), and the indirect enhancement of the stellar activity due to the formation of layers of cold and dense gas within the turbulent, shocked ISM (*positive feedback*).

However, on larger scales bubbles and super-bubbles of hot and thin material are expected to develop due to the action of energy injection, expanding in the ISM and giving rise to galactic winds. Detailed simulations by Ceverino & Klypin (2009) showed how the correct modeling of the energy injection from supernova explosions is sufficient to eject a large fraction of the gaseous content from a galactic disk. How this phenomenon is expected to influence the evolution of massive spheroids is still to be cleared. On the other hand, it is widely accepted that the energy release from Active Nuclei (AGN) should be a fundamental ingredient for quenching the star formation activity in such systems. However, we show in this study how stellar feedback may be sufficient to achieve the goal, provided it is modeled in an efficient way.

Recently, many numerical studies have focused on the formation of galaxies. Among the most recent ones, Sales et al. (2010) produced different galaxy models with different feedback prescriptions; Sommer-Larsen & Toft (2010) simulated a cluster and studied its evolution and galactic population; Stinson et al. (2010) used a multi-resolution technique to study the formation of galaxies within high-mass halos; Croft et al. (2009) examined a large set of models extrapolated from a large scale cosmological simulation; Naab et al. (2007) presented three models produced *without* any source of feedback; Cox et al. (2006) investigated the process of merging between spirals; Kobayashi (2004, 2005) created a very large number of low resolution chemo-dynamical models of elliptical galaxies; finally, Chiosi & Carraro (2002) presented the first chemo-dynamical NB-TSPH simulations of galaxy formation based on the monolithic scheme, including star formation, cooling, energy feedback, chemical enrichment and galactic winds at varying the initial total mass and density of the proto-galaxy, and brought into evidence that a continuous transition of the star formation mode from a single initial episode down to a recurring series of star forming events is possible at varying these two parameters.

In the present study, we focus on the relation between the initial conditions of the host halo and the final properties of the galaxy. With the aid of a new set of NB-TSPH numerical simulations, performed using the parallel code EVoL (Merlin et al. 2010, 2012 in preparation), we investigate the formation process within a fully consistent cosmological scenario of isolated galaxies with different total masses and initial over-densities. The aim of this study is to cast light on whether the internal properties of a “real”, cosmologically consistent galactic halo made of DM and Baryonic Matter (BM) are sufficient to obtain the variety of morphologies and physical properties which characterizes the population of early-type galaxies. To this aim, we simulate the formation and evolution of twelve isolated early-type galaxy models, from the initial stages of the non-linear evolution of the proto-galactic halos at very high redshift down to at least redshift $z = 1$, and beyond in some cases. The dichotomy between *nature* (that is, the entirety of the initial properties of a proto-galactic system, set *ab initio* from the primordial cosmological fluc-

tuations) and *nurture* (that is, the action of environmental factors) in the process of galaxy formation is fiercely discussed nowadays. In this context, we claim that the initial over-density of an halo should be generally considered part of its *nature*, since it is essentially fixed by the cosmological perturbations from which the halo forms, rather than part of the *nurture* processes, as it is sometimes viewed. For example, belonging to a group or a cluster of galaxies (virializing at recent or present times) imply that the mean over-density of the region is high, and therefore it is likely that galactic halos belonging to the region had, on average, higher initial over-densities than their counterparts forming in the field. Of course, isolated *but* strongly over-dense galactic halos are possible, but they are less probable. If the initial cosmological perturbed density field is a linear superposition of independent waves, having the same power on all scales in scale-free models, then the highest peaks of the density field are expected in regions having, on average, larger over-densities, rather than being isolated. In this picture, the only “real” *nurture* processes are those relative to the action of nearby objects in terms of feedbacks, ram pressure, gas stripping, galactic mergers, etc. Our thesis is that these processes, while being of great importance in some cases, are not those leading the basic mechanisms of galaxy formation and evolution.

By varying the initial halo over-density of our models (in the way we are describing in details in the dedicated Section, see below), we try to mimic the action of the “natural” over-density. In some way, one could consider our “dense” models as systems belonging to generally over-dense regions ultimately becoming groups or clusters of galaxies, and our “less dense” models as totally isolated (field) galaxies.

All the model galaxies are obtained from the same initial proto-halo, however suitably modified in order to systematically change its properties from model to model (see below). We also run a few more cases with different prescriptions, to better explore the effects of some key physical quantities and assumptions.

The paper is subdivided as follows. In Sect. 2 we describe the numerical methods and the setting of the initial conditions. In Sect. 3 we analyze the results of the simulations. In Sect. 4 we discuss the mass density profiles of the stars we have obtained for the models. In Sect. 5 we discuss the positions of the model galaxies on the Fundamental Plane, and compare them with the observational data. Finally, in Sect. 6 we draw some final remarks and conclusions.

2. Methods

2.1. Setting of the initial conditions

To compare the structure and evolution of galaxies with different masses and initial densities, we first generate a reference proto-halo containing particles of DM and BM with specified masses, positions, and velocities; we do so by means of the free software COSMICS by Bertschinger

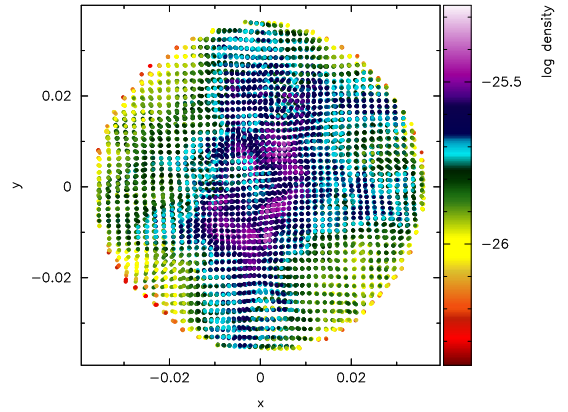


Fig. 1. Initial displacement of gas particles in a central slice along the XY plane for model IDIM. Colors represent mass density [g/cm^3]. All the other models have similar initial conditions.

(1995). Then we systematically modify this proto-halo by varying its mass and/or density, in the way we are going to describe below. By doing so, we introduce only the changes that are strictly necessary to obtain the desired differences in the IC from model to model.

The first proto-halo, which is the largest (i.e., more massive) and densest of the whole set, is obtained generating a box of size $l = 9.2$ comoving Mpc, populated with a grid of 46^3 particles, perturbing their otherwise regular positions consistently with cosmological random gaussian fluctuations, and finally imposing a constrained density peak, to force the formation of a virialized structure at some early time close to the center of the box. The cosmological background adopted here is the Λ -CDM concordance cosmology, with values inferred from the WMAP-5 data (Hinshaw et al. 2008): flat geometry, $H_0 = 70.1$ km/s/Mpc, $\Omega_\Lambda = 0.721$, $\Omega_b = 0.046$ (giving a baryon ratio of $\simeq 0.1656$), $\sigma_8 = 0.817$, and $n = 0.96$.

COSMICS allows to specify the properties of the constrained density peak to model the density field in the desired way. We force a gaussian spherical over-density with average linear density contrast $\delta\rho = 3$, smoothed over a region of radius 3.5 comoving Mpc¹. COSMICS returns the initial comoving positions and the initial peculiar velocities of the particles at the moment in which the particle with the highest density is exiting the linear regime (i.e. it has $\rho/\rho_{bg} - 1 = 1$).

We then single out a sphere of radius $r = l/2$ centered on the center of the box, change the particles coordinates from comoving to the proper physical values (this is simply achieved by dividing the comoving value by the initial cosmological expansion parameter $a = 1/z - 1$, with z the initial redshift that is also provided by COSMICS),

¹ The linear density contrast is defined as $\langle \rho \rangle / \rho_{bg} - 1$ (where ρ_{bg} is the average matter density of the Universe), and it can be extrapolated beyond the linear regime; it would be equal to 1.86 at the epoch of virialization. Requiring a higher value in the software simply bounces back in time the epoch of virialization.

and add a radial outward velocity to each particle, proportional to its radial position and to the initial redshift of the simulation, thus mimicking the effects of a centered outward directed Hubble flow. A minimal amount of solid-body rotation is also added, with spin parameter $\lambda = 0.02$.² Finally, each particle is split into a DM and a BM particle (gas), this latter with a small displacement from its original position to avoid numerical divergences. Indicating with m_0 the mass of the original particle and with $f_{BD} = \rho_{BM}/\rho_{DM} \simeq 0.1656$ the cosmological ratio between the BM and DM densities, we assign to each gas particle the mass $m_{gas} = f_{BD} \times m_0$. Consequently, $m_{DM} = (1 - 0.1656) \times m_0$. The final proto-halo consists of $\simeq 58,000$ DM particles plus an equal number of gas particles. In the following, this first *high density, high mass* halo is identified with the abbreviation HDHM.

To obtain the different galaxy models, we then systematically modify the properties of the HDHM halo.

By varying the spatial dimensions of the box, COSMICS returns the consistent modifications resulting in smaller (and therefore less massive) haloes, without varying their average density (the particle masses are obviously consequently re-scaled).

On the other hand, to obtain constant-mass but varying-density halos, *before* calculating the perturbed positions and velocities of the particles we artificially decrease the densities at each point of the initial grid, thus varying the average density but conserving the total mass. Since the ICs are produced at the moment in which the highest perturbation peak exits the linear regime, the initial spatial configuration is still a superposition of independent plane waves with different wave-numbers and random phases, i.e. a *gaussian field* with $\delta(\mathbf{q}) = \sum_{\mathbf{k}} \delta_{\mathbf{k}} \exp(i\mathbf{k} \cdot \mathbf{q})$, where $\mathbf{k}$ is the wavenumber and $\mathbf{q}$ is the Lagrangian coordinate corresponding to the unperturbed comoving position of a mass element. Dividing local (over-)densities $\delta(\mathbf{q})$ by a constant factor f returns a self-similar gaussian field in which the Fourier coefficients $\delta_{\mathbf{k}}$ is divided by the same factor, implying that the variance $\sigma^2 \propto \sum_{\mathbf{k}} \delta_{\mathbf{k}}^2$ of the perturbation is reduced by a factor f^2 . COSMICS subsequently computes the effective displacement and velocities of the particles via the Zel'dovich approximation,

$$\begin{aligned} \mathbf{x}(\mathbf{q}) &= \mathbf{q} + D_+ \mathbf{d}(\mathbf{q}), \\ \mathbf{v}(\mathbf{q}) &= \dot{D}_+ \mathbf{d}(\mathbf{q}), \end{aligned} \quad (2)$$

² The spin parameter λ is

$$\lambda = \frac{J|E|^{1/2}}{GM^{5/2}} \quad (1)$$

where J is the angular momentum, E is the initial binding energy, and M the total mass of the system. Typical values of λ range from 0.02 and 0.08 (White 1984), which corresponds to angular velocities of the order of fractions of a complete rotation over time-scales as long as about ten *free-fall* time-scales (Carraro et al. 1998).

where the displacement field $\mathbf{d}(\mathbf{q})$ is given by the relation $\nabla \cdot \mathbf{d} = -D_+^{-1} \frac{\delta\rho(\mathbf{q})}{\rho}$, and D_+ is the cosmic growth factor at the initial redshift, which is a function of the cosmological model. We avoid the simplest choice of imposing a less pronounced density peak (i.e., a lower value of the initial density contrast δ in the constrained field), because this would reduce the density of the constraint *without* modifying the random perturbations, thus resulting in a very different global field with more spectral power on the small scales.

The mean density is reduced by a factor of 15% to obtain *intermediate density* halos, and by another 15% to obtain *low density* halos (these halos therefore initially having 72.25% of the average density of the densest halos). Finally, a set of halos with very low initial over-density was obtained reducing the mean density by 50%. As already pointed out, the different densities can be considered as a way to mimic the possible environments in which a proto-halo begins to form.

On the other hand, given the extremely wide range of galactic masses observed in the Universe, we modified the total mass of the halos rescaling it by factors of $\simeq 64$ each time (and reducing the radius by factors of 4).

The initial redshifts of models with equal initial density but different mass are different from one another, because the moment of exiting from the linear regime is a function of the mass of the perturbation.

In this way, we obtain twelve initial halos, each one being identified by a string of four or five letters, of which the first two (or three) refer to its density and the last two to its mass (e.g., IDLM is for *intermediate density, low mass*; VLDHM stands for *very low density, high mass*, etc.). The key initial value of a few important parameters for all the twelve halos are listed in Table 1, whereas a sketch of the density contrast inside the initial sphere projected on the XY plane for the case of the model IDIM is shown in Fig. 1 where a number of lumps of matter are clearly evident. The number of particles in all models is similar to that of the HDHM case, with some minor differences due to the different displacement of the particles in the outskirts of the retained sphere.

We point out that all these halos are fully consistent with the underlying cosmological background. The expected DM halo mass function follows the laws by, e.g., Press & Schechter (1974) or Sheth et al. (2001), and/or the analytical fits of numerical simulations (e.g. Warren et al. 2006). In this picture, a $M \simeq 10^{13} M_\odot$ halo is becoming “typical” on a ~ 10 Mpc scale (the typical linear size of a large galaxy cluster) at redshifts below $z \sim 5$, which is compatible to the collapsing redshift of our HDHM model (see below). Such massive halos are also expected not to be rare on ten times larger scales at $z \sim 10$ (see e.g. Lukić et al. 2007, their Fig. 2).

Table 1. Initial parameters for the twelve models of proto-galaxies. Left to right: total masses, corresponding (initial) gaseous masses, mean halo over-densities ($\delta\rho - 1$) at redshift $z = 30$, initial redshift, halo initial proper physical radius, mass of a gas particle.

Model	$M_{tot} [M_\odot]$	$M_{gas,ini} [M_\odot]$	$\langle \delta\rho - 1 \rangle_{z=30}$	z_{ini}	R_{ini} [kpc]	$m_{gas} [M_\odot]$
HDHM	1.75×10^{13}	2.90×10^{12}	0.39	46.34	97.17	4.97×10^7
IDHM	1.75×10^{13}	2.90×10^{12}	0.30	39.24	114.31	4.97×10^7
LDHM	1.75×10^{13}	2.90×10^{12}	0.23	33.20	134.49	4.97×10^7
VLDHM	1.75×10^{13}	2.90×10^{12}	/	22.67	194.34	4.97×10^7
HDMM	2.69×10^{11}	4.45×10^{10}	0.46	53.79	20.99	7.79×10^5
IDIM	2.69×10^{11}	4.45×10^{10}	0.33	45.57	24.69	7.79×10^5
LDIM	2.69×10^{11}	4.45×10^{10}	0.25	38.59	29.05	7.79×10^5
VLDIM	2.69×10^{11}	4.45×10^{10}	/	26.37	41.98	7.79×10^5
HDLM	4.17×10^9	6.91×10^8	0.54	63.23	4.48	1.22×10^4
IDLM	4.17×10^9	6.91×10^8	0.39	53.60	5.27	1.22×10^4
LDLM	4.17×10^9	6.91×10^8	0.29	45.40	6.20	1.22×10^4
VLDLM	4.17×10^9	6.91×10^8	0.16	31.11	8.96	1.22×10^4

2.2. The NB-TSPH code EvoL

EvoL is a Lagrangian N-Body (NB) parallel code designed to study the evolution of astrophysical systems on any spatial scale, from large cosmological volumes to sub-galactic regions. It is based on the standard Tree Algorithm for the description of the gravitational interactions Barnes & Hut (1986), and on a modern version of the Smoothed Particle Hydrodynamics (SPH) method to model gas dynamics. Its basic features concerning the treatment of gravitational interaction and hydrodynamics are described in details in Merlin et al. (2010) and Merlin et al. (2012 in preparation); here, we briefly recall some of its characteristics.

EvoL includes some peculiar features, such as the lagrangian formulation of the SPH method including ∇h correcting terms, an artificial thermal conduction to smooth out unphysical surface discontinuities, and finally the variable adaptive softening lengths with consistent correcting terms in the equations of motion. In the present set of models, we switch off the artificial thermal conduction, since its usage in a simulation including radiative cooling has not been tested yet and may lead to unphysical behaviors, with particles exchanging heat because of numerical conduction, and simultaneously losing energy due to physical cooling; this issue deserves an accurate analysis which is planned for future work. We also adopt variable softening lengths; this is an important difference with respect to other similar experiments, where a constant softening has been used almost exclusively (a noticeable exception being Nelson 2006, although in that case no correcting terms in the equations of motion were considered). In the present models, we let the softening (and smoothing) lengths vary freely. In a few side models calculated for the sake of comparison, we made use of a minimum value of $\epsilon_{min} = h_{min} = 100$ pc. Such a low limit would provide sufficient resolution in all the cases of interest. However, these side models show that the results are very sensitive to these parameters, even if all the models

are essentially similar in their global aspects (see Sect. 3.5 for a discussion on this issue).

In the present models, we also include a density-dependent pressure limit to avoid artificial clumping of poorly resolved gas. The method is similar to the one presented by Robertson & Kravtsov (2008). In practice, at each time-step all SPH particles compute the local Jeans mass, which is a function of density and temperature, i.e.

$$m_{Jeans} = \frac{4\pi}{3} \rho_{gas} \left(c_s \sqrt{\frac{\pi}{G\rho_{tot}}} \right)^3, \quad (3)$$

where $c_s = \sqrt{\gamma(\gamma-1)u}$ is the local sound speed, ρ_{gas} is the density of the SPH particle, and ρ_{tot} is the *total* local density (i.e., gas plus DM and/or stars)³. Then, the internal energy of each particle (limited to computing the hydrodynamical forces) is chosen as the maximum between the real value and an “effective” value, defined as the energy that the gas sphere should have to be stabilized against collapse:

$$u_{eff} = N_{jeans} \times \frac{1}{\gamma(\gamma-1)} \rho_{tot} \left(\frac{3N_{SPH}m}{4\pi^{\frac{5}{2}}\rho_{gas}} \right)^{\frac{2}{3}}, \quad (4)$$

where m is the particle mass, N_{SPH} is the number of particles required to resolve a region (indicatively, this number can be considered equal to the typical number of neighbours, i.e. about 60 in our models, or a multiple of this value), γ is the adiabatic index, and N_{jeans} is a free parameter ≥ 1 . In these models we use $N_{jeans} = 15$ (see Merlin et al. 2012 in preparation, for all details).

Note that u_{eff} is only used to compute the mechanical acceleration due to the internal gas pressure. In the expressions for the variation of internal energy, dissipation, and cooling, the *real* value of u is adopted. In this way, the temperature of gas particles can reach low values, without reaching high densities, thus creating regions

³ This expression for the Jeans mass of a gas sphere in presence of a non collisional component can be obtained from a perturbative analysis of the stability of a pressurized cloud.

of cold material without spurious numerical clumping of particles.

We run the simulations in physical coordinates since at present we are not interested in large scale motions. Moreover, this choice rules out the need for periodic boundary conditions, which would require a large amount of additional CPU-time. However, we point out that the evolution *in void* has some non-negligible drawbacks. First, the matter infall cannot proceed beyond the limit imposed by the absence of particles outside the models. Second, the feedback-heated gas leaving the galaxy will escape freely without feeling any drag. Third, the density of the systems cannot be consistently computed and the models evolve as small closed Universes rather than overdense regions within a flat Universe, so the redshifts are only indicative of the real evolutionary time-line.

Finally, we include radiative cooling, as well as star formation and stellar feedback. As they are described in detail in Merlin et al. (2012 in preparation), in the sections below we limit ourselves to summarize their key aspects.

2.3. Radiative cooling

We adopt the cooling functions described in Carraro et al. (1998), which are based on those elaborated by Chiosi et al. (1998). In brief, for temperatures greater than 10^4 K they lean on the Sutherland & Dopita (1993) tabulations for a plasma under equilibrium conditions and metal abundances $\log[Z/Z_\odot] = -10$ (no metals), -3 , -2 , -1.5 , -1 , -0.5 , 0 (solar) and 0.5 . The chemical properties of gas particles are followed in details, tracking the abundances of Fe, O, Mg, Si and C, and the global metallicity Z .

For temperatures in the range $100 < T < 10^4$ K the dominant source of cooling is the H_2 molecule becoming rotationally and/or vibrationally excited through a collision with an H atom or another H_2 molecule and decaying through radiative emission; the data in use have been derived from the analytical expressions of Hollenbach & McKee (1979) and Tegmark et al. (1997). We point out that in the present simulations it is assumed that *all* hydrogen becomes molecular below 10^4 K, so the cooling rate in this range of temperatures may be somewhat overestimated. However, cooling from collisions of neutral hydrogen atoms with electrons from ionized metals (when present) is neglected. This may somewhat compensate for the excess of cooling due to molecular hydrogen. In any case this is a (marginally) weak point of our current treatment of cooling that should be improved.

Finally, for temperatures lower than 100 K, Sutherland & Dopita (1993) incorporate the results of Hollenbach & McKee (1979) and Hollenbach (1988) for CO molecule as the dominant coolant. The mean fractionary abundance of CO is given as a function of $[Fe/H]$ (see Carraro et al. 1998, for further details).

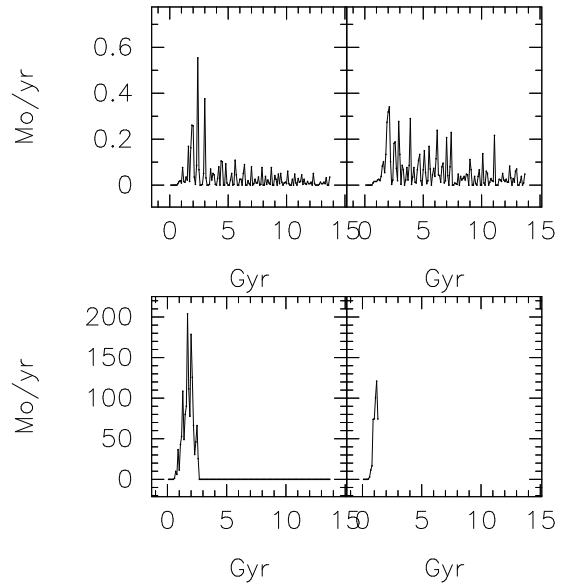


Fig. 2. Star formation histories in four test models (see text). Top panels: $M_{tot} = 1.38 \times 10^{10} M_\odot$; bottom panels: $M_{tot} = 1.38 \times 10^{12} M_\odot$. Left panels: star formation efficiency $\epsilon_{SF}=1$. Right panels: star formation efficiency $\epsilon_{SF}=0.2$.

We also include the contribution by inverse Compton cooling

$$\Lambda_{Compton} = 5.41 \times 10^{-36} x_e [T - T_{CMB}(1+z)](1+z)^4 \quad (5)$$

in erg/s/cm³, with the usual meaning of symbols (see e.g. Ikeuchi & Ostriker 1986).

The radiative cooling is computed separately from the SPH equation of energy conservation, due to its very short time-scales. During a dynamical time-step, the density and the metallicity of gas particles are kept fixed, as well as the mechanical heating rate. On the contrary, the temperature is let vary under the action of cooling, and the new cooling rates are simultaneously obtained as a function of the new temperature. The process is iterated until the end of the time-step.

2.4. Star formation

Star formation is modeled by means of a stochastic method, similar to that introduced by Churches et al. (2001) and Lia et al. (2002). First, only gas particles belonging to convergent flows (such that $\nabla \cdot v < 0$) and denser than a suitable threshold ρ_* are considered eligible to form stars. In our models we adopt $\rho_* = 5 \times 10^{-25}$ g/cm³. We do not impose any restriction on the temperature; this choice is motivated by the fact that thermal instabilities can produce star forming sites even within high temperature gas.

Then, if a particle satisfies these criteria, it is assumed to form stars with a rate $d\rho_*/dt = \epsilon_{SF}\rho_g/t_{ff}$, where $t_{ff} \simeq 0.5/\sqrt{G\rho_{tot}}$ is the free-fall time and ρ_{tot} is the local total mass density (DM plus BM), and ϵ_{SF} the dimensionless efficiency of the star formation process. This means

that a gas particle is expected to transform a fraction ϵ_{SF} of its mass into stars over its free-fall time scale. However, a stochastic description based of the star forming mechanism is best suited to avoid the creation of exceedingly large numbers of star particles. Thus, gas particles undergo a Monte Carlo selection to see whether they will actually form stars - in this case being instantaneously transformed into collisionless star particles - or not (see also Lia et al. 2002; Merlin et al. 2012 in preparation); to do so, a random number $r \in [0, 1]$ is drawn and compared to the probability that the particle is actually forming stars. An obvious choice would be to use the probability $P = \epsilon_{SF}/t_{ff} \times \Delta t$, where Δt is the dynamical time-step. However, the exact evaluation of the probability the real probability must take into account that the dynamical time-steps are generally much shorter than the free-fall times of particles. This implies that within a free-fall time scale a number of star forming events and corresponding random draws are possible depending on Δt , and therefore so does the global probability for a single particle to be turned into stars. If the global probability over t_{ff} is P , the stochastic process of random selection must be corrected considering that the probability summed over the number of draws $n \simeq \Delta t/t_{ff}$ must be equal to the probability of a single draw within the whole free-fall time. A first step is to take as the probability within a single time-step the value $p \simeq P/n$. However, the conjuncted probability of a successful event over n draws is $p \times \sum_{i=1}^n (1-p)^{i-1} < P$. The correcting multiplicative term is easily obtained as

$$f = \frac{P}{\frac{p}{n} \times \sum_{i=1}^n (1 - \frac{p}{n})^{i-1}}, \quad (6)$$

so the final probability during the time-step is $f \times p$. If $r \leq fp$, then the gas particle is instantaneously turned into a stellar particle.

In the present models, we assume $\epsilon_{SF} = 1$, an extreme choice which turns out to be much larger than the value inferred from the observational data in the local Universe, i.e. $\epsilon_{SF} \simeq 0.025$ (Lada & Lada 2003; Krumholz & Tan 2007), and theoretical considerations on the global stellar-to-total mass in galaxies suggesting $\epsilon_{SF} \leq 0.1$. While it is unclear whether ϵ_{SF} should be maintained constant during different cosmic epochs given the different mechanisms leading the star formation process at high redshifts and in metal poor regions with respect to local molecular clouds, it is easy to understand that a 100% efficiency (meaning that *all* gaseous particles satisfying the SF criteria will turn the totality of their mass into stars within a free-fall time) cannot be considered a physically plausible assumption.

However, adopting lower values for ϵ_{SF} turns out not to be useful. While the results for the star formation histories adopting small values of ϵ_{SF} are apparently quite similar to those with $\epsilon_{SF} = 1$, the simulations become much more time-consuming. Fig. 2 shows the star formation history of two galaxies with high and intermediate mass in which first $\epsilon_{SF} = 1$ and then $\epsilon_{SF} = 0.2$. As far

as the star formation history is concerned, the results are very similar to each other, at least up to the times reached by all the models. However the simulations proceed at very different speed. The models with $\epsilon_{SF} = 0.025$ proceed so slowly that they could reach the age of a few Gyr only after several weeks of uninterrupted run. The reason is that dense and cold clumps of matter, interacting with hotter material heated up by close SN explosions, require extremely small time-step; if gas can easily form stars, it can elude this critical situation, and the simulation can proceed more fluently.

The apparently low impact of ϵ_{SF} on the final SF history may be explained considering the self-regulation cycle between star formation and energy feedback. A high SF efficiency implies a strong and sudden burst of stellar activity; but young stars soon pressurize their surroundings via energy feedback (see below), quenching the formation of new stars. However, if the gas is sufficiently dense, radiative cooling is effective and further star formation can soon take place (positive feedback). With a low efficiency, less stars form. Their heating is consequently lower, and more stars can soon form *before* the feedback halts their formation. This ultimately leads to the same situation as in the previous case, with perhaps the minor consequence of a delayed enrichment in heavy elements of the medium. Clearly, other parameters play a more important role; for example, the density threshold ρ_* and the efficiency of feedback. Therefore the choice to assume $\epsilon_{SF} = 1$ and speed up the calculations seems to be the best compromise. However, one should always keep in mind that adopting $\epsilon_{SF} = 1$ may have other consequences in terms of the dynamical evolution of the systems, favouring a collisionless collapse instead of a dissipative one (see Sects. 5 and 6).

2.5. Energy feedback

When a gas particle is turned into a star particle, it can be considered to represent a Single Stellar Population (SSP) made of many real stars, which starts re-fueling the ISM with heavy chemical elements and energy, mainly because of winds from young massive stars, and Supernova (SN) explosions.

We consider two regimes for the energy injection by winds. When the SSP is young, the main source of energy are young massive stars. Their winds have very high speeds. We assume a constant velocity $v_{YSO} = 1500$ km/s, and the kinetic energy of these winds is assumed to be thermalized and released within the surrounding medium with an efficiency of 20% (see e.g. Dyson & Williams 1997). In the late stages of the SSP evolution, slow velocity winds from smaller stars becomes predominant; in these cases we assume an outflow speed $v_{OSO} = 60$ km/s, and the same thermalization efficiency.

Each *real* SN explosion is expected to deposit some 10^{51} kinetic ergs in a very small region, and on a short time scale. However, most of this energy is soon radiated

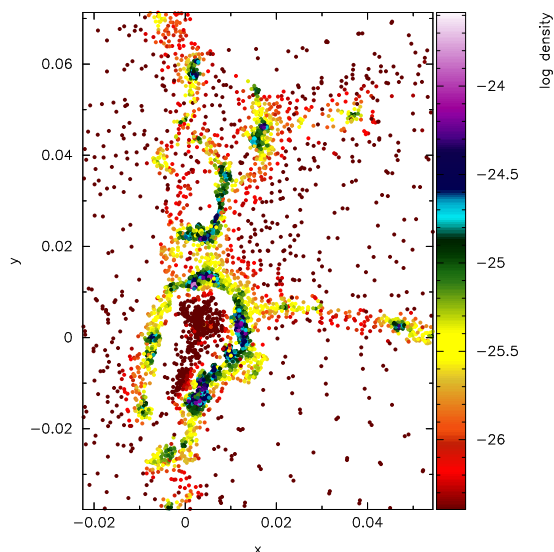


Fig. 3. Snapshot showing the displacement of gas particles on a central slice on the XY plane (model IDIM, $z = 7.6$). Features caused by energy injections from stars (the clumped brown particles near the origin of coordinates) such as expanding bubbles and shock fronts are clearly visible, as well as collapsing cosmological filaments.

away, and only a few percent is subsequently thermalized. We use the results by Thornton et al. (1998) to obtain an analytic approximation of the fraction of the initial energy which becomes available at the end of the expanding phase of the SN bubble. However, in the real Universe SN explosions take place in already shock-heated regions, with temperatures raised up to some 10^6 K, because of the action of a photo-ionizing flux from massive young stars, stellar winds, and previous SN explosions. We point out that all this is not taken into consideration in Thornton’s study (while it is, for example, in Cho & Kang 2008), so a large range of uncertainty still affects the description of the whole process. A single star particle represent an entire SSP, so a large number of SN explosions are expected to take place within a single star particle over a rather long time scale. The number of SN explosions (attention must be paid to release the energy in discrete bursts consistently with the number of real explosions), as well as the amount of gas released by a SSP of given age and mass during a time-step and its chemical composition are computed with the technique described in Lia et al. (2002), using theoretical SSPs (in EVoL, the Padova tracks are adopted, see Greggio & Renzini 1983), and adopting the initial mass function for the real star in the SSP (star particle) of Kroupa (2001). We refer the reader to the above papers for all the details. It is worth clarifying that we do *not* adopt the Lia’s stochastic approach to model SN explosions. On the contrary, each SSP continuously releases gas and energy at each time step.

Finally, the total budget of thermal energy produced (by stellar winds and SN explosions) at each time-step is given continuously to a sort of gas reservoir assigned to star particles, ultimately formed by the gaseous mate-

rial ejected from the SSP. This reservoir, due to its high temperature, acts a piston on the neighboring particles, effectively injecting kinetic energy into the surrounding interstellar medium. In the next Section we clarify some aspects concerning this method.

2.6. Fate of the gas ejected by SSPs

A crucial issue in numerical simulations of galaxy formation is the *correct treatment of the gas ejected by SSPs*. Since it is not feasible to continuously form new gas particles from the gas ejected by SSPs, it is necessary to develop a consistent, yet simple method to model this process.

There are two main sources of uncertainty. First, one has to decide how to deal with the hot gas *soon after* its ejection from the SSPs. This gas is essentially composed by matter ejected by stars in form of winds and/or SN explosions. How should it be let cool? How to compute its density? It has long been known that distributing the feedback energy among neighboring particles and then letting them normally evolve among the other cold ones gives poor results and very low feedback efficiencies (Katz 1992): the energy is soon radiated away and, in critical situations, hot particles escape from the region due to excessive hydrodynamical friction with their neighbors, which is not physically grounded. Also, distributing the gas released by the SSP to nearby SPH particles at each time-step has proven to be inefficient, since the large amounts of energy released locally are smoothed out too strongly. Therefore, as a first step we choose to keep the ejected gas fixed to the parent stellar particles. This choice, while sounded in the first stages of the lifetime of the ejected gas, becomes clearly unphysical as time passes by. Sooner or later, the gas ejected by SSPs should mix with its nearby particles and begin to move freely. The second issue is therefore to decide how and when such gas should be let evolve “normally”, as a standard SPH gas particle - i.e., when it should dynamically decouple from its parent star particle.

We therefore subdivide the evolution of the ejected gas into two temporal steps. The gas is kept locked to the parent SSP during the first stages of its evolution, namely from its first appearance in the SSP to the age of $t_{\text{eject}} \simeq 8 \times 10^7$ years, a typical time scale for Type II Supernovae (short lived) to disappear. During this first stage, this gas does not contribute to the acceleration of the *parent* particle, but acts as a piston exerting pressure on the neighbouring particles because of its high internal energy. Its density is computed normally with the SPH method, and radiative cooling is let occur normally during this phase. In this way, this “almost-zero-mass” SPH particle, locked to a collisionless stellar particle, can effectively act as a source of feedback, pressurizing the region and giving rise to complex structures resembling observed bubbles and super-bubbles (see Fig. 3). The mechanism is similar to the one proposed by Pelupessy (2005), who used zero-mass *pressure particles* to model the stellar feedback. However, instead of introducing zero-mass particles,

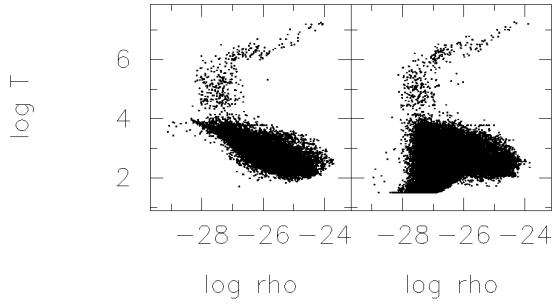


Fig. 4. Phase diagram ρ vs. T for the IDLM model at $z = 8.5$, with (left) and without (right) the inclusion of a photo-ionizing background radiation at $z = 9$. See text for details.

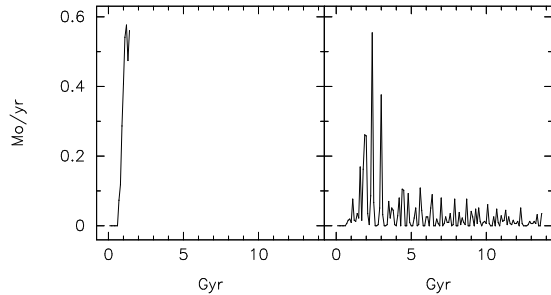


Fig. 5. Star formation histories in two small models (see text), with (left) and without (right) energy feedback from the stars

in EVOL the real gaseous mass ejected from SSPs during the current time-step (always much smaller than the typical mass of a SPH particle) is computed, and added to a gaseous reservoir locked to the star particles; these “hybrid” particles interact hydrodynamically with the surrounding gas particles only considering the pressure exerted by its small fractional gaseous mass.

When the age t_{eject} is reached, the hybrid stellar/gaseous particle is let free to search among its neighboring hybrid particles, to collect a sufficient amount of gas (ejected by nearby SSPs, which must be beyond their first evolutionary phase too) to form a whole new gas particle. If a mass of gas similar to the mass of a standard SPH particle is collected (in the present models we allow a difference in particle masses of $\sim 50\%$), then a new gaseous particle is spawned, with position, velocity and temperature obtained from the weighed mean over all contributing parent star particles (which are consequently depleted of their gaseous content). From now on, the new particle is let free to evolve normally. This method has proven to give satisfactory results, at the expense of a slight increase of the total number of particles during the run.

2.7. Chemical enrichment

Stellar gaseous ejecta contain heavy elements, which are redistributed within the surrounding gas through a diffusive process, similar to the diffusive approximation for the thermal conduction mechanism. All details are described in Merlin et al. (2012 in preparation) to whom the reader should refer for the sake of brevity.

2.8. Re-ionization and the first generation of stars

Cosmic re-ionization, as inferred from the WMAP-5 data, is currently believed to have taken place at $z \leq 10$. In our models, we include a crude treatment of photo-heating from the re-ionized free electrons. Following Pawlik & Schaye (2009), we increase the temperature of all gas particles fueling their internal energy reservoir with an additional budget of 2 eV per proton, which are instantaneously injected at $z = 9$. The effect of this energetic boost is to raise the temperature of the low density particles to some $\sim 10^4$ K. In high density regions, however, radiative cooling (see below) is more efficient in keeping the temperature well below this value. Fig. 4 compares two snapshots taken from models in which this re-ionization effect is (is not) included. However, it is worth noting that this way of proceeding lets the gas particles free to cool down after the sudden injection of energy; consequently, the cosmic web may happen to get too cold at later times (this can be seen in Fig. 18 in Sect. 3.4).

We also include in a very simple fashion a model for an early population of metal-free stars otherwise known as Population III (Pop III) stars (e.g., Bromm et al. 2002; Yoshida et al. 2003 and references therein for more details). We assume that when a metal-free gas particle is selected to form stars for the first time (see Sec. 2.4) it generates a Pop III SSP, mainly composed of massive stars because the underlying initial mass function is currently believed to be heavily skewed towards massive stars (Bromm et al. 2002). Consequently the Pop III SSP is short lived and nearly suddenly injects lots of energy into the interstellar medium by stellar winds and supernova explosions. In our model, a gas particle undergoing this process does *not* turn into a stellar particle, but has simply its temperature raised to the temperature $T \simeq 2 \times 10^7$ K, and acquires some metallicity, approximately $Z \simeq 10^{-4}$. Finally, in the present model we do not track the fate of the Pop III remnants in the form of ultra-compact objects or black holes. Since these remnants could be the seeds to form to a central super-massive black hole, most likely fueling a period of AGN activity, a complete description of the remnants from Pop III stars is in progress.

2.9. General remarks

Before going into the results of our study, we summarize here a few possible drawbacks of our models.

First, as already mentioned the stellar efficiency ϵ_{SF} is 1, in contrast to what observation and theoretical order-of-magnitude analysis would suggest. This has proven to cause non-negligible effects on the dynamical evolution of the systems; they will be discussed in the following Sections. Second, no ad-hoc treatment for the multi-phase nature of the interstellar medium has been included. A subdivision into different phases naturally develops within the gas (see Sect. 3.1); however, it has been long known that numerical effects like the *overcooling* may arise if the gas is modeled according to the standard SPH method

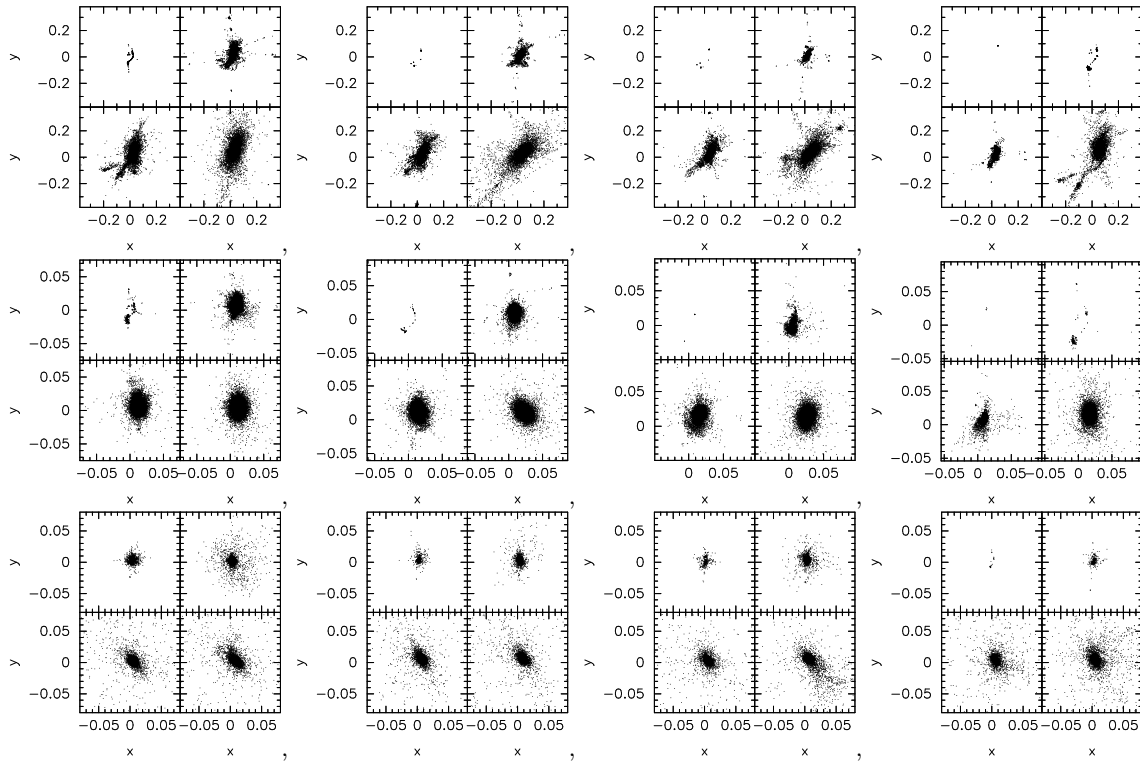


Fig. 6. Evolution as a function of the redshift of the twelve models with different total mass and initial densities. The position of the star particles are projected on the XY plane. The upper row corresponds to the high mass case, the central row to the intermediate mass case, and the bottom row to the low mass case. The initial density decreases from left to right. So the models displayed in the three groups of small panels in the left column are HDHM, HDIM, HDLM. The same in the other columns but for the ID-, LD- and VLD- cases, respectively. In each sub-panel the redshift changes from left to right and top to bottom: $z \simeq 11.5$ (7 for VLDHM), 5, 3, 1.

(Merlin & Chiosi 2007), so we point out that a more detailed treatment would probably give a more accurate description. Third, we did not consider the possibility of the presence of nuclear activity in our model galaxies, neglecting both the dynamical action of a central super-massive black hole, and the feedback from an AGN, both of which are currently believed to play important roles in the evolution of a proto-galaxy. Fourth, we did not include any different sources of feedback apart from stellar - i.e., magnetic pressure or cosmic rays feedback. Finally, our cooling functions do not take into account the state of the Hydrogen gas; in particular, the molecular fraction of Hydrogen, which strongly influences the cooling efficiency below 10^4 K in primordial, non-enriched gas (see Sect. 2.3).

3. Model results

All the simulations have been run on the HP Blade Server MONSTER with 14 hosts and 116 CPUs at the Astronomy Department of Padova. Each major galaxy model made use of four parallel CPUs, whereas companion test models of small size were run on single processors. On the whole, the simulations took $\sim 100,000$ CPU hours, requiring from 30,000 to 80,000 time-steps.

There are two groups of models: the first one contains the test computations performed to assess the model re-

sponse to varying some important parameters. They are calculated with a smaller number of particles (usually from 5000 to 10000). The following questions are addressed:

(i) **Star formations efficiency.** These are models already described in sect. 2.4, that are calculated to check the sensitivity and overall behaviour at varying the SF dimensionless efficiency ϵ_{SF} of the star formation rate.

(ii) **Stellar feedback.** Then, we run a simulation excluding the feedback from stars. As expected, the star formation rate grows dramatically, so that the run is stopped after a few Myr (see Fig. 5). Our prescription for the stellar feedback can therefore be considered as reasonably efficient.

Not all the models have been calculated until the present time to save computational time but all of them have been carried out to the stage in which the mass assembly is completed (say a few Gyrs).

Second, there is the reference set of models, with higher resolution. All of them are calculated well beyond the completion of the mass assembly, a few even down to the present age or very close to it. All important data referring to the models at the same reference redshift $z = 1$ are summarized in Table 2, whereas in Table 4 are displayed similar data but for the last computed stage of each model (different age and redshift). In the following we shortly describe the main results for the twelve reference models.

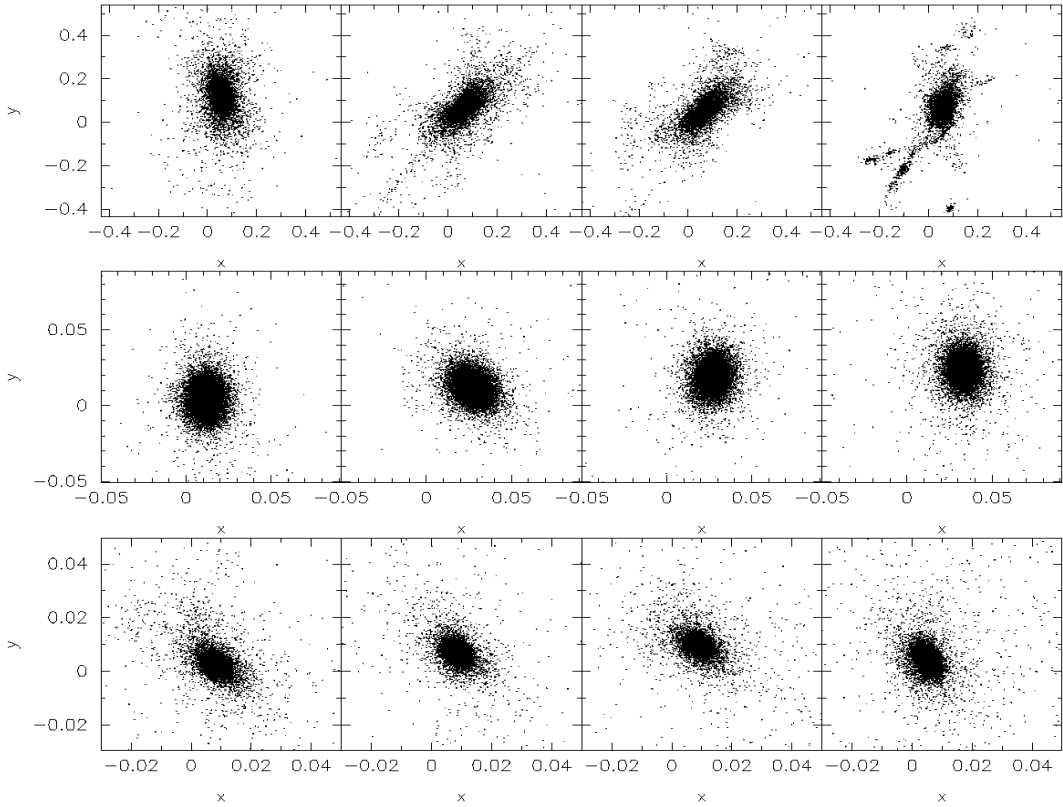


Fig. 7. Stellar systems at their final redshifts, projected on the XY plane. Left to right, top to bottom: HDHM ($z = 0.22$), IDHM ($z = 0.77$), LDHM ($z = 0.49$), VLDHM ($z = 0.83$); HDIM ($z = 1.0$), IDIM ($z = 0.75$), LDIM ($z = 0.58$), VLDIM ($z = 0.15$); HDLM ($z = 0.36$), IDLM ($z = 0.22$), LDLM ($z = 0.05$), VLDLM ($z = 0.0$). Positions are in proper Mpc.

We start showing in Fig. 7 a snapshot of their geometrical structure at the last computed stage projected on the XY plane. In most cases the spatial distribution of stars resembles that of classical elliptical structures; in one case a filamentary substructure, and perhaps satellites companions are visible. Fig. 10 displays the star formation histories of the twelve models along their lifetime, down to the last simulated epoch. As found by Chiosi & Carraro (2002) and expected here, the kind of star formation at work changes with the galaxy mass and initial density. It is monolithic-like (a single dominant initial episode) in high mass and/or high initial density, it gradually turns into a broad rather long and mild activity at decreasing mass/and/or initial density, and finally it gets like a series of discontinuous very episodes (bursting mode) for low mass and/or low initial densities galaxies. The point will be expanded in more detail below.

3.1. Dynamical evolution and final morphologies

All the models go through an initial phase of expansion, as they follow the Hubble flow. The central regions, in which the density peak is more pronounced, soon detach from the outward motion and start re-collapsing, forming the core of the virialized structure. As time goes on, more and more external regions turn around and fall onto the central virializing object. The process can be roughly described as a secondary infall, with time-scales strongly

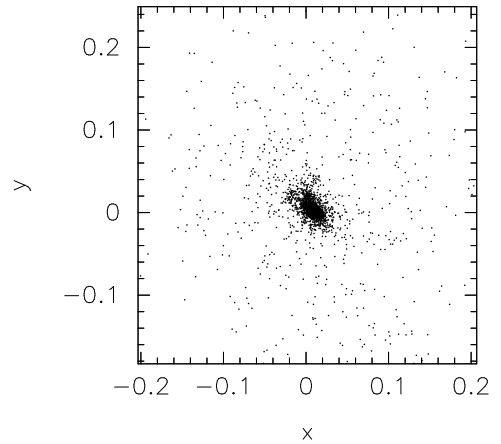


Fig. 8. Low mass models develop extended halos of stellar particles around the central object (in this picture, model HDLM is shown at $z = 1$; distances are in proper Mpc).

dependent on the properties of the particular halo under consideration. More massive halos globally virialize later with respect to smaller ones, as expected in the hierarchical cosmology. However, denser halos also re-collapse sooner than the shallower ones, so the epoch of virialization is decided by the interplay between these two effects.

At times depending on the depth of the central potential well, the central regions of the halos begin to reach the critical density ρ_* at which the gas becomes able to form

Table 2. Properties of the twelve major models at $z = 1$ (i.e. after $\simeq 6$ Gyr of evolution). Left to right: total stellar mass, total mass of the virialized halo, fraction of the total initial mass composed by star, fraction of the initial gas converted into stars, virial radius of the whole system (i.e. stars, Dark Matter and gravitationally bound gas; it is the radius at which the mass density is 200 times the background density at the epoch of virialization, which we assume to have happened at $t \simeq 2$ Gyr looking at the energy trends of the models: see Fig. 23), half mass radius of the stellar system projected on the XY plane, axis ratios of the stellar system projected on the XY plane.

Model	M_* [$M_\odot$]	M_{vir} [$M_\odot$]	M_*/M_{vir}	$M_*/M_{gas,ini}$	$r_{vir,tot}$ [kpc]	$R_{1/2,*}$ [kpc]	b/a_{XY}
HDHM	7.8×10^{11}	1.1×10^{13}	0.071	0.27	153.0	15.7	0.53
IDHM	7.5×10^{11}	1.0×10^{13}	0.075	0.26	141.8	16.8	0.54
LDHM	7.4×10^{11}	9.4×10^{12}	0.080	0.26	133.8	14.5	0.57
VLDHM	6.3×10^{11}	7.5×10^{12}	0.080	0.22	112.5	11.2	0.52
HDIM	2.0×10^{10}	1.6×10^{11}	0.13	0.45	37.6	5.7	0.62
IDIM	1.9×10^{10}	1.5×10^{11}	0.13	0.43	35.7	5.8	0.63
LDIM	1.9×10^{10}	1.4×10^{11}	0.14	0.42	33.3	5.3	0.58
VLDIM	1.3×10^{10}	1.1×10^{11}	0.12	0.29	28.3	5.8	0.57
HDLM	1.2×10^8	2.6×10^9	0.028	0.17	9.2	2.2	0.69
IDLm	1.0×10^8	2.5×10^9	0.040	0.14	10.0	2.5	0.64
LDLM	8.9×10^7	2.3×10^9	0.021	0.13	11.8	2.2	0.56
VLDLM	5.0×10^7	1.7×10^9	0.011	0.07	10.5	2.7	0.58

Table 3. Properties of the twelve major models at the final stage of the simulated evolution. Left to right: final redshift, final age, total stellar mass, total mass of the virialized halo, fraction of the total mass composed by star, fraction of the initial gas converted into stars, virial radius of the whole system (i.e. stars, Dark Matter and gravitationally bound gas; it is the radius at which the mass density is 200 times the background density at the epoch of virialization, which we assume to have happened at $t \simeq 2$ Gyr looking at the energy trends of the models: see Fig. 23), half mass radius of the stellar system projected on the XY plane, axis ratios of the stellar system projected on the XY plane.

Model	z_{end}	t_{last} [Gyr]	M_* [$M_\odot$]	M_{vir} [$M_\odot$]	M_*/M_{vir}	$M_*/M_{gas,ini}$	$r_{vir,tot}$ [kpc]	$r_{1/2,*}$ [kpc]	b/a_{XY}
HDHM	0.22	11.0	7.5×10^{11}	1.5×10^{13}	0.050	0.26	153.0	15.6	0.56
IDHM	0.77	8.0	7.4×10^{11}	1.5×10^{13}	0.050	0.26	141.8	16.5	0.48
LDHM	0.50	8.7	7.3×10^{11}	1.5×10^{13}	0.049	0.25	133.8	15.8	0.57
VLDHM	0.83	6.6	6.3×10^{11}	1.3×10^{13}	0.048	0.22	112.5	11.2	0.52
HDIM	1.0	5.8	2.0×10^{10}	2.1×10^{11}	0.10	0.45	37.6	5.7	0.62
IDIM	0.75	7.0	1.9×10^{10}	2.1×10^{11}	0.08	0.43	35.7	5.8	0.63
LDIM	0.58	8.1	1.9×10^{10}	2.0×10^{11}	0.10	0.42	33.3	5.2	0.75
VLDIM	0.15	11.8	1.7×10^{10}	1.4×10^{11}	0.12	0.38	28.3	4.9	0.83
HDLm	0.36	9.7	1.5×10^8	3.3×10^9	0.045	0.19	9.2	2.3	0.74
IDLm	0.22	11.0	1.4×10^8	3.3×10^9	0.04	0.16	10.0	2.4	0.67
LDLM	0.05	13.0	1.4×10^8	3.2×10^9	0.04	0.19	11.8	2.1	0.79
VLDLM	0.0	13.7	1.0×10^8	3.0×10^9	0.03	0.10	10.5	2.7	0.65

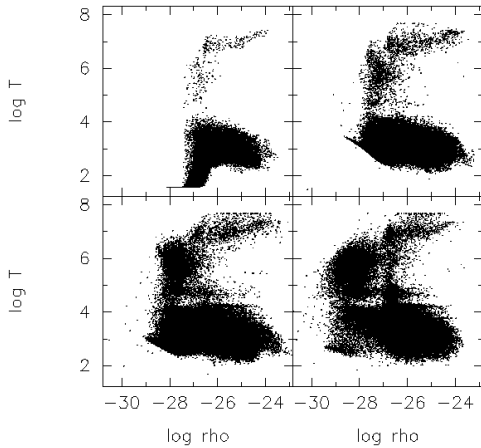


Fig. 9. Phase diagram density ρ (g/cm^3) vs. temperature T (K) of gas particles for the IDIM model, at four stages of evolution ($z \simeq 10.0, 7.0, 5.0, 3.5$). The hot gas decreasing its density is leaving the galaxy, due to stellar feedback.

stars. Due to the strongly peaked density perturbations, all models soon develop a central stellar system; however, in the very first stages of the evolution we note the formation of many small clumps of stars, which soon merge into a single entity. In most cases, additional smaller stellar systems form in regions far from the center because of local concentrations of cold gas, eventually merging onto the major body of the proto-galaxy.

All models finally develop a central dense stellar system, located at the centre of the collapsed halo of DM. Some amount of spurious global bulk motion of the system (due to numerical instabilities) can be observed in some cases (see Fig. 7). At $z = 1$, most of the systems have almost completely relaxed into stable configurations, although in the massive ones pronounced irregular features are clearly visible (arms of star forming regions, diffuse halos of stars around the central object). These are *not* formed by old populations infalling onto the central ob-

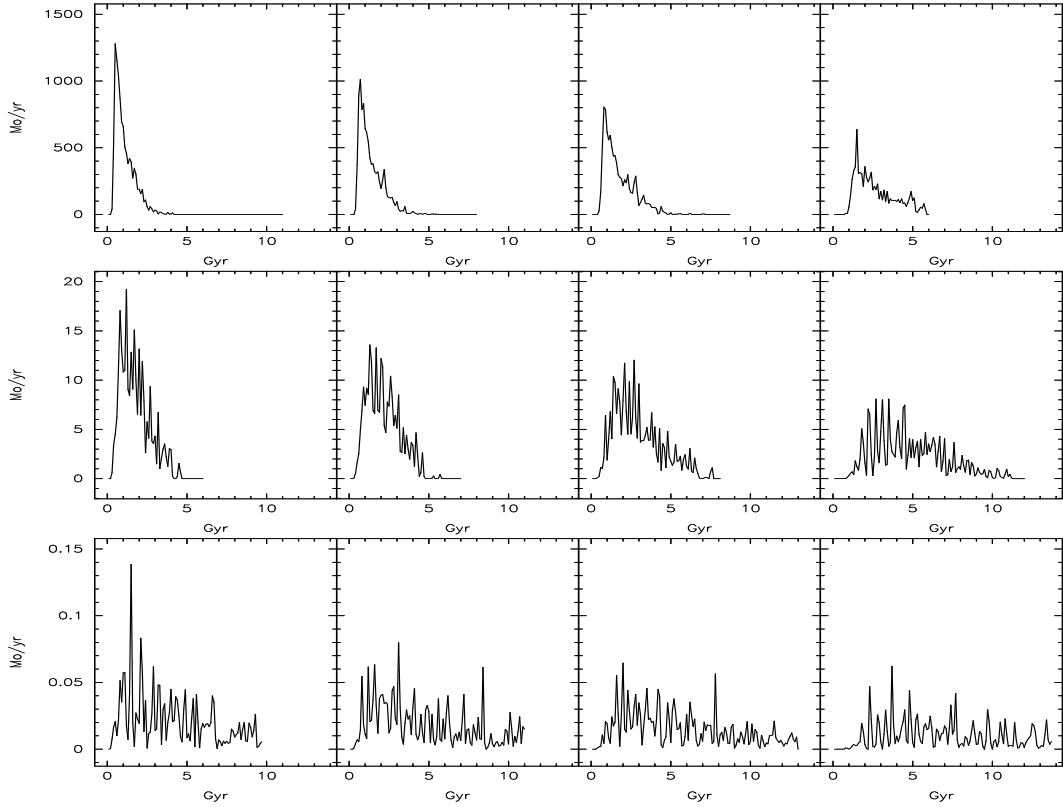


Fig. 10. Star formation histories for the twelve models. Left to right: high density, intermediate density, low density, very low density. Top row: high mass; intermediate row: intermediate mass; bottom row: low mass. SFRs are in $M_{\odot}/\text{yr}$, times in Myr.

ject, but by young SSPs escaping from the galaxy (this can be inferred by checking the birth date of the stellar particles and their velocities). The bright, spherical stellar clump visible in the bottom of the VLDHM model in Fig. 7 has mass $M_* \simeq 10^{10} M_{\odot}$, much larger than the masses of the low mass models. Such stellar clumps must therefore be considered tidal galaxies, formed by gas condensation. Many examples of such systems are known to exist (see e.g. Elmegreen et al. 2007), even if they are generally thought to be generated within interacting systems. In our cases, local condensations of gas, compressed and pushed outwards by the pressure exerted by the inner regions of the system, collapse to form stars while being ejected from the galaxy. The tidal objects are finally engulfed by the massive galaxy, but usually leave a faint expanding trail of debris.

On the other hand, low mass systems are surrounded by extended halos of sparse stellar particles (Fig. 8). Perhaps, these may be identified with the observed intra-cluster light (ICL, Monaco et al. 2006), which could therefore be the result of diffuse star formation within gas escaping from low mass galaxies, rather than the consequence of tidal stripping and/or disruption of bounded systems.

It is worth recalling here that a different choice for the dimensionless efficiency ϵ_{SF} may have lead to different dynamical histories, as discussed below. In particular, a lower efficiency could have reduced the formation of diffuse stellar halos.

All galaxies have triaxial shapes; their axis ratio at $z = 1$ and at the final redshifts, obtained from their projection on the XY plane, are summarized in Tables. 2 and 4. If observed from this angle of view, all the models would be classified as E3 - E5 galaxies in the Hubble system. Obviously, more spherical shapes could be obtained changing the angle of view, and almost all models could be morphologically classified as almost perfect spherical galaxies choosing a suitable projection. Noticeably, the minimal initial rigid rotation can hardly be considered the main cause for the obtained strong triaxiality of the models, which must be therefore a consequence of their dynamical history. In particular, observing the first stages of the formation of these systems it appears that their collapsing motions took place along the filamentary structures of the cosmological over-densities, and stars tend to conserve elongated orbits. This view is strengthened by the fact that reducing ϵ_{SF} , by requiring a higher degree of dissipation and condensation of gaseous particles, yields more concentrated and spherically-shaped structures (see Sect. 5). It would be worth investigating how this would affect to present day structure these systems. Perhaps, in this context, a non-negligible role could also be played by the absence of large scale tidal motions caused by the adopted prescription for the evolution of the whole proto-galaxy (the initial sphere) *in void*.

The injection of energy from stars into the ISM causes turbulence and large scale motions in the gas (see Fig. 3). Depending on the particular circumstances under which

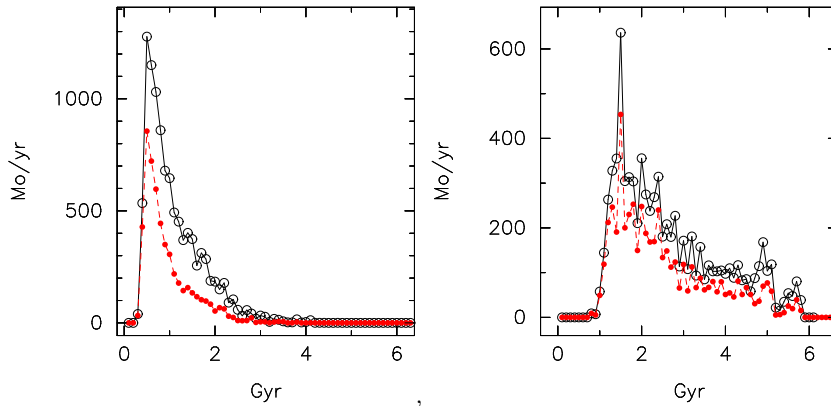


Fig. 11. Star formation histories in the whole simulated galaxies (solid black line, large circles) compared with the corresponding SFHs within the central region ($r_{inner} \simeq 20$ kpc; red dashed lines, large points) for two large mass systems. Left panel: model HDHM. Right panel: model VLDHM.

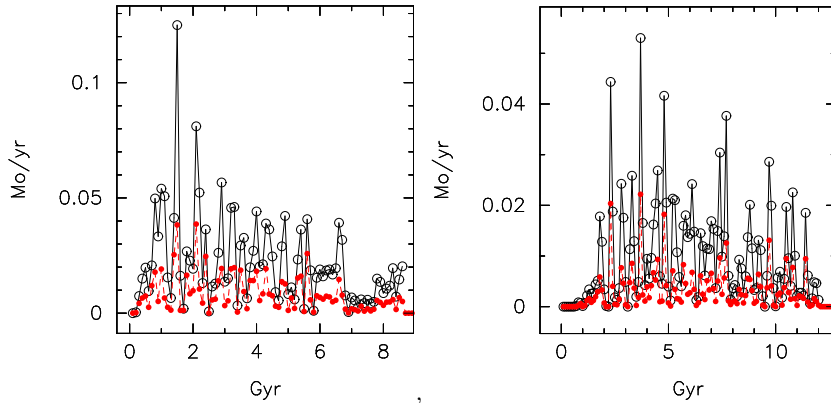


Fig. 12. Same as in Fig. 11, but for two small systems (here $r_{inner} \simeq 2$ kpc). Top panel: model HDLM, bottom panel: model VLDLM.

it takes place, the stellar energy feedback can lead to the ejection of large amounts of gaseous material from the potential well of the galaxy, or to the formation of condensed structures, resulting in new star forming sites. The minor impact of feedback in low mass halos ultimately results in the large differences in their star formation histories, as explained below.

Different phases naturally develop in the gas during the first stages of the galaxy formation as clearly shown by Fig. 9 for the prototype model IDIM). First, the initially cold, expanding gas begins to heat to a typical temperature of $T \simeq 10^4$ K in the central regions, as soon as they detach from the Hubble flow and collapse. When the radiative cooling becomes effective, this gas begins to cool and to form stars. The ejecta from the SSP subsequently heat the surroundings⁴. Now, the infalling gas is forced to interact with these heated regions. If the hot gas is enough, this may lead to expanding shock fronts (see below). Thus, after several Myr of activity, the gas can be roughly split as follows. A cold and dense phase is formed by gas in the central regions of the galaxies, where stars are

forming. On the opposite extreme, hot and thin gas *either* is leaving the halo (galactic wind) after being heated by stellar feedback, *or* it is infalling onto the central object, after being shock-heated because of the interaction with the central, hot regions. Two more phases are clearly detectable: the hot *and* dense gas, where the stellar feedback is heating the gas at very high temperatures before these regions expand under the action of the local pressure, and the cold but thin gas belonging to the cosmological cosmic web (before re-ionization takes place, or cooling after re-ionization; see Sect. 2.8).

3.2. Star formation histories

Looking at Fig. 10, it is clear that the initial conditions of each halo deeply influence the star formation history (SFH) of the parent proto-galaxy. Even at a first glance, two clear trends can be easily identified.

First: *the more massive the halo, the more peaked is the SFH*. All massive haloes ($M_{tot} \simeq 10^{13} M_{\odot}$) show a single, strong burst at very early times, followed by a slow decrease down to a substantial halt. A weak tail of activity persists down to later times in the less dense systems, thus accounting for a weak dependence on its density (or environment). Smaller halos, in turn, display a more

⁴ Once more, it is worth pointing out that heating of the surrounding gas is due to the mechanical pressure exerted by the stellar particles, and *not* to a redistribution of their thermal budget; see Sect. 2.5.

Table 4. Properties of the star formation history for the twelve models. Left to right: maximum peak of star formation rate, epoch of maximum activity [Gyr], percentage p_* of the final stellar mass assembled at four different redshifts, and epoch at which different fractions f of the final stellar mass (i.e. at $z = 1$) has formed [Gyr] (with corresponding redshift).

Model	$SFR_{max} [M_{\odot}/yr]$	t_{SFRmax}	$p_{*,z=10}$	$p_{*,z=5}$	$p_{*,z=2}$	$p_{*,z=1.5}$	$t, z_f=1/2$	$t, z_f=99/100$
HDHM	1380	0.5	14.9	67.3	99.5	100.0	0.863 6.5	3.029 2.2
IDHM	1020	0.6	2.9	55.2	98.0	100.0	1.097 5.4	3.778 1.8
LDHM	800	0.9	<0.1	39.7	93.2	98.5	1.417 4.4	4.398 1.5
VLDHM	350	1.5	0.0	5.2	67.2	80.1	2.423 2.7	5.083 1.2
HDIM	19.0	1.2	2.8	35.5	92.2	98.9	1.589 4.0	4.392 1.5
IDIM	13.5	1.5	0.1	19.9	86.0	98.3	1.976 3.3	4.530 1.4
LDIM	12.0	2.5	<0.1	9.5	69.4	85.9	2.568 2.6	5.806 1.0
VLDIM	8.0	2.5	0.0	1.1	42.8	66.3	3.537 1.9	5.837 1.0
HDLM	0.135	1.5	2.3	17.9	61.5	76.1	2.78 2.4	5.94 1.0
IDLm	0.075	3.0	0.6	10.2	49.1	69.6	3.48 1.92	5.85 1.0
LDLM	0.06	2.0	0.2	6.2	54.9	79.2	3.00 2.23	5.80 1.0
VLDLM	0.55	3.8	0.0	4.8	25.2	46.0	4.40 1.47	5.800 1.0

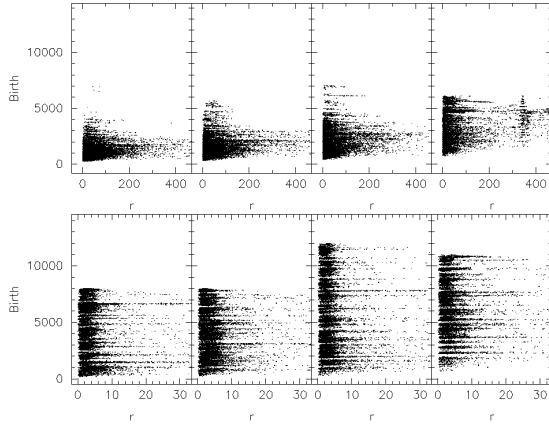


Fig. 13. Age of the stars (in Myr) as a function of their radial positions (distance from the barycenter in kpc) for the high and low mass models (left to right: decreasing density; top to bottom: decreasing mass), at the final time-step of their evolution (the same age as in Fig. 7). The virial radii of the stellar systems are similar to the extension of the X axis.

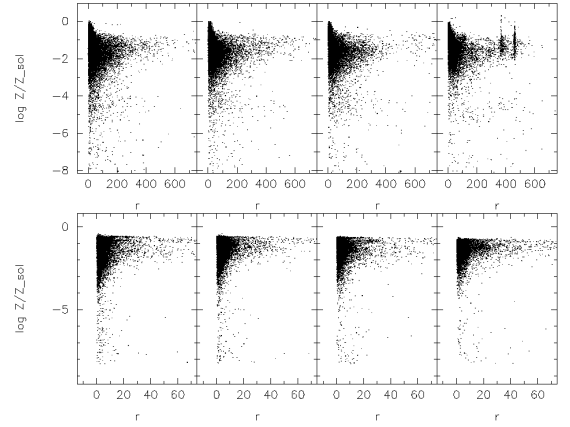


Fig. 14. Metallicity of the stars as a function of their radial positions for the high and low models (left to right: decreasing density; top to bottom: decreasing mass), at the final time-step of their evolution (the same age as in Fig. 7). The metallicity is expressed as the logarithm of Z ; for reference, $\log(Z_{\odot}) \simeq -1.72$. The radial distance from the barycenter of the stellar system (kpc).

fragmented and prolonged star formation, often alternated with periods of quiescence.

Second: *lower densities correspond to delayed activity (the first burst starts later) and minor overall efficiency (the peaks of activity have lower magnitude)*. A delay up to $\simeq 1$ Gyr in the beginning of the SF activity can be seen in the low density models. Moreover, the maximum rate is generally halved, and the activity is prolonged (this second effect is of crucial importance in intermediate mass halos, $M_{tot} \simeq 10^{11} M_{\odot}$). Apparently, the mass of the halo is the dominant characteristic to determine its evolution, but intermediate mass halos can have very different stories depending on their global over-density.

Table 4 summarizes the properties of the SFHs, some of which are also graphically displayed in Fig. 15. All the models begin forming Population II stars between $z \simeq 20$ and $z \simeq 7$. The HDHM halo experiences the strongest and earliest peak of activity, reaching some $1.5 \times 10^3 M_{\odot}/yr$ at $z \simeq 10$. More than 50% of its total stellar mass is assem-

bled *before* $z \simeq 6$. If observed now from the Earth, this galaxy would appear massive ($M_* \simeq 3 \times 10^{11} M_{\odot}$) and essentially red at $z \simeq 4$. Its tail of SF activity continues down to $z \simeq 1.6$, contributing 19% of the total stellar mass. Looking at less dense halos of the same mass, it can be noticed how the maximum of activity decreases in magnitude and is delayed to later times, down to a peak of $7.5 \times 10^2 M_{\odot}/yr$ at $z \simeq 5$ in the VLDHM model, where moreover an intense activity is prolonged to much later epochs. Despite the similarity between the final objects formed from massive halos, a clear trend can be seen in Fig. 15: the lesser dense the halo, the later the mass assembly takes place (with the VLDHM halo assembles half of its total stellar mass at $z \simeq 2.7$). The overall efficiency of the star formation process in these models is 20%-30%.

Intermediate mass systems apparently have more complex histories. Moving from denser to less dense systems, the shape of the star formation curve changes dra-

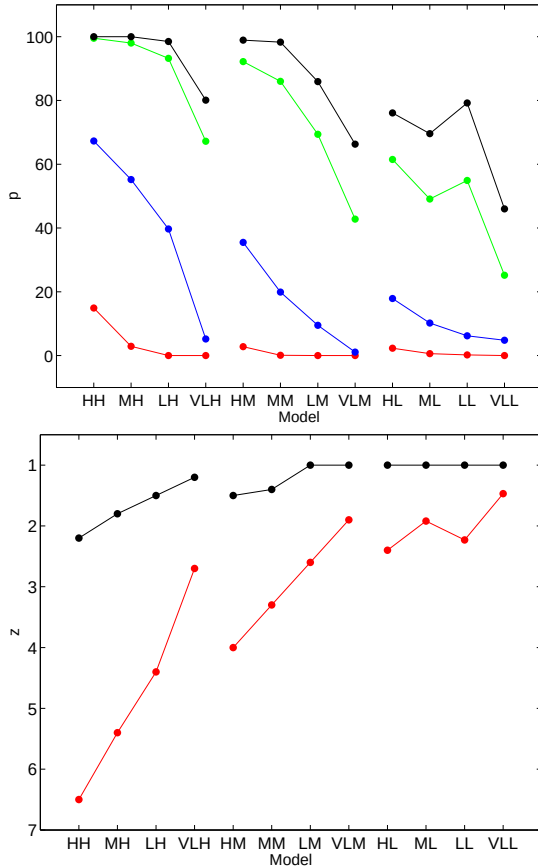


Fig. 15. Evolution of the mass assembly in the twelve models. The X axis lists the models ordered as in Tab. 4 (the acronyms have been shortened for reasons of space; however, in each entry the first letter(s) refer to the density, the last letter to the mass) so that each set of four point linked with a solid line shows the data for four equal mass models with decreasing density moving from left to right. Top panel: the Y axis displays the percentage p of assembled stellar mass at a given redshift z with respect to the $z = 1$ stellar mass; red: $z = 10$; blue: $z = 5$; green: $z = 2$; black: $z = 1.5$. Bottom panel: the Y axis displays the redshift z at which a given percentage p of the total stellar mass at $z = 1$ was assembled; red: $p = 50\%$; black: $p = 99\%$.

matically. Strongly over-dense halos undergo an intense burst at early times, much as the most massive systems. However, less dense halos gradually display more prolonged, uneven, bursty histories, with many different peaks. Apparently, in this range of masses the initial density plays a dominant role in deciding the fate of the mass assembly of the galaxy. The efficiency of the star formation process is very high in these cases: 30% - 50% of the initial gas is converted into stars.

All low mass systems show a continue, bursty SFH, sometimes with long periods of quiescence. In these systems, the feedback from stars is sufficient to halt the star formation process soon after its beginning. However, this means that as soon as the first stars cease their strong feedback activity, the gas is able to recollapse and form new generations of stars. This galactic “breathing” leads to systems in which many different generations of stars are

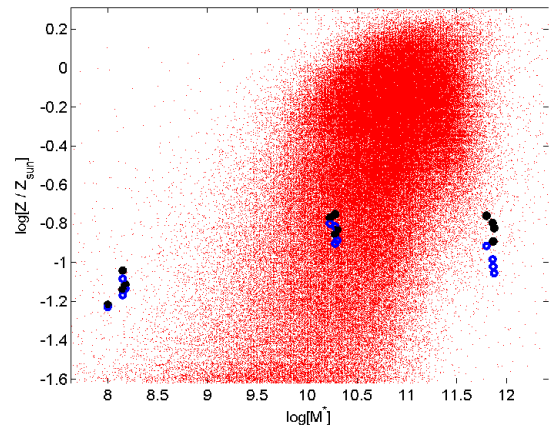


Fig. 16. Averaged metallicity $\log[Z/Z_{\odot}]$ versus stellar mass $\log M^* [M_{\odot}]$ for the twelve models, compared to the observed mass-metallicity relation. Red points: data from Gallazzi et al. (2005). Blue open circles: averaged metallicity of all stellar particles in each model. Black points: averaged metallicity within the half light radii of the models.

present. The overall efficiency of the star formation mechanism is lower in this cases, with 10-20% of the initial gaseous mass turned into stars (giving high M/L ratios, comparable to those of many observed dwarf galaxies).

Figs. 11 and 12 compare the SFH of four models (HDHM, VLDHM, HDLM, VLDLM) *across the whole systems* with the SFH *within their central regions* (that is, within the central 20 kpc in massive systems and within the central 2 kpc in the small ones). It can be noticed that massive systems build their central regions before their outskirts, while in less massive galaxies the SF process is active both in the inner and in the outer regions during the whole lifetime of the galaxies. This is consistent with some recent observational findings (van Dokkum et al. 2010) on the way galaxies are built-up.

3.3. Stellar populations: ages and metallicities

The distribution of the galactic ages ($T_{G,i}$ at which the star particles are created as a function of the radial distance is shown in Fig. 13 limited to the high and low mass models (the behaviour of the intermediate mass ones falls in between these two). The real star particles ages are $T_* = T_G - T_{G,i}$. In early epochs, the stars are preferentially created in the central regions, then the star forming activity expands to larger radii (inside-out mechanism), and moving towards the present time, the stellar activity tends to shrink again towards the centre. This simply mirrors the star formation history and the mechanism of mass assembly presented above.

Fig. 14 shows the metallicity of the stellar components in the high and low mass models, as a function of their radius. The overall values resemble those of observed galaxies. The majority of the stars in the central regions of the massive systems have higher metallicity than those in the outskirts (strong negative gradients); on the contrary, in less massive systems the metallicity of the central stars is

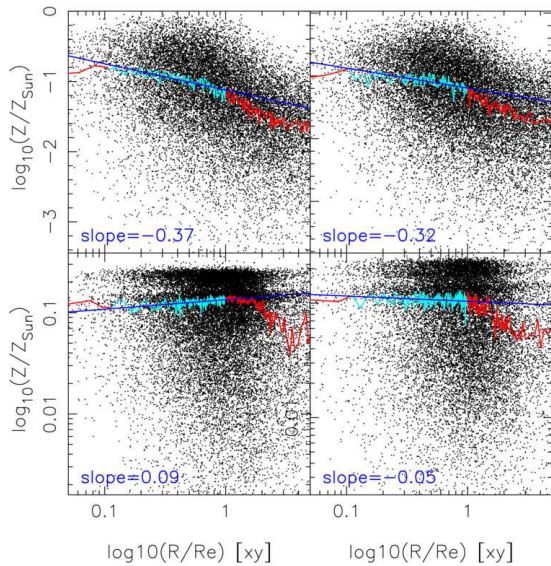


Fig. 17. Gradients in metallicity. Left to right, top to bottom: models HDHM, LDHM, HDLM, LDLM at their last computed snapshot. Plotted is the metallicity $\log[Z/Z_{\odot}]$ versus the fractional radial distance r/r_{eff} of all particles. Also shown are the median-binned metallicity profile, the number of particles being the same in each bin (blue and red solid lines, referring to the inner and outer region, respectively), and the linear best fit (black solid line; the slope is written in the left bottom corner of each panel). See text for details.

generally equal to that of the peripheral ones. Interesting features to note are (i) on the average, the metallicity distribution of the star particles get narrower at increasing radial distance; (ii) the maximum values are always reached in the centre, (iii) great deals of stars are made of recycled gas; (iv) tidal satellites have their own chemical history (top right panel of Fig. 14).

We also look at the average stellar metallicity of the models. The results are shown in Fig. 16 where the blue dots refer to all stellar particles, whereas black dots only to those within the effective radius of each model. The small red dots are the data by (Gallazzi et al. 2005). In general, the metallicity increases with the galaxy mass, but for the most massive systems for which the effect tends to level off or even to slightly decrease. This same behaviour was already pointed out by Chiosi & Carraro (2002) and was interpreted as due to combined effect of star formation efficiency and galactic winds. This theoretical prediction resembles the trend shown by the brightest (likely most massive) galaxies studied by Tremonti et al. (2004). Furthermore the theoretical data suggest that the massive systems are more enriched in the central regions, compared to intermediate and low mass models. The mean values are in general consistent with those measured in the early type galaxies by Gallazzi et al. (2005). However, the observational data spread over much larger intervals of metallicities that the model cannot recover. Furthermore the mean values of theoretical predications are lower (at least by one dex) than the corresponding ones indicated by the data. Part of the discrepancy may arise because of

the different methods and stellar population ingredients used to obtain observed and theoretical metallicities. For example, the use of the Kroupa IMF, which is tailored to fit the solar vicinity, may cause an underestimation of the enrichment by massive stars. Moreover, the high star formation efficiency can again play a role, favouring the formation of low metallicity stars at early times and reducing the average metallicity of the models. Finally, as shown by Scodreggio (2001), at least some part of the observed color- (and hence metallicity-) mass relation might just be a spurious consequence of an aperture effect, as observed colors (and metallicities) are usually measured within a fixed aperture, rather than a given fraction of the galaxy effective radius. Despite these drawbacks and the need of a better description of the chemical enrichment in our models for early type galaxies, the theory and observational data are considered to reasonably agree with each other.

3.3.1. Metallicity gradients

Radial gradients in spectro-photometric properties of ETGs are known to exist (see e.g. La Barbera et al. 2010, and references therein). The majority of ETGs in the local Universe features red cores (La Barbera & de Carvalho 2009), and less frequently blue cores, mostly found at low mass (Suh et al. 2010). The radial gradients are interpreted as age and metallicity gradients of the constituent stellar populations (e.g. ?).

In Fig. 17 we plot the metallicity of each particle as a function of projected distance to the galaxy center (from left to right, and top to bottom, models LDHM, HDHM, LDLM and HDLM are displayed). For each model, the median-binned metallicity profile is shown, the number of particles being the same in each bin. In order to estimate the metallicity gradient, we performed an orthogonal least-square fit of the metallicity profile in the range $0.1 r_{eff}$ to $1 r_{eff}$, i.e. the same range used in observational studies (Peletier et al. 1990). The fitting was repeated for 100 different two-dimensional projections, the average slope of the best-fitting lines giving the metallicity gradient, $\nabla_Z = d \log Z / d \log r$, of a given model. We repeated the computation of ∇_Z by applying different weighting schemes, where each particle was assigned no-weight, mass-weighted, and luminosity-weighted, finding no significant differences within the errors, especially at high mass (see Tab. 5). High-(Low-) mass models exhibit metallicity gradients spanning the range from -0.4 (-0.02) to -0.27 (0.16), depending on the weighting scheme and environment. In general, low-density models tend to have more negative metallicity gradients. Moreover, at given density, high- (relative to low-) mass systems have more negative gradients.

The above predictions for ∇_Z can be compared to recent observational results. A steepening of the metallicity gradient with galaxy mass, qualitatively consistent with that seen in our models, has been reported by La

ID (1)	∇_Z (no-weight) (2)	∇_Z (mass-weighted) (3)	∇_Z (lum-weighted) (4)
LDHM	-0.39 ± 0.04	-0.40 ± 0.04	-0.33 ± 0.03
HDHM	-0.32 ± 0.04	-0.31 ± 0.04	-0.27 ± 0.03
LDLM	0.00 ± 0.04	$+0.01 \pm 0.02$	0.00 ± 0.06
HDLM	$+0.09 \pm 0.02$	-0.02 ± 0.01	0.16 ± 0.03

Table 5. Metallicity gradients of four ETG models in the radial range of $0.1 r_{eff}$ to $1 r_{eff}$. Column 1 is the model ID label. Columns 2, 3, and 4, correspond to cases where the ∇_Z is computed with no-weight, mass-weight, and luminosity-weight assigned to each particle. Errors are the rms of ∇_Z estimates among 100 projections of each model.

Barbera et al. (2010), and, independent of galaxy environment, by La Barbera et al. (2011). On the other hand, other observational studies, based on smaller samples, do not find a clear correlation of ∇_Z with mass (Spolaor et al. 2009), or even a double-value behavior (Tortora et al. 2010). For a galaxy mass of $3 \times 10^{11} M_\odot$ (similar to that of $\sim 7 \times 10^{11} M_\odot$ of models HDHM and LDHM), La Barbera and collaborators found metallicity gradients of -0.37 ± 0.02 and -0.41 ± 0.02 for low- and high-density ETGs, respectively. At low density, the estimate of is fully consistent with our model predictions (for no-weight and mass-weighted ∇_Z 's). At high-(relative to low-) density, the models exhibit shallower gradients, by ~ 0.07 , in contrast to the data, where a (marginal) difference of -0.04 ± 0.03 is found. For what concerns low-mass models, they exhibit almost null gradients, consistent with the observational finding of Spolaor et al. (2009) that the ∇_Z vanishes at a galaxy mass of $\sim 10^9 M_\odot$. Whilst these results point to a fair agreement between our models and (some) available data-set, we postpone a more thorough investigation of both metallicity, age, and color gradients, as well as their correlation with other model properties, to a forthcoming contribution.

3.4. Cold and hot gas accretion and ejection (galactic winds)

Fig. 18 shows the gas density vs. temperature diagram for the high and low mass models of our set at $z = 2$ (the behavior of the intermediated mass ones falls in between these two). As expected from theoretical considerations (Rees & Ostriker 1977), a bimodal gas accretion can be observed in our models, as a function of the total mass of the proto-galactic halo. In low mass halos, the gas accretes from the cosmic web without being shock heated to the virial temperature, and proceeds to flow along filaments towards the center of the halo, where it will eventually shock. In contrast, massive halos soon shock the infalling gas to very high temperatures, suppressing the formation of local substructures and forming a central distribution of hot gas. Halos which do support shocks close to the virial radius are expected to contain a quasi-hydrostatic atmosphere of hot gas. Birnboim & Dekel (2003) claim that shocks can only form close to the virial radius in halos with mass greater than $10^{11} M_\odot$ for primordial gas

(or around $10^{12} M_\odot$ for gas of Solar metallicity), in good agreement with our models.

Looking at Fig. 18, it can be noticed that the situation shows a strong dependence on the mass of the halo. In massive and dense systems, almost no cold gas ($T < 10^4$ K) is present, so the star formation cannot proceed further. Moreover, an expanding shock front is evident (most of the gas is gathered in a narrow density strip moving outwards at high temperatures); this will result in a *hot* galactic wind. The hot but thin gas is being shocked during its infall, in a hot accretion mode. The low density, cold tail is formed by gas expanding and never collapsing onto the galaxy, cooling down after being photo-heated by reionizing radiation at $z = 9$ (see Sect. 2.8). More cold gas is present in less dense systems, both in high density regions (potentially forming new stars) and in the outskirts, the shock front being less pronounced. A large amount of dense ($\rho \simeq 10^{-25}$ g/cm³) and hot ($T > 10^6$ K) gas is also present in massive systems which are still forming stars, due to the action of feedback.

In the intermediate mass systems, a large amount of cold and dense gas is still present, ensuring a continuing star formation. In low mass systems, almost no gas is heated to $T > 10^5$. Thus, the gas is ejected from the galaxy at low temperatures, and no shock front develops.

Figs. 19 and 20 plot the state of the gas in two reference models (namely, the HDHM and the LDLM models; see the caption for more details), in terms of radial velocity at different epochs, with respect to the escape velocities of the systems. The mass of the ejected gas increases with time in all the models. The average velocity is proportional to the mass of the system, with massive galaxies generating winds at $\simeq 2000$ km/s, and low mass galaxies at about one tenth of the speed. Interestingly, the average radial velocity is proportional to the distance, like in a Hubble-like flow; the ejection of the gas is therefore an explosive phenomenon, rather than a slow outflow. However, looking more carefully it can be noticed that there are different linear velocities at the same radial distances, meaning that many explosive events have taken place. There is also a huge mass of gas moving outwards at slower speed, at late times (particularly in the HDHM model).

In Fig. 24 we plot fractional amounts of escaping gas as a function of time, for four of the models (namely, HDHM, LDHM, HDLM, LDLM; the other models show similar behaviours). We only plot the trends *after* the systems have

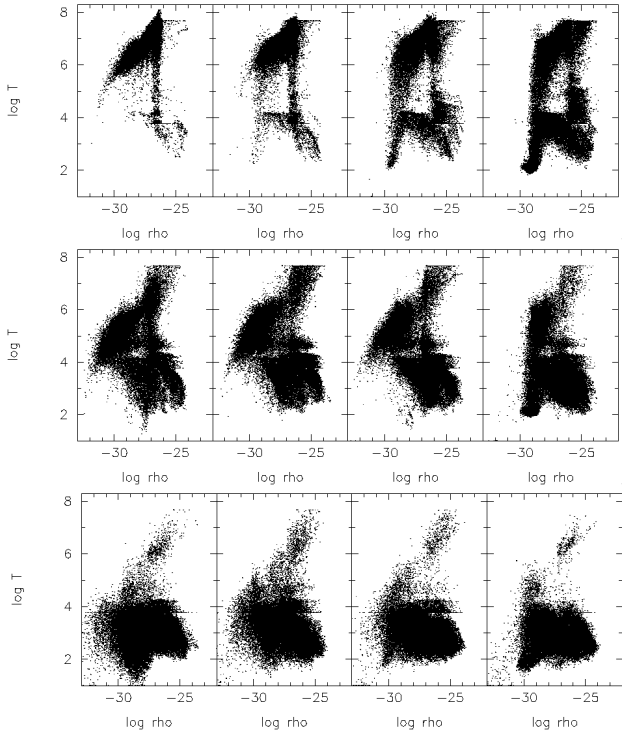


Fig. 18. Phase diagram density ρ (g/cm^3) vs. temperature T (K) of gas particles for the twelve models, at redshift $z = 2$. In each panel: left to right: high density, intermediate density, low density, very low density; top row: high mass; intermediate row: intermediate mass; bottom row: low mass.

virialized; in other words, the total gravitational energy must have become constant to avoid the initial phases of galaxy assembly during which the gas accumulates toward the center and the galaxy is subject to episodes of quick expansions and/or contractions. The red dotted lines display the fractional mass of gas which is found *outside* a given radius r_{ext} from the centers of the systems at the given time. For massive systems we adopt $r_{\text{ext}} = 200$ kpc, while for small systems $r_{\text{ext}} = 10$ kpc (these values are similar to the virial radii of the models, see Tab. 4). The blue dashed lines are relative the fraction of gas which has an outward-directed radial velocity, and is moving faster than the Hubble flow at the correspondent redshift. Finally, the black solid lines are the fraction of gas which is also above the escape velocity of the system. Clearly, two different regimes are at work. In the high mass realm, the fraction of gas which is *really* leaving the galaxy is only a few percent. On the contrary, in low mass systems more than half of the total gas has sufficient velocity to escape. This is easily understandable. Assuming constant density in models of different total masses, the gravitational potential energy of a system is proportional to $M^{5/3}$, while the energy injection from SN explosions is proportional to the number of stars in the galaxy, i.e. roughly $\propto M$. Thus, the binding energy of a system grows more rapidly than the kinetic energy of its components, making it more difficult to escape from the system. However, this distinction is theoretical, and what we observe in the real Universe is simply the

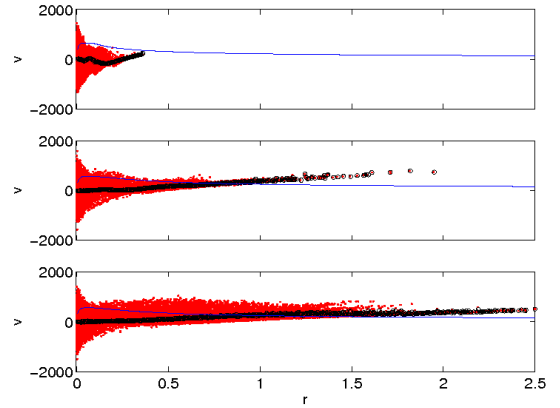


Fig. 19. Galactic winds in the HDHM model. Red points: radial velocities of gaseous particles (km/s) against the distance from the barycenter of the systems (Mpc). Black circles: mean radial velocity in the spherical shell at radius r . Solid blue line: escape velocity as a function of the radius. Top to bottom: $z = 4.4, 1, 0.2$.

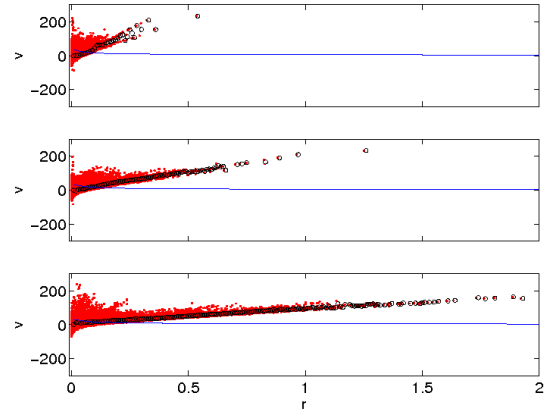


Fig. 20. Galactic winds in the LDLM model. Symbols as in Fig. 19. Top to bottom: $z = 2.2, 1, 0.05$.

mass of gas that can be found outside a galaxy, moving outwards (i.e., the blue dashed line in the diagram). In this case, the situation is much more similar between the two cases, however still with a higher efficiency in the low mass regime. The escaping gas is chemically enriched, as illustrated in Fig. 25 for the HDHM model.

3.5. Critical discussion of the results

Is the strong dependence of the SFH on the initial galaxy mass and density a physical result, or is it just a spurious numerical effect?

The large difference in particle masses between the various simulations, a necessary choice to model galaxies spanning a wide range of sizes, is the major reason for this serious uncertainty. In high mass models, the gas particles have a mass $m_{\text{HM}} \simeq 5 \times 10^7 M_{\odot}$, whereas in the low mass models the mass of a gas particle is $m_{\text{LM}} \simeq 1.2 \times 10^4 M_{\odot}$, i.e. a factor of $\simeq 4 \times 10^3$ times smaller. Does this difference in mass resolution affect the global results of our study? To check this issue, we run three low resolution models.

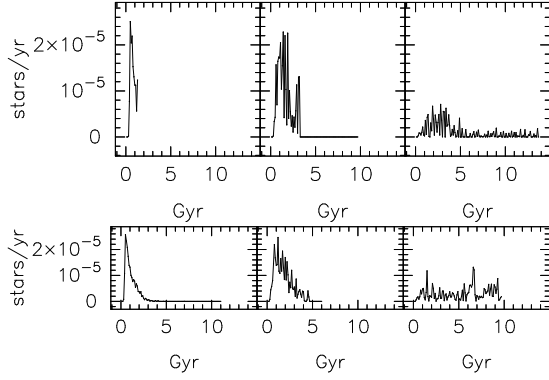


Fig. 21. Star formation histories for the three low resolution models, lrHDHM, lrHDIM, lrHDLM (in the bottom panels the corresponding SFH for the high resolution models are plotted for comparison). The rates have been normalized to the mass of the particles.

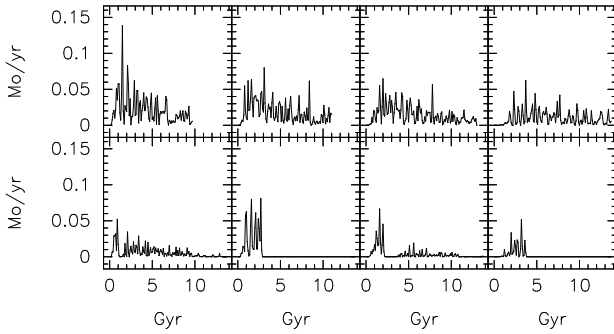


Fig. 22. Star formation histories for the low mass models, without (top panel) and with (bottom panel) imposing a minimum softening and smoothing length.

The initial halos for these tests were crudely obtained from the three high density models HDHM, HDIM, and HDLM, by simply considering one particle out four when reading the input files, and contextually multiplying the particle masses by a factor of four. Although this procedure is not rigorous, still it can be safely used to get a quick insight of the problem. The new models shortly indicated by lrHDHM, lrHDIM, lrHDLM have the total number of particles reduced by a factor of 4 and can be calculated on a single CPU. Their star formation is compared to that of their “parent” models and the results are plotted in Fig. 21. Even if some minor differences can be noticed, it is evident that the global trends are quite similar at the different resolutions. This is reassuring, implying that we can trust the results of our reference models, at least as far as their different star formation histories is concerned. This also shows that meaningful results can be obtained even using a small number of particles with much less numerical effort.

Finally, we consider the effects of imposing constant minimum threshold values to the softening and smoothing lengths of particles. To do so, we re-run the low mass models imposing minimum values $\epsilon_{min} = h_{min} = 100$ pc. We plot the SFHs of the resulting models in Fig. 22, and compare them with the SFHs of the same models belong

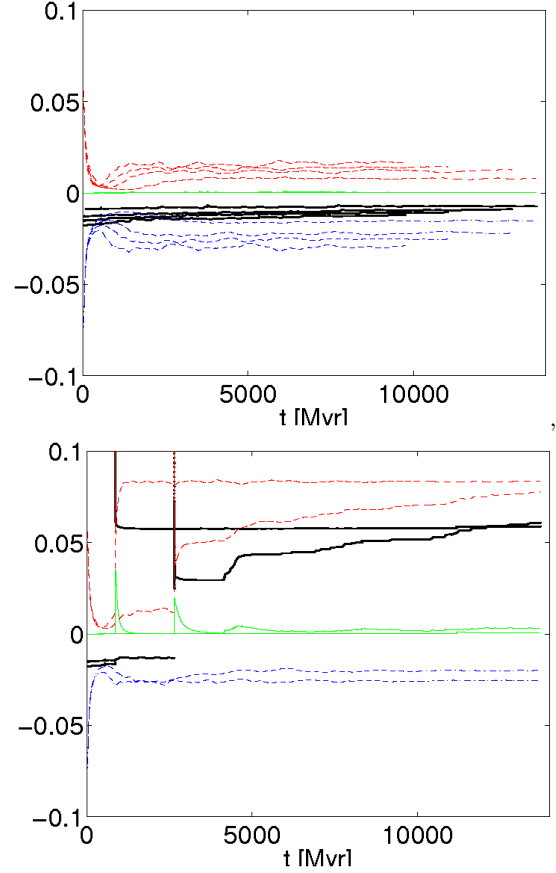


Fig. 23. Energy trends in the low mass models. In each panel the global values of the energies (in code units) are plotted versus time, superposing the results of different models. Left panel: models without minimum softening/smoothing lengths; right panel: models with minimum softening/smoothing lengths ($\epsilon_{min} = h_{min} = 100$ pc). Red dashed lines: kinetic energy; blue dashed lines: gravitational potential energy; green solid lines: thermal energy; black lines: total energy.

to the reference set in which no limitations have been imposed to the softening and smoothing lengths. Clearly, there are non-negligible differences. The global efficiency in the star formation process is lower in the models including the threshold. Moreover and perhaps even more important, the dynamical evolution is substantially different, because in these models star formation is essentially halted after a strong peak of activity, whereas in the reference models it proceeds at nearly constant efficiency. To cast light on this issue we look at the temporal variation of the kinetic, gravitational potential, thermal, and total energies of the models under consideration. They are plotted in Fig. 23. It is clear that imposing the limit values to the softening and smoothing lengths causes an uncorrect behaviour at the epoch of the strong ejection of hot gas following the first intense burst of star formation, when the dynamical stability of the system is put in troubles. The model response is not physically adequate to drive the correct dynamical behaviour of the whole system. This means that the wrong choice of these parameters can deeply af-

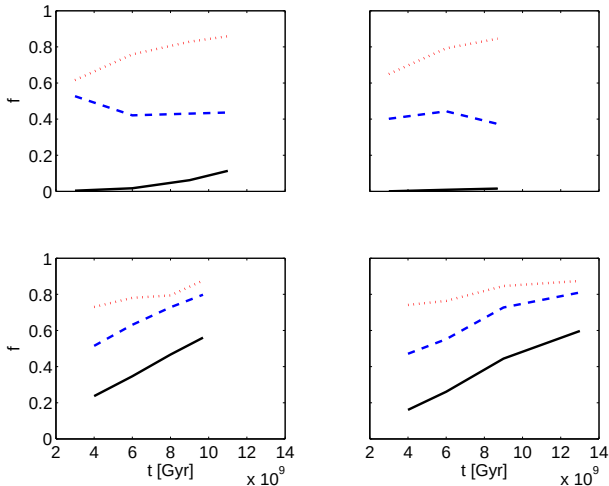


Fig. 24. Fractions of gas leaving the systems as a function of time. Top panels, left to right: HDHM, LDHM models. Bottom panels, left to right: HDLM, LDLM models. Red dotted lines: fractional mass of gas found outside r_{ext} from the centers of the systems at the given time. Blue dashed lines: fraction of gas which has an outward-directed radial velocity, and is moving faster than the Hubble flow. Black solid lines: fraction of gas which also has radial velocity above the escape velocity of the system. See text for more details.

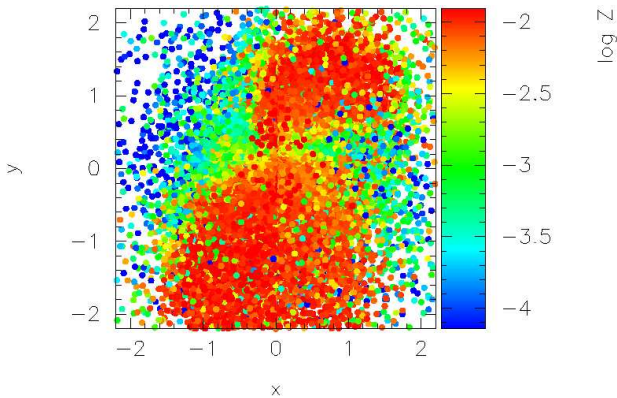


Fig. 25. Metallicity of the ejected gas (wind) in the HDHM model at $z_{fin} = 0.22$. Chemically enriched gas (red particles), with $Z \simeq Z_{\odot} \simeq 0.019$, is leaving the galaxy (which is located near the origin). Note the different metallicity, one or two orders of magnitudes lower, of the gas particles in the outskirts (green and blue): these do *not* belong to the “galactic wind”, since they haven’t been ejected from the galaxy; they are part of the “cosmic web”.

fect the overall evolution of the model. Variable softening and smoothing lengths are therefore highly recommended. Nevertheless, a great deal of the general behavior of the reference models (in particular the gross features of the star formation history) still survives also in those with the constrained softening and smoothing lengths, thus securing that even with wrong choices for these two parameters we catch the essence of the problem.

4. Mass density profiles

The geometrical structure of the model galaxies is best illustrated by the surface and volume density profiles. To compare these we choose a certain evolutionary stage, namely the model galaxies at a certain redshift, i.e. $z = 1$. Fig. 26 displays the surface density radial profiles projected on the XY plane at such redshift. Over-plotted is the best Sersic (1968) profile,

$$\sigma_S(r) = \sigma_0 \times e^{(0.324-2m)\left[\left(\frac{r}{R_e}\right)^{1/m}-1\right]} \quad (7)$$

where R_e is the effective radius of the galaxy, as defined e.g. in Hernquist (1990), σ_0 is the surface density at R_e , and m is the *Sersic index* (if $m = 4$ the de Vaucouleurs profile is obtained). All profiles are computed starting at 0.2% of the virial radius of the galaxies to avoid the very central region of a galaxy. The best-fitting Sersic index is $m \sim 4$, $m \sim 1.5$, and $m \sim 2.5$, for high-, intermediate-, and low-mass models, respectively. In other words, high-(relative to low- and intermediate-)mass models tend to have higher m , in qualitative agreement with the existence of a luminosity-Sersic index relation of ETGs (Caon et al. 1993). However, one should notice that the most massive ellipticals in the local Universe tend to have $m \sim 8$ (e.g. Ferrarese et al. 2006), while we find $m \sim 4$. Moreover, our intermediate-mass models have somewhat lower m than the low-mass ones, which is only marginally consistent with luminosity- m relation, considering its large scatter.

The overall agreement between the models and the Sersic curves is good in the external regions. However, a clear departure from the expected fits is evident in all models at small radii (some fraction of the effective radius, indicated with the black diamond shown in each panel). The central regions of the model galaxies tend to *flatten out* with a plateau in the mass density profile. Given the adaptiveness of the force softening, this feature can hardly be ascribed to numerical artifacts. The close scrutiny of the problem, suggests us ascribe it to the high value of the dimensionless efficiency, $\epsilon_{SF} = 1$, that we have adopted for the star formation rate.

This conclusion is strengthened by the three-dimensional mass density profiles shown in Fig. 27. These profiles are computed starting at 0.05% of the virial radii, and they are compared with the best fits of the Hernquist (Hernquist 1990) and (Navarro et al. 1996) profiles (NFW), for stars and DM, respectively. While DM halos smoothly follow the expected NFW profiles, the density of the stellar component never exceeds that of the DM, and the central regions of the model galaxies are dominated by the DM, contrary to the expectations (e.g. Tortora et al. 2009; Padmanabhan et al. 2004, but see also Grillo, 2010).

The relations between the efficiency of star formation and the gas density at which star form is straightforward. If star formation proceeds slowly (low values of ϵ_{SF} , the gas can get high densities before it gets consumed by star formation itself. The opposite if star formation proceeds

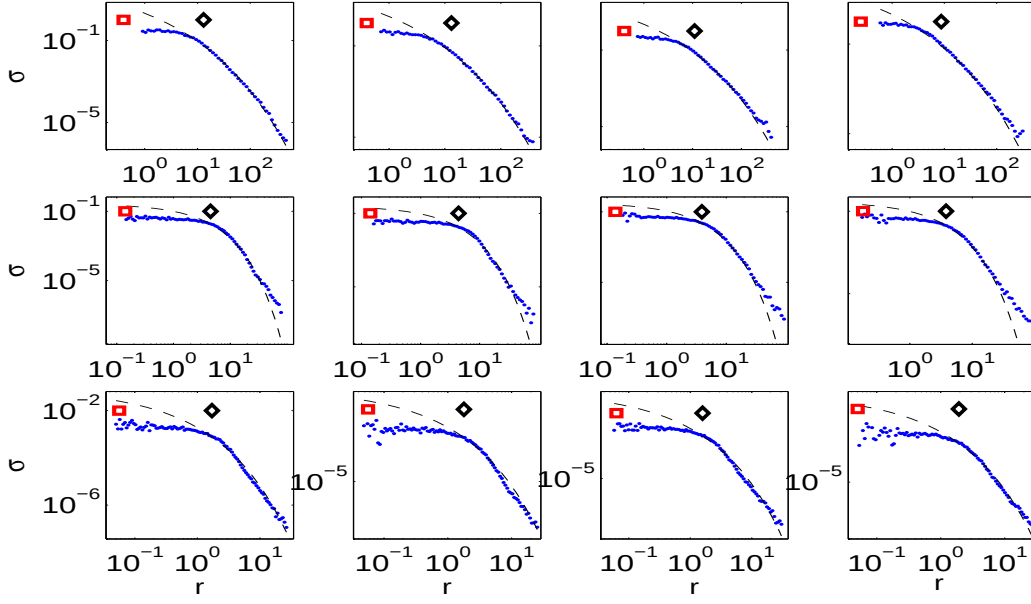


Fig. 26. Stellar surface density profiles for the twelve models at the final time-step of each simulation. Plotted is the projected surface density σ [g/cm^2] against the logarithmic radius [kpc]. Blue dots: stellar particles radial averages; dashed line: best Sersic fit (see text for details). The black diamonds are placed at the effective radii R_e . The red squares are placed at the smallest value reached by adaptive softening lengths, thus indicating the effective resolution of the simulations. Left to right: high density, intermediate density, low density, very low density. Top row: high mass; intermediate row: intermediate mass; bottom row: low mass.

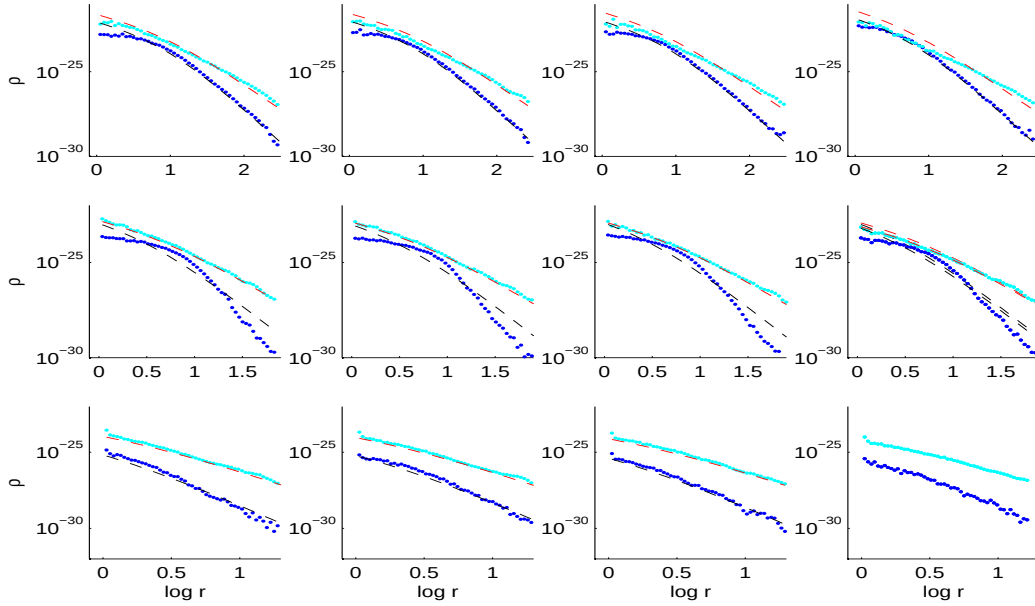


Fig. 27. Density profiles for the twelve models at $z \simeq 1$. Plotted is the density ρ [g/cm^3] against the logarithmic radius [kpc]; blue dots: stellar particles; cyan dots: DM particles; dashed lines: best Hernquist and NFW fits. Left to right: high density, intermediate density, low density, very low density. Top row: high mass; intermediate row: intermediate mass; bottom row: low mass.

at high efficiency, high values of ϵ_{SF} . As a consequence there is an immediate correlation between star formation and the dynamical behavior of a galaxy. Allowing gas particles to turn into stars as soon as they fulfill the required numerical criteria (that is, with unit efficiency) will generally result in quite diffuse stellar systems, with shallow potential wells and large effective radii. On the contrary,

a low star formation efficiency leads to dense clumps of cold gas which give birth to more compact stellar clouds (and to a delay in the beginning of the activity).

Interestingly, this puts a constraint on the modality in which gas collapses and forms stellar systems. Historically, there are two possible and competing theoretical scenarios to describe the assembly of the stellar mass of an ellipti-

cal galaxy (excluding the possibility of merger events). In the first one, stars originally form far away from the effective radius of the galaxy and subsequently accrete onto its inner regions, in a non-dissipative fashion. In the other case, first the gas flows into the central regions and then it is turned into stars. If the correct scenario is the first one, smoothing of the central density cusp of the profiles is expected, because the infalling stellar clumps loose orbital energy through dynamical friction and the halo heats up (Bertin et al. 2003). In the second case, the central distribution is expected to steepen up, because the gas radiates away its internal energy and an adiabatic contraction takes place (Gnedin et al. 2004). Obviously, both processes are expected to play a role in reality, and this is indeed what we see in our models, in which however the final density profiles show a plateau in the central regions, as expected in the “infalling stars” model.

Our results also clearly suggest that if stars are formed too early, the resulting stellar systems are too shallow, ending up in “wrong” profiles: stars will tend to conserve their velocity dispersion as they fall into the potential wells and will therefore have elongated orbits, which is not the case when gas is let dissipate part of its kinetic energy and reach very high densities before turning into stars.

In this context, it is worth recalling that there is some observational evidence for a deficit in the central luminosity of a few early type galaxies as recently found in observational data by Côté et al. (see 2007, their Fig. 1, and references therein). However, the observed flattening begins at smaller radii (a few percent of r_{eff}), and it is present only in bright (i.e. massive) systems, whereas the small systems seem to present a luminosity excess. Indeed, looking at the panels in the bottom row of Fig. 26, one can notice that the central profiles of low mass models show some scatter and tend to steepen in the very central regions (especially in the VLDLM model). Recalling that the observational information stands on the luminosity profile, given the presence of young and hence luminous stellar populations in the cores of our low mass galaxy models, (and their absence in the cores of the massive ones), we can speculate that the trends of the observed curves should be fairly reproduced by translating the mass profiles into luminosity into luminosity profiles. This is an interesting point, because the excess or deficit of light could be explained just in terms of the stars and DM orbits, without the need for a super-massive black hole affecting the global dynamics in the center of the systems, as instead suggested in the cited works.

5. The Fundamental Plane

Among the many possible applications of these models to the interpretation of the fundamental properties of early type galaxies, such as the morphology, the chemical and photometric data, the chemical gradients, the scale relations in their many versions (e.g. Kormendy relation, Fundamental Plane, etc.) we briefly focus here on the Fundamental Plane (FP) relation, as it provides the most

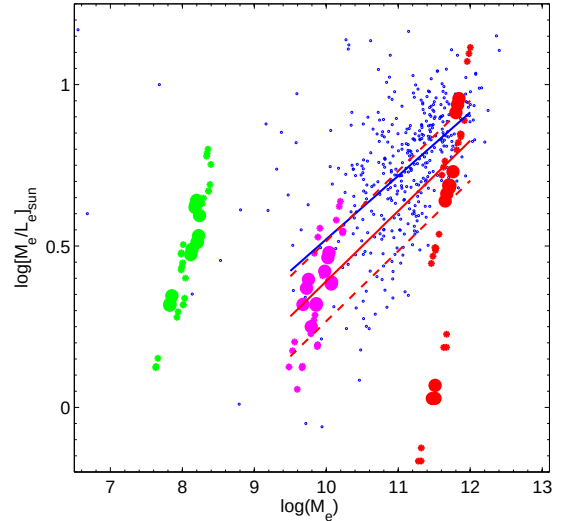


Fig. 28. Fundamental plane of EGs in the B pass-band, i.e. the relation $\log(M_e/L_e)_\odot = \log(M/L_B)_\odot$ versus $\log M_e$, where $\log M_e$ is the dynamical mass enclosed within the half light radius, in units of $10^{12} M_\odot$ calculated according to Burstein et al. (1997) (see Appendix A), and L_e is the blue luminosity enclosed within the half light radius. The small circles are the data from Burstein et al. (1997), whose linear fit is shown as a blue continuous line. The big filled circles refer to our sets of models (12 models $\times$ 3 projection planes = 36 models). Small filled circles: our models taking into account the effect of $\pm 20\%$ uncertainty in the velocity dispersion whose linear fit is represented with a dashed line.

tight constraints to the underlying formation mechanism of EGs (Dressler et al. 1987; Djorgovski & Davis 1987).

In Fig. 28 we show the M/L vs. M projection of the FP, as obtained by Burstein et al. (1997), in the B pass-band, compared with the position of our models in this same diagnostic plane at the time t_{last} given in Tab. 4. It is worth recalling that since some models have been calculated up to some final redshift $z > 0$, the time t_{last} does not correspond to the present age of the Universe t_{Uni} nor the real age of the galaxy $t_G = t_{Uni} - t_{z,ini}$, but in those cases $t_{last} < t_G$. The B luminosities L_B inside R_e have been calculated as follows. In each model of our set, the star particles are centered on the barycenter and projected on the XY-plane, XZ-plane and YZ-plane. In this way we can take into account the deviation of the object from the spherical symmetry by means of the three different projections corresponding to three different positions of an external observer. Each star particle represents a SSP of metallicity Z , born at the time t and with age $\tau = t_{last} - t$. The total luminosity in the B-band and the half-light radius R_e in kpc are then obtained subdividing the 2D projections in a fine grid of circular rings and adapting the population synthesis technique for N-body simulations presented in Tantalo et al. (2010). Since we tried to be as consistent as possible with the Burstein et al.

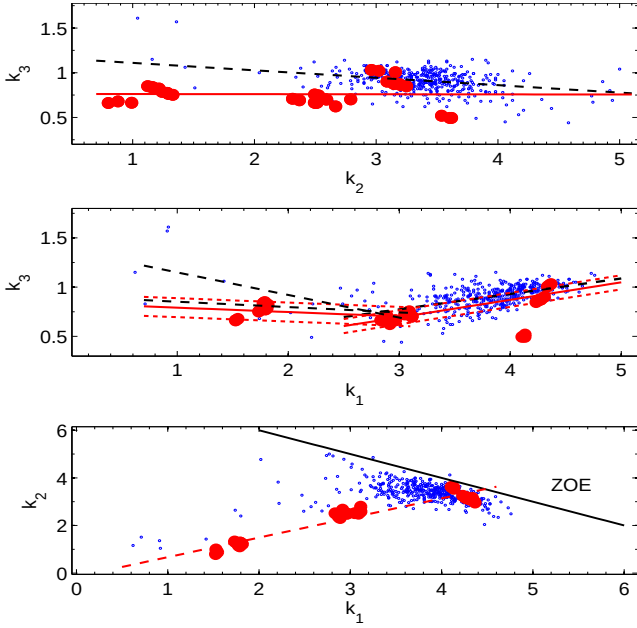


Fig. 29. Three projections of the kappa-space. In all the panels the small circles are the elliptical galaxies of Burstin et al. (1997), from dwarfs to giants, while the big circles represent our 36 models (12 models $\times$ 3 projection planes). The **bottom panel** represents the effect of I_e , that varies of a big factor over the mass interval. The ZOE is also represented. The **middle panel** shows the FP seen edge on. Dashed lines represent a fit to the observed data for $M > 10^{10} M_\odot$ and $M < 10^{10} M_\odot$, while the fit to our models is represented as a continuous line. Dashed lines take into account the uncertainty on σ_c . In the **upper panel** we show a fit to the whole set of data and to our models that agree only marginally in the k_2 vs. k_3 space.

(1997) estimates of masses and luminosities, the masses M_{eff} inside the effective radius R_{eff} are derived from:

$$M_e = 4.65 \times 10^5 \sigma_c^2 R_e M_\odot \quad (8)$$

where σ_c is the velocity dispersion in km s^{-1} . This is obtained by averaging over the particles within the central region inside $R_e/8$. As a consequence of the gravitational softening, the estimates of the velocity dispersion in the innermost region of the models can be affected by a significant degree of uncertainty. This is because the velocities of the particles are not consistent with the real Newtonian accelerations. In the plane $\log(M_e/L_e)_\odot$ vs $\log(M_e)$ of Fig. 28, the slope of the linear fit of the Burstin et al. (1997) data [$\log(M_e/L_e)_\odot = 0.2177 \log(M_e) - 1.786$] and to our models ($\log(M_e/L_e)_\odot = 0.1966 \log(M_e) - 1.445$) are nearly parallel, but for a slight off-set along the Y-axis. Trying to mimic the observational data, we introduced a 20% uncertainty on the theoretical estimate of the velocity dispersion σ_c : this is enough to make the fits almost coincident. Both fits have been obtained for the models with masses $M_e > 10^{10} M_\odot$ therefore excluding the model VLDHM because this galaxy is still forming stars at a significant rate. Indeed, its B-band mass-to-light ratio is much lower than for all other models. In any case,

the observed and simulated mass-to-light ratios in the B band, M/L_B , show a consistent behaviour, increasing with galaxy mass, which might actually explain a large fraction of the tilt of the FP (e.g. Bolton et al. 2008).

In any case, the observed and simulated mass-to-light ratios in the B band M/L_B agree and increase with the galaxy mass (*tilt* of the FP; see e.g. Ciotti et al. 1996); this can be explained considering the mean age of the dominant stellar population, implying that decreasing the stellar masses requires significant populations of young stars. Our models do account for such a requirement and at the final simulated redshift they lie in the region of the diagram populated by the early-type galaxies.

The low mass models invert the trend and have high values of the M/L ratios at t_{last} compared with the intermediate mass models. Considering that in this mass range the scatter in the observational data is very large, the location of the models on this diagnostic plane may be taken as reasonable. Dwarf galaxies are known to have large mass-to-light ratios (Mateo 1998), so these results are consistent with the observations.

The kappa-space. It is worth of interest to compare our models with the observational data directly in the kappa-space defined by Bender et al. (1992) as:

$$k_1 = \frac{1}{\sqrt{2}} (2 \log \sigma_c + \log R_e) \quad (9)$$

$$k_2 = \frac{1}{\sqrt{6}} (2 \log \sigma_c + 2 \log I_e - \log R_e) \quad (10)$$

$$k_3 = \frac{1}{\sqrt{3}} (2 \log \sigma_c - \log I_e - \log R_{eff}) \quad (11)$$

where I_e is in $L_\odot \text{ pc}^{-2}$ and is defined by the relation $L_e = \pi \times 10^6 I_e R_e$ in the B-band. The axes of the κ -space are related to galaxy mass, mass-to-light ratio and a third quantity mainly depending on the surface brightness. The results are shown in Fig. 29. The data are the same as in Fig. 28.

The three panels of Fig. 29 show the correlation between pairs of the k_1 , k_2 and k_3 parameters. The mid panel k_1 vs. k_3 shows the FP seen edge on. The data are grouped in two regions according to the mass, i.e. $M \geq 10^{10} M_\odot$ and $M < 10^{10} M_\odot$. The linear best fit of our models limited to masses greater than $10^{10} M_\odot$ yields $k_3 = 0.1777 k_1 + 0.1604$ (continuous line) to be compared with the fit of the Burstin et al. (1997) data ($k_3 = 0.1567 k_1 + 0.3044$; dashed line). We excluded again the VLDHM model for the reasons cited above, so that the agreement between simulations and observations is good. If also in this case we include a 20% uncertainty in the dispersion velocity (dotted lines) we easily recover the vertical offset of the models compared to the observational data. Most likely, the effect of the softening of the gravitational force is to underestimate the velocity dispersion. If we cut the three most extreme objects, there is also good agreement between models and observations even for the low-intermediate mass range where again

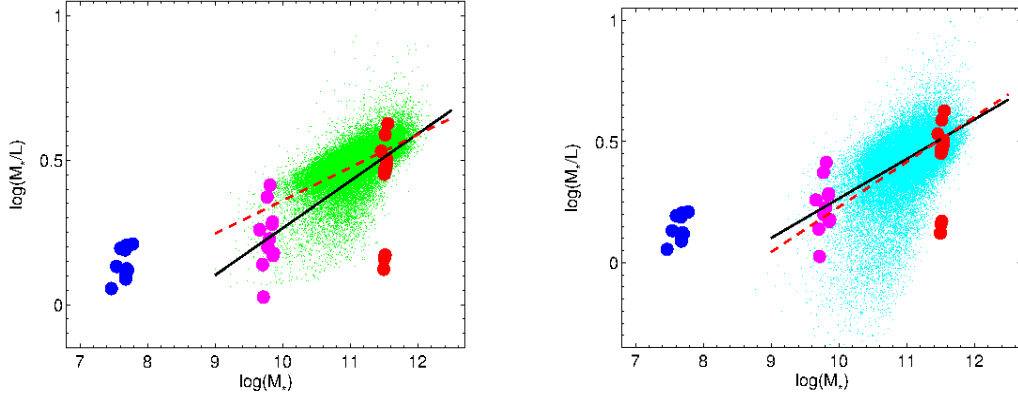


Fig. 30. M/L vs. M projection of the FP in the SDSS r' pass-band, i.e. the relation $\log(M_e/L_e)_\odot = \log(M/L_{r'})_\odot$ versus $\log M_e$, where $\log M_e$ is the stellar mass enclosed within the half-light radius, in units of $10^{12} M_\odot$ and calculated according to Bell et al. (2003) (left panel) and Gallazzi et al. (2005) (right panel). $L_{r'}$ is the r' luminosity enclosed within the half light radius. The small dots in both panels are the data from the catalogue by Bernardi et al. (2010) (private communication), whose linear fit is shown as a continuous line. The big filled circles refer to our sets of models (12 models $\times$ 3 projection planes = 36 models) and the best fit of the intermediate and high mass models is shown as a dashed line.

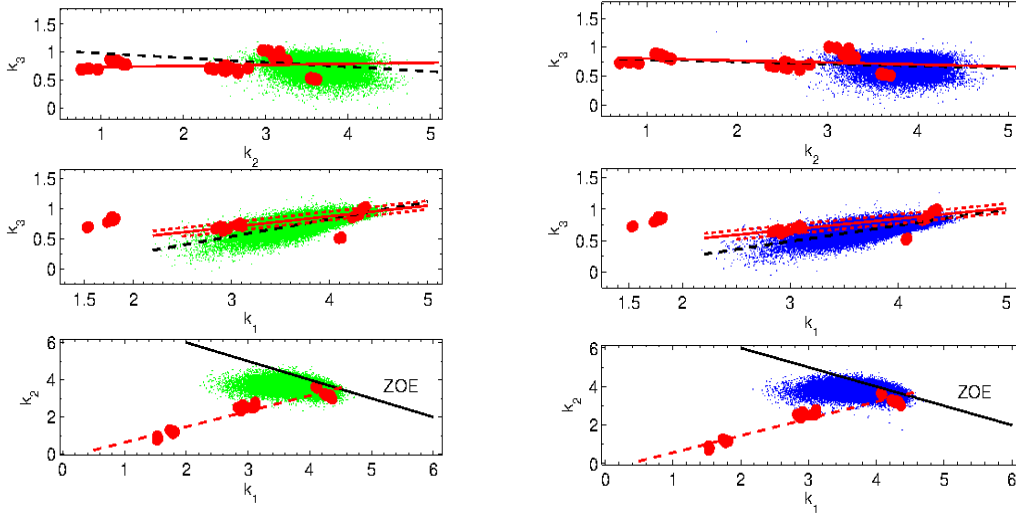


Fig. 31. Fundamental plane in the kappa-space for the the SDSS g' (left) and r' pass-bands. In all the panels the small dots are the elliptical galaxies of Bernardi et al. (2010) (private communication). The meaning of the symbols is the same as in Fig. 29.

we can apply the same considerations as above to the estimates of the mass and velocity dispersion. The global fit of the low-intermediate mass-galaxies with all the objects included, yields a steeper fit (dashed line) that agree only in the trend of the slope with the models. In any case, the sample is too small to draw any conclusions. In the bottom panel we display the correlation between k_2 and k_1 . The dashed line is the linear fit to all the models. As already noted by Chiosi & Carraro (2002), most of the galaxies follow the slope of (and remain below) the line $k_1 + k_2 \leq 8$, while the sample of dwarf galaxies are distributed in a long tail with a slope similar to that of the theoretical models. In particular, the

boundary line $k_1 + k_2 \leq 8$ simply reflects that in the course of the Universe evolution the mean initial density and upper mass limit of proto-galaxies get lower and higher, respectively.

In Fig. 30, we compare our models to the mass-to-light ratios obtained with SDSS observations of EGs (Bernardi et al. 2010). In particular we focus on the r' band, for which we adopt the magnitudes and effective radii derived from a combination of De Vaucouleurs and exponential profiles. In order to estimate the mass-to-light ratio, two choices are available for the mass: (1) the estimate of Bell et al. (2003); (2) the estimate of Gallazzi et al. (2005)

using a combination of medium-high resolution spectra and different SFH. For all the details see Bernardi et al. (2010), in particular Appendix A and Sect. 2. We find a reasonable agreement with the Bell et al. (2003) mass determinations: the theoretical models yield $\log(M/l) = 0.1632 \log(M) - 1.367$ whereas the data yield $\log(M/L) = 0.1143 \log(M) - 0.7828$. Better agreement is achieved using the Gallazzi et al. (2005) masses, in the case the fit of the observational data yields $\log(M/L) = 0.1859 \log(M) - 1.629$.

Finally, in Fig. 31 we show the FP for EGs in the g' and r' pass-bands. The agreement with the observations is good, once again when a $\pm 20\%$ correction in the dispersion velocity is applied. Compared to Fig. 29, now there is a small difference in the slope between the fits to our models and observations in the k_1 vs. k_3 plane. In the k_2 vs. k_3 plane the agreement is now better than for the B band, in particular for r' pass-band. It must be pointed out that we are looking for a general agreement between models and observations to validate our N-body simulations and our photometric calculations. The sample of (Bernardi et al. 2010) contains almost sixty thousands galaxies with variable redshift, going from $z=0$ to about $z=0.27$, while our models have a final redshift of the simulation going from $z=0$ to $z=1$. A more detailed analysis would require more simulations, in particular around $\log(M_*) \sim 11$, and to complete all the simulations up to $z=0$.

5.1. The Mass-Radius Relation

To conclude our analysis, we briefly consider the Mass-Radius Relation (MRR). In what follows, we limit ourselves to anticipate the results of a companion study by Chiosi et al. (2012, in preparation) on this subject. Fig. 32 shows the observational MRR by (Bernardi et al. 2010; Burstein et al. 1997) and the position of our models. It also displays the linear fit of the data by van Dokkum et al. (2010) for high-redshift galaxies. At a first glance, one would say that high and intermediate mass models fairly agree with the observations, whereas the low mass ones apparently have too large radii with respect to their masses, and therefore the models in general fail in reproducing the observational MRR. Nevertheless, reality may be more complicated than this. We begin by noting that models with equal initial over-densities fall very close to straight lines $R_{1/2} \propto M^\alpha$, with slope α ranging from 0.2 to 0.3 (in Fig. 32 the dashed black lines have $\alpha = 0.27$ and $\alpha = 0.19$). These slopes are similar to those found by Chiosi & Carraro (2002) for their models (see their Eq. 40), and for the observed dwarf galaxies (see their Eqn. 38 and their Fig. 15). Moreover, they resemble the slope of the theoretical MRR of galactic systems assembled within spherical DM perturbations, $R_{1/2} \propto M^{1/3}$ (see e.g. Fan et al. 2010). The deviation from the pure $R_{1/2} \propto M^{1/3}$ relationship is larger towards galaxies of lower mass and it is due to the complex baryon physics; the discussion by Chiosi & Carraro (2002) on the issue still applies to the

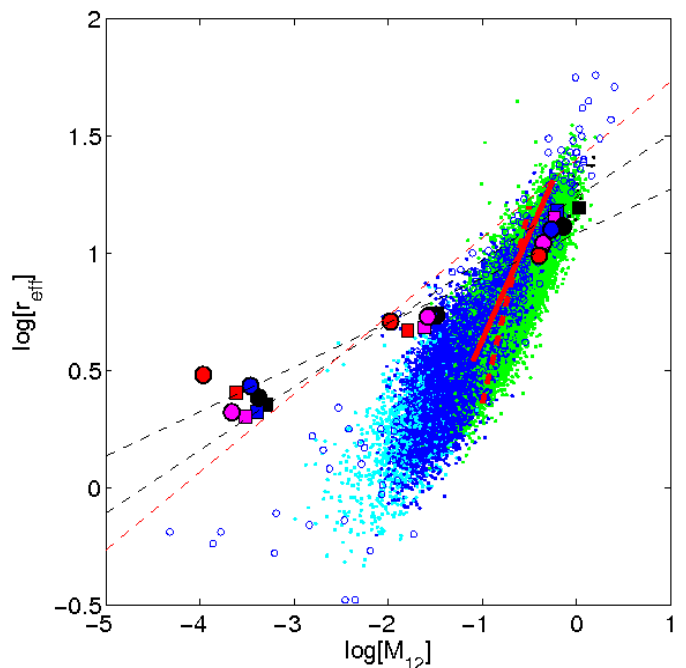


Fig. 32. Mass-radius relation, i.e. the relation $\log(r_{1/2})$ versus $\log M_{12}$, where $r_{1/2}$ is the half light radius, and $\log M_{12}$ is the total (dynamical) mass enclosed within the half light radius, in units of $10^{12} M_\odot$. Small points are data from Bernardi et al. (2010), coloured by redshift (cyan: $z < 0.05$; blue: $0.05 \leq z < 0.1$; green: $0.1 \leq z < 0.3$; black: $z \geq 0.3$). The small open circles are the data from Burstein et al. (1997). Different large symbols refer to different sets of models with the same initial density; while the same colors refer to same initial densities. Large circles: the twelve major models at $z = 1$. Large squares: the models at the last simulated age. Black: high density. Blue: intermediate density. Magenta: low density. Red: very low density. The red solid line is the best fit from Guo et al. (2009). The thick red dashed line is the evolutionary path found by van Dokkum et al. (2010). The thin dashed black lines are fits to the models (see text). The thin dashed red line is the $r \propto m^{1/3}$ slope, plotted for reference.

present models and it is strengthened by the more physically sounded cosmological ICs adopted in the present study⁵.

However, this is not the whole story. Starting from a similar analysis of the MRR made Chiosi & Carraro (2002) we argue that there is another effect to consider which turns out to ultimately drive the observational MRR. As already said, all details of this discussion are left to the companion paper by Chiosi et al. (2012, in preparation); here we only outline the main ideas and results. In brief,

⁵ In brief, Chiosi & Carraro (2002) argued that the different slope of the MRR traced by the models and the one indicated by the high-mass galaxies cannot be ascribed to mergers, because in this case the models would fall even farther from the loci populated by real galaxies. They argue that the difference is due to the effect of *galactic winds*. See Sect. 3.4 for more details on the same issue.

we argue and demonstrate that the observational MRR is the result of two competing mechanisms:

- First, proto-galaxies made of DM and BM undergoing collapse give rise to virialized systems whose mass and radius obey the general relation $R \propto M^{1/3}$, where the proportionality factor is simply the initial density excess with respect to surrounding medium of the perturbation generating the galaxy in question. The density excess depends on the formation redshift and ultimately fixes the zero point of the MRR associated to that particular galaxy. Systems which virialize first have larger average densities, since the Universe was denser when they collapsed. This states that, as expected from simple analytical collapse models, the density of an object is defined by the background density at the moment of its virialization. Given the mass of the proto-galaxy, the initial radius follows immediately. This is true for the DM component and to large extent the BM component as well. The different slopes displayed by the models (the deviation being larger at lower galaxy masses so that the final MRR is close to that traced by galaxies passing from high to low masses objects), are the consequence of the “disturbing” action of baryon physics.
- Second, the cosmological background fixes the typical mass of halos collapsing at any redshift, the *halo growth function* (see e.g. Lukić et al. 2007). The intersections between the lines of the theoretical MRR for systems collapsing at a given redshift and the corresponding typical halo mass at that redshift (assuming a given overall efficiency of star formation measured by the ratio $m = M_{\text{halo}}/M_{\text{stars}}$) form a region which fits astonishingly well the region populated by real galaxies on the MR plane, and whose mean slope nicely agrees with that of the data. Therefore, the observational MRR is the locus of galaxies born at different redshift whose mass is close to the most probable value for objects collapsing at each redshift. In this context, the dwarf galaxies (whose halos became common at early epochs, and whose masses are far from the high probability boundary) can seat on their own $R \propto M^{1/3}$ relation apparently deviating from the the observational MRR.

In this picture, some of the models may even fall outside the region populated by real objects simply because they are built *arbitrarily* assigning them initial conditions (namely, redshift, density contrast, masses and radii) that were indeed more appropriated to other epochs. Typically, this can be the case of some of the low mass models, which are supposed to collapse in a rather recent epochs with respect to the real epoch at which galaxies of this mass are expected to collapse and virialize more commonly. These models “correctly” set on their collapse line in the MR plane, but the probability that such systems come to existence in the real Universe is so low that no real galaxy is observed there. Interestingly, our massive models *do fall*

within the observed region. This means that such systems have reasonable formation histories.

6. Summary and conclusions

We have presented a numerical investigation on the star formation histories of elliptical galaxies, performed with the aid of hydrodynamical NB-TSPH simulations of galaxy formation in cosmological context. By varying the initial properties of a single model in terms of total mass and initial over-density, we investigated how the initial configuration influences the formation of the system along the Hubble time.

The main result of our study is that the star formation history of a galaxy is directly dependent on its initial total mass and initial overdensity. Massive galaxies ($M_{\text{tot}} \simeq 10^{13} M_{\odot}$) build their stellar content passing through an initial burst of activity at early times. In contrast, low mass systems ($M_{\text{tot}} \simeq 10^9 M_{\odot}$) continue to form stars throughout their lifetime, with episodes of intense activity separated by period of relative quiescence. Moreover, galaxies with the same initial mass have different star formation histories depending on their initial over-density: the deeper the perturbation, the more peaked, early and intense is the star forming activity. This is particularly true for intermediate mass galaxies ($M_{\text{tot}} \simeq 10^{11} M_{\odot}$). Thus, denser objects, which may be identified with cluster and group central galaxies, have more peaked and earlier burst of star formation, while less dense objects (i.e., field and satellite galaxies) usually have delayed and lower activity. This is largely consistent with previous analyses by Chiosi & Carraro (2002), Clemens et al. (2006), and Thomas et al. (2005) to mention a few.

The structural properties of our models are in good agreement with those of the observed real galaxies, with some important differences (e.g. the density profiles in the central regions, the mass-to-metallicity relation) which we ascribe to the unusual choice we have made for the star formation dimensionless efficiency parameter, $\epsilon_{SF} = 1$.

The picture emerging from this study is that an early-hierarchical, monolithic-like intense activity of star formation at high redshift is possible starting from physically sounded cosmological conditions, and is fully adequate to explain the observed features of massive elliptical galaxies.

Major mergers are certainly possible, but they are not strictly required. It is indeed easy to understand that on the contrary they would alter the subtle game among different physical ingredients all contributing to shape the body of properties and their mutual tight correlations shown by early type galaxies such as for instance the MRR itself (Chiosi & Carraro 2002; Chiosi et al. 2012). Along this line of thought Buonomo et al. (2000) showed how a major merger would shift the position of the parent objects towards the upper region of the MR plane, where no observed galaxy is found. Moreover, chemical features of the elliptical galaxies cannot be reproduced if (wet) merging events are considered. Finally, the shape of the galaxy mass function (Bundy et al. 2006; Pérez-González

et al. 2008) rules out the possibility that late major dry mergers are responsible for the mass assembly of massive systems, out of smaller but already passively evolving objects. Our models show instead that massive galaxies can form out of single, peaked perturbations, consistently with the cosmological background.

Another interesting point to note is that the stellar feedback seems to be sufficient to quench the star formation process in massive objects *without* invoking more exotic sources of energy, like Active Nuclei, provided it is modeled with sufficient accuracy.

Furthermore, we are also able to recover the complex, episodic star formation histories typical of many dwarf galaxies. The galactic breathing phenomenon is a natural consequence of an accurate treatment of the stellar feedback process.

We conclude that the fate of a proto-galaxy is essentially determined by its initial conditions in terms of mass and over-density, while external factors such as encounters, mergers and disruptions, while substantially altering the evolution of the involved systems, are not a fundamental ingredient in the global evolution of early type galaxies population.

Acknowledgements. We express our gratitude to Dr. M. Bernardi for providing her database of SDSS observations of early type galaxies and many important clarifications. This study has been financed by the University of Padua under the Strategic Research Project "Algorithms and Architectures for Computational Science and Engineering".

References

- Barnes, J. & Hut, P. 1986, *Nat*, 324, 446
- Bell, E. F., McIntosh, D. H., Katz, N., & Weinberg, M. D. 2003, *ApJS*, 149, 289
- Bell, E. F., Wolf, C., Meisenheimer, K., Rix, H.-W., Borch, A., Dye, S., Kleinheinrich, M., Wisotzki, L., & McIntosh, D. H. 2004, *ApJ*, 608, 752
- Bender, R., Burstein, D., & Faber, S. M. 1992, *ApJ*, 399, 462
- Bernardi, M., Shankar, F., Hyde, J. B., Mei, S., Marulli, F., & Sheth, R. K. 2010, *MNRAS*, 404, 2087
- Bertin, G., Liseikina, T., & Pegoraro, 2003, *A&A*, 405, 73
- Bertschinger, E. 1995, *ArXiv Astrophysics e-prints*, astro-ph/9506070
- Birnboim, Y. & Dekel, A. 2003, *MNRAS*, 345, 349
- Bolton, A. S., Burles, S., Koopmans, L. V. E., Treu, T., Gavazzi, R., Moustakas, L. A., Wayth, R., & Schlegel, D. J. 2008, *ApJ*, 682, 964
- Bromm, V., Coppi, P. S., & Larson, R. B. 2002, *ApJ*, 564, 23
- Bundy, K., Ellis, R. S., Conselice, C. J., Taylor, J. E., Cooper, M. C., Willmer, C. N. A., Weiner, B. J., Coil, A. L., Noeske, K. G., & Eisenhardt, P. R. M. 2006, *ApJ*, 651, 120
- Buonomo, F., Carraro, G., Chiosi, C., & Lia, C. 2000, *MNRAS*, 312, 371
- Burstein, D., Bender, R., Faber, S., & Nolthenius, R. 1997, *AJ*, 114, 1365
- Caon, N., Capaccioli, M., & D’Onofrio, M. 1993, *MNRAS*, 265, 1013
- Carraro, G., Lia, C., & Chiosi, C. 1998, *MNRAS*, 297, 1021
- Ceverino, D. & Klypin, A. 2009, *ApJ*, 695, 292
- Chiosi, C., Bressan, A., Portinari, L., & Tantalo, R. 1998, *A&A*, 339, 355
- Chiosi, C. & Carraro, G. 2002, *MNRAS*, 335, 335
- Chiosi, C., Merlin, E., & Piovan, L. 2012, in prep.
- Cho, H. & Kang, H. 2008, *New A*, 13, 163
- Churches, D. K., Nelson, A. H., & Edmunds, M. G. 2001, *MNRAS*, 327, 610
- Ciotti, L., Lanzoni, B., & Renzini, A. 1996, *MNRAS*, 282, 1
- Clemens, M. S., Bressan, A., Nikolic, B., Alexander, P., Annibali, F., & Rampazzo, R. 2006, *MNRAS*, 370, 702
- Côté, P., Ferrarese, L., Jordán, A., Blakeslee, J. P., Chen, C., Infante, L., Merritt, D., Mei, S., Peng, E. W., Tonry, J. L., West, A. A., & West, M. J. 2007, *ApJ*, 671, 1456
- Cox, T. J., Jonsson, P., Primack, J. R., & Somerville, R. S. 2006, *MNRAS*, 373, 1013
- Croft, R. A. C., Di Matteo, T., Springel, V., & Hernquist, L. 2009, *MNRAS*, 400, 43
- De Lucia, G. & Blaizot, J. 2007, *MNRAS*, 375, 2
- Djorgovski, S. & Davis, M. 1987, *ApJ*, 313, 59
- Dressler, A., Lynden-Bell, D., Burstein, D., Davies, R. L., Faber, S. M., Terlevich, R., & Wegner, G. 1987, *ApJ*, 313, 42
- Dyson, J. E. & Williams, D. A. 1997, *The physics of the interstellar medium (The physics of the interstellar medium. Edition: 2nd ed. Publisher: Bristol: Institute of Physics Publishing, 1997. Edited by J. E. Dyson and D. A. Williams. Series: The graduate series in astronomy. ISBN: 0750303069)*
- Eggen, O. J., Lynden-Bell, D., & Sandage, A. R. 1962, *ApJ*, 136, 748
- Elmegreen, D. M., Elmegreen, B. G., Ferguson, T., & Mullan, B. 2007, *ApJ*, 663, 734
- Fan, L., Lapi, A., Bressan, A., Bernardi, M., De Zotti, G., & Danese, L. 2010, *ApJ*, 718, 1460
- Ferrarese, L., Côté, P., Jordán, A., Peng, E. W., Blakeslee, J. P., Piatek, S., Mei, S., Merritt, D., Milosavljević, M., Tonry, J. L., & West, M. J. 2006, *ApJS*, 164, 334
- Gallazzi, A., Charlot, S., Brinchmann, J., White, S. D. M., & Tremonti, C. A. 2005, *MNRAS*, 362, 41
- Gao, L., Yoshida, N., Abel, T., Frenk, C. S., Jenkins, A., & Springel, V. 2007a, *MNRAS*, 378, 449
- Gao, Y., Carilli, C. L., Solomon, P. M., & Vanden Bout, P. A. 2007b, *ApJ*, 660, L93
- Gnedin, O. Y., Kravtsov, A. V., Klypin, A. A., & Nagai, D. 2004, *ApJ*, 616, 16
- Gobat, R., Daddi, E., Onodera, M., Finoguenov, A., Renzini, A., Arimoto, N., Bouwens, R., Brusa, M., Chary, R., Cimatti, A., Dickinson, M., Kong, X., & Mignoli, M. 2011, *A&A*, 526, A133+
- Greggio, L. & Renzini, A. 1983, *A&A*, 118, 217

- Guo, Y., McIntosh, D. H., Mo, H. J., Katz, N., van den Bosch, F. C., Weinberg, M., Weinmann, S. M., Pasquali, A., & Yang, X. 2009, *MNRAS*, 398, 1129
- Harrison, C. D., Colless, M., Kuntschner, H., Couch, W. J., de Propris, R., & Pracy, M. B. 2011a, *MNRAS*, 159
- . 2011b, *MNRAS*, 413, 1036
- Hernquist, L. 1990, *ApJ*, 356, 359
- Hinshaw, G., Weiland, J. L., Hill, R. S., Odegard, N., Larson, D., Bennett, C. L., Dunkley, J., Gold, B., Greason, M. R., Jarosik, N., Komatsu, E., Nolte, M. R., Page, L., Spergel, D. N., Wollack, E., Halpern, M., Kogut, A., Limon, M., Meyer, S. S., Tucker, G. S., & Wright, E. L. 2008, *ArXiv e-prints*
- Hollenbach, D. 1988, *Astrophysical Letters Communications*, 26, 191
- Hollenbach, D. & McKee, C. F. 1979, *ApJS*, 41, 555
- Ikeuchi, S. & Ostriker, J. P. 1986, *ApJ*, 301, 522
- Katz, N. 1992, *ApJ*, 391, 502
- Kobayashi, C. 2004, *MNRAS*, 347, 740
- . 2005, *MNRAS*, 361, 1216
- Kroupa, P. 2001, *MNRAS*, 322, 231
- Krumholz, M. R. & Tan, J. C. 2007, *ApJ*, 654, 304
- La Barbera, F. & de Carvalho, R. R. 2009, *ApJ*, 699, L76
- La Barbera, F., De Carvalho, R. R., De La Rosa, I. G., Gal, R. R., Swindle, R., & Lopes, P. A. A. 2010, *AJ*, 140, 1528
- La Barbera, F., Ferreras, I., de Carvalho, R. R., Lopes, P. A. A., Pasquali, A., de la Rosa, I. G., & De Lucia, G. 2011, *ApJ*, 740, L41
- Lada, C. J. & Lada, E. A. 2003, *ARA&A*, 41, 57
- Lia, C., Portinari, L., & Carraro, G. 2002, *MNRAS*, 330, 821
- Lukić, Z., Heitmann, K., Habib, S., Bashinsky, S., & Ricker, P. M. 2007, *ApJ*, 671, 1160
- Marchesini, D., Whitaker, K. E., Brammer, G., van Dokkum, P. G., Labbé, I., Muzzin, A., Quadri, R. F., Kriek, M., Lee, K., Rudnick, G., Franx, M., Illingworth, G. D., & Wake, D. 2010, *ApJ*, 725, 1277
- Mateo, M. L. 1998, *ARA&A*, 36, 435
- McKee, C. F. & Ostriker, J. P. 1977, *ApJ*, 218, 148
- Merlin, E., Buonomo, U., Grassi, T., Piovan, L., & Chiosi, C. 2010, *A&A*, 513, A36+
- Merlin, E. & Chiosi, C. 2006, *A&A*, 457, 437
- . 2007, *A&A*, 473, 733
- Merlin, E., Grassi, T., Buonomo, U., Piovan, L., & Chiosi, C. 2012, in prep.
- Monaco, P., Murante, G., Borgani, S., & Fontanot, F. 2006, *ApJ*, 652, L89
- Mortlock, D. J., Warren, S. J., Venemans, B. P., Patel, M., Hewett, P. C., McMahon, R. G., Simpson, C., Theuns, T., González-Solares, E. A., Adamson, A., Dye, S., Hambly, N. C., Hirst, P., Irwin, M. J., Kuiper, E., Lawrence, A., & Röttgering, H. J. A. 2011, *Nature*, 474, 616
- Naab, T., Johansson, P. H., Ostriker, J. P., & Efstathiou, G. 2007, *ApJ*, 658, 710
- Navarro, J. F., Frenk, C. S., & White, S. D. M. 1996, *ApJ*, 462, 563
- Nelson, A. F. 2006, *MNRAS*, 373, 1039
- Padmanabhan, N., Seljak, U., Strauss, M. A., Blanton, M. R., Kauffmann, G., Schlegel, D. J., Tremonti, C., Bahcall, N. A., Bernardi, M., Brinkmann, J., Fukugita, M., & Ivezić, Ž. 2004, *New A*, 9, 329
- Pawlik, A. H. & Schaye, J. 2009, *MNRAS*, 396, L46
- Peletier, R. F., Valentijn, E. A., & Jameson, R. F. 1990, *A&A*, 233, 62
- Pelupessy, F. I. 2005, PhD thesis, Leiden Observatory, Leiden University, P.O. Box 9513, 2300 RA Leiden, The Netherlands
- Pérez-González, P. G., Rieke, G. H., Villar, V., Barro, G., Blaylock, M., Egami, E., Gallego, J., Gil de Paz, A., Pascual, S., Zamorano, J., & Donley, J. L. 2008, *ApJ*, 675, 234
- Press, W. H. & Schechter, P. 1974, *ApJ*, 187, 425
- Rees, M. J. & Ostriker, J. P. 1977, *MNRAS*, 179, 541
- Robertson, B. E. & Kravtsov, A. V. 2008, *ApJ*, 680, 1083
- Sales, L. V., Navarro, J. F., Schaye, J., Vecchia, C. D., Springel, V., & Booth, C. M. 2010, *MNRAS*, 409, 1541
- Scodeggio, M. 2001, *AJ*, 121, 2413
- Sersic, J. L. 1968, *Atlas de galaxias australes (Cordoba, Argentina: Observatorio Astronomico, 1968)*
- Sheth, R. K., Mo, H. J., & Tormen, G. 2001, *MNRAS*, 323, 1
- Sommer-Larsen, J. & Toft, S. 2010, *ApJ*, 721, 1755
- Spolaor, M., Proctor, R. N., Forbes, D. A., & Couch, W. J. 2009, *ApJ*, 691, L138
- Stinson, G. S., Bailin, J., Couchman, H., Wadsley, J., Shen, S., Nickerson, S., Brook, C., & Quinn, T. 2010, *MNRAS*, 408, 812
- Suh, H., Jeong, H., Oh, K., Yi, S. K., Ferreras, I., & Schawinski, K. 2010, *ApJS*, 187, 374
- Sutherland, R. S. & Dopita, M. A. 1993, *ApJS*, 88, 253
- Tantalo, R., Chinellato, S., Merlin, E., Piovan, L., & Chiosi, C. 2010, *A&A*, 518, A43+
- Tegmark, M., Silk, J., Rees, M. J., Blanchard, A., Abel, T., & Palla, F. 1997, *ApJ*, 474, 1
- Thomas, D. 2011, *ArXiv e-prints*
- Thomas, D., Maraston, C., Bender, R., & Mendes de Oliveira, C. 2005, *ApJ*, 621, 673
- Thornton, K., Gaudlitz, M., Janka, H.-T., & Steinmetz, M. 1998, *ApJ*, 500, 95
- Tortora, C., Napolitano, N. R., Cardone, V. F., Capaccioli, M., Jetzer, P., & Molinaro, R. 2010, *MNRAS*, 407, 144
- Tortora, C., Napolitano, N. R., Romanowsky, A. J., Capaccioli, M., & Covone, G. 2009, *MNRAS*, 396, 1132
- Tremonti, C. A., Heckman, T. M., Kauffmann, G., Brinchmann, J., Charlot, S., White, S. D. M., Seibert, M., Peng, E. W., Schlegel, D. J., Uomoto, A., Fukugita, M., & Brinkmann, J. 2004, *ApJ*, 613, 898
- van Dokkum, P. G., Whitaker, K. E., Brammer, G., Franx, M., Kriek, M., Labbé, I., Marchesini, D., Quadri, R., Bezanson, R., Illingworth, G. D., Muzzin, A., Rudnick, G., Tal, T., & Wake, D. 2010, *ApJ*, 709, 1018

- Warren, S., Lawrence, A., Almaini, O., Cirasuolo, M., Foucaud, S., Hambly, N., Hewett, P., Jameson, R., Leggett, S., Lodieu, N., Lucas, P., McLure, R., McMahon, R., Mortlock, D., Pinfield, D., & Venemans, B. 2006, *The Messenger*, 126, 7
- White, S. D. M. 1984, *ApJ*, 286, 38
- White, S. D. M. & Rees, M. J. 1978, *MNRAS*, 183, 341
- Yoshida, N., Abel, T., Hernquist, L., & Sugiyama, N. 2003, *ApJ*, 592, 645