PARALLEL TREE-SPH: A TOOL FOR GALAXY FORMATION

C. LIA
SISSA/ISAS, via Beirut 2, I-34013 Trieste, Italy; E-mail: liac@sissa.it

G. CARRARO
Department of Astronomy, Padova Univesity, Vicolo Osservatorio 5, I-35122, Padova, Italy
E-mail: carraro@pd.astro.it

Abstract. We describe a new implementation of a parallel Tree-SPH code with the aim of simulating galaxy formation and evolution. The code has been parallelized using SHMEM, a Cray proprietary library to handle communications between the 256 processors of the Silicon Graphics T3E massively parallel supercomputer hosted by the Cineca Super-computing Center (Bologna, Italy). The code combines the smoothed particle hydrodynamics (SPH) method to solve hydrodynamical equations with the popular Barnes and Hut (1986) tree-code to perform gravity calculation with a $N \times \log N$ scaling, and it is based on the scalar Tree-SPH code developed by Carraro *et al.* (1998). Parallelization is achieved by distributing particles along processors according to a workload criterion. Benchmarks of the code, in terms of load balance and scalability, are analysed and critically discussed against the adiabatic collapse of an isothermal gas sphere test using 2×10^4 particles on eight processors. The code turns out to be balanced at more than 95% level. If the number of processors is increased, the load balance worsens slightly. The deviation from perfect scalability at increasing number of processors is negligible up to 64 processors. Additionally we have incorporated radiative cooling, star formation, feedback and an algorithm to follow the chemical enrichment of the interstellar medium.

1. Introduction

Carraro, Lia and Chiosi (1998) developed a pure particle code, combining the Barnes and Hut (1986) octo-tree with SPH and applying this code to the formation of a spiral galaxy such as the Milky Way. The code is similar to the Hernquist and Katz (1989) TreeSPH. It uses SPH to solve the hydrodynamical equations.

In SPH a fluid is sampled using particles; there is no resolution limitation due to the absence of grids, but there is great flexibility due to the use of a time- and space-dependent smoothing length. Shocks are included by adopting an artificial viscosity tensor, and the neighbour search is performed using the octo-tree. The octo-tree, combined with SPH, allows a time scaling of $N \times \log N$. A real advantage of such codes is that it is easy to introduce new physics, such as cooling and radiative processes, magnetic fields, and so forth. Finally the kernel, which is utilised to perform hydrodynamical quantity estimates, can be made adaptive by using anisotropic smoothing lengths.

It is widely recognized that TreeSPH codes, although deficient in some aspects, can give reasonable answers in many astrophysical situations, such as simulations of fragmentation and star formation in giant molecular clouds (GMCs), supernrova explosions, globular cluster formation, merging of galaxies, galaxy and cluster formation and the Lyman alpha forest. Galaxy formation in particular requires a huge dynamical range (Davé, Dubinski and Hernquist, 1997). In fact an ideal galaxy formation simulation would start from a volume as large as the Universe to follow the initial growth of the cosmic structures and at the same time would be able to resolve regions as small as GMCs, where stars form and drive galaxy evolution through their interaction with ISM. This ideal simulation would encompass a dynamic range of 10^9 (from Gpc to parsec), 10^6 times smaller than that achievable with present-day codes.

Huge efforts have been made in recent years to widen as much as possible the dynamical range of numerical simulations, mainly using more and more powerful supercomputers. Scalar and vector computers indeed cannot handle efficiently a quantity of particles greater than half a million. Davé *et al.* (1997) developed for the first time a parallel implementation of a TreeSPH code (PTreeSPH) which can follow both collisionless and collisional matter. They report results of simulation runs on a Cray T3D computer of the adiabatic collapse of an initially isothermal gas sphere (using 4096 particles), of the collapse of a Zel'dovich pancake (32768 particles) and of cosmological simulation (32768 gas and 32768 dark particles). Their result are quite encouraging, being similar to those obtained with the scalar TreeSPH code (Hernquist and Katz, 1989).

Porting a scalar code to a parallel machine is far from being an easy task. A massively parallel computer (such as the Silicon Graphics T3E) links together hundreds or thousands of processors aiming at significantly increasing the computational power. For this reason they are very attractive, although several difficulties can arise in adapting a code to these machines. Any processor possesses its own memory and can assess other processors' memories by means of communications which are handled by a hardware network, and are slower than the computational speed. Great attention must be paid to avoiding situations in which a small number of processors is working while the rest are standing idle. Usually one has to invent a proper data distribution scheme allowing the subdivision of particles into processors in such a way that each processor handles about the same number of particles and does not need to make heavy communications. Moreover, the computational load must be shared between processors, thus ensuring that processors exchange information all together, in a synchronous way, or that each processor is performing different kinds of work when it is waiting for information coming from other processors, in an asynchronous fashion (Davé *et al.*, 1997).

In this paper we present a parallel implementation of the TreeSPH code described in Carraro *et al.* (1998). The numerical ingredients are the same as in the scalar version of the code. However, the design of the parallel implementations required several changes to the original code. The key idea that guided us

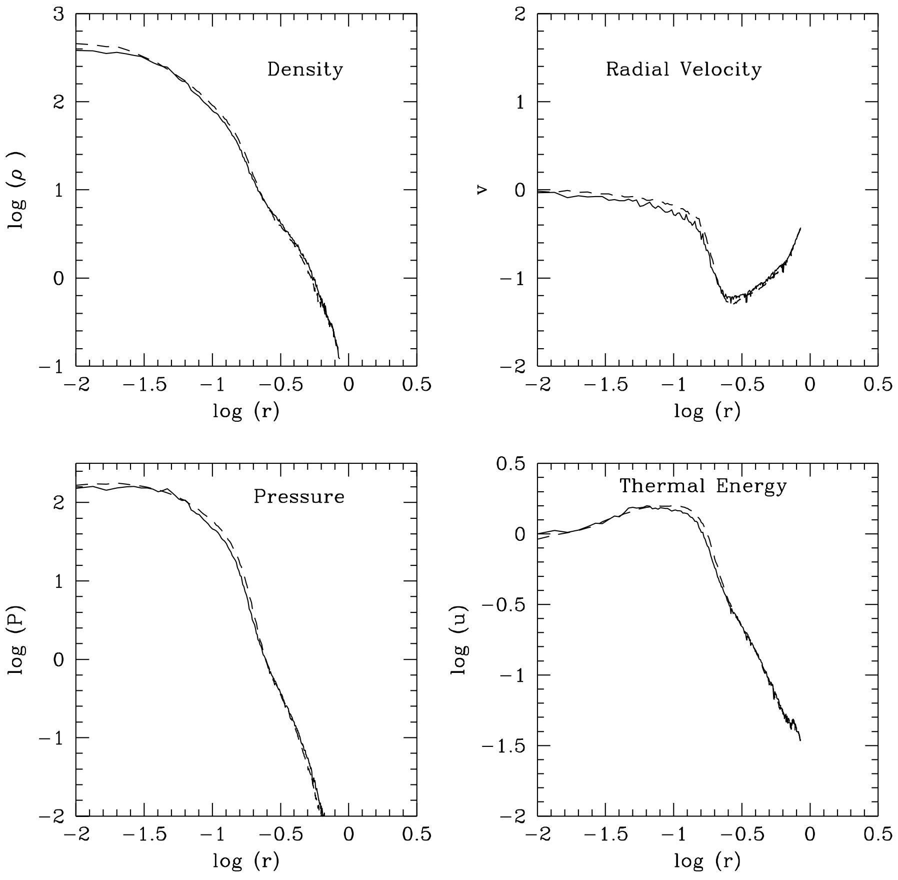

Figure 1. Adiabatic collapse: snapshots of the density, radial velocity, pressure and internal energy at the time of the maximum compression. The results of the test performed using 2×10^4 particles with eight processors are shown by *dashed lines*. *Solid lines* show the results obtained with the same number of particles, but using the scalar code, for comparison.

in building the parallel code was to avoid continuous communications, limiting the information exchange at a precise moment in the code flow. This clearly reduces the communications overhead. We have also decided to tailor the code to the machine, improving its efficiency. Since we are using a T3E massively parallel computer, a natural choice was to handle communications using the SHMEN libraries, which permit asynchronous communications and are intrinsically very fast, being produced directly by Cray for the T3E supercomputer. At present the code is also portable to other machines, such as the SGI Origin 2000, and will be portable to any other machine with the advent of the second release of the Message Passing Interface (MPI).

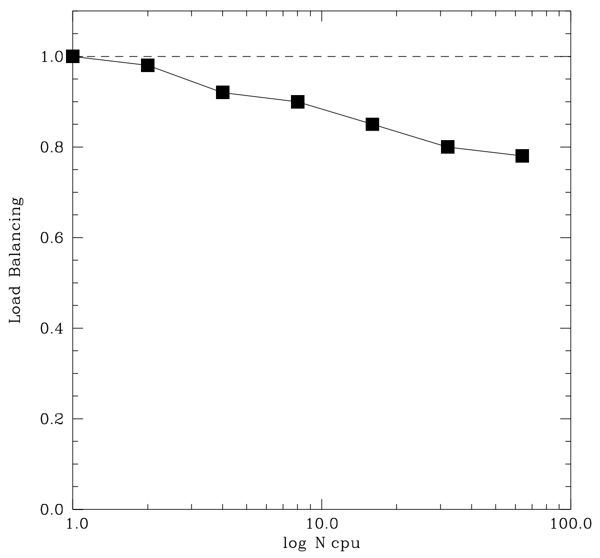

Figure 2. Overall code load balance, averaged over 50 time-steps (*solid line*). *Dashed line* indicates ideal scalability.

2. A Test of the Code

We consider the adiabatic collapse of an initially non-rotating isothermal gas sphere. This is a standard test for SPH codes (Hernquist and Katz, 1989). In particular, it is an ideal test for a parallel code, due to the large dynamic range and high density contrast. To facilitate the comparison of our results with those by the above authors, we adopt the same initial model and the same units ($M = R = G = 1$). The system consists of a $\gamma = 5/3$ gas sphere, with an initially isothermal density profile:

$$\rho(r) = \frac{M(R)}{2\pi R^2}\frac{1}{r}, \tag{1}$$

where $M(R)$ is the total mass inside the sphere of radius R. The density profile is obtained by stretching an initially regular cubic grid.

The total number of particles used in this simulation is 2×10^4. All the particles have the same mass. The specific internal energy is set to $u = 0.05GM/R$. For this test the viscosity parameters α and β adopted are 0.5, in agreement with Davé *et al.* (1997). The gravitational softening parameter, ϵ, adopted for this simulation is 5×10^{-3}. The state of the system at the time of the maximum compression is shown in the various panels of Figure 1, which displays the density, radial velocity, pressure and specific internal energy profiles. Each panel shows the variation of the physical quantity under consideration (in suitable units) as a function of the normalized radial coordinate at time equal to 0.88. The initial low internal energy is

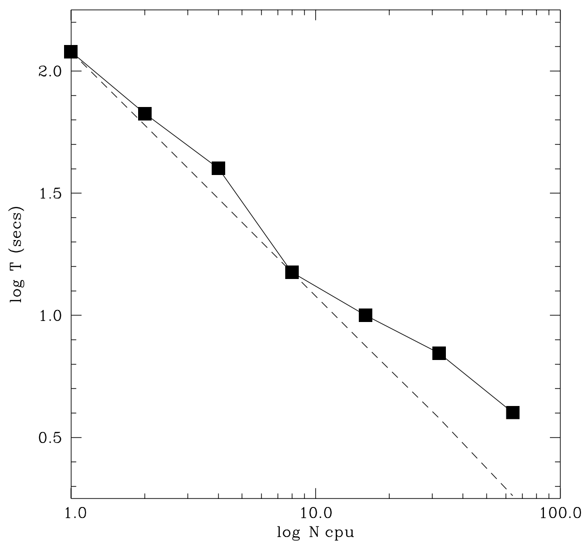

Figure 3. Overall code scalability, averaged over 50 time-steps (*solid line*). *Dashed* line indicates ideal load balance.

not sufficient to support the gas cloud, which starts to collapse. Approximately after one dynamical timescale, a bounce occurs. The system afterwards can be described as an isothermal core plus an adiabatically expanding envelope pushed by the shock wave generated at the stage of maximum compression. After about three dynamical times the system reaches virial equilibrium with total energy equal to a half of the gravitational potential energy (Hernquist and Katz, 1989). The present results agree fairly well with the mean values of the Hernquist and Katz (1989) simulations, which in turn agree with the 1D finite difference results.

We run the adiabatic collapse test up to the time of the maximum compression ($t \simeq 1.1$) using 2×10^4 particles on 1, 2, 4, 8, 16, 32 and 64 processors, and looked at the performances in the following code sections (see also Table I):

- Total wall-clock time
- Data up-dating data
- Parallel computation, which consists of barriers, the construction of the *ghost-tree* and the distribution of data among processors
- Search for neighbour particles
- Evaluation of the hydrodynamical quantities
- Evaluation of the gravitational forces

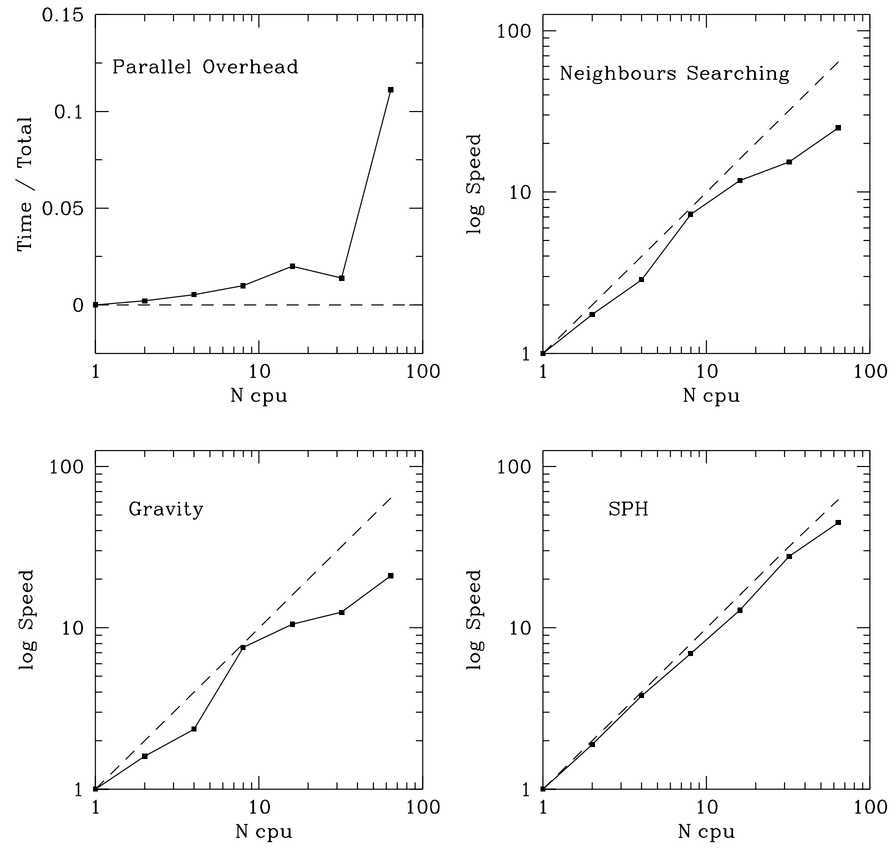

Figure 4. Code scalability in different code sections, averaged over 50 time-steps (*solid line*). *Dashed line* indicates ideal load balance.

• Miscellaneous, which encompasses I/O and kernel computation

The results summarized in Table I present the total wall-clock time per time-step per processor, averaged over 50 time-steps, together with the time spent in each of the five subroutines (data updating, neighbour searching, SPH computation, gravitational interaction and parallel computation). The gravitation interaction takes about one-third of the total time, while the search for neighbours takes a roughly comparable time. The evaluation of hydrodynamical quantities takes about one-fourth of the time, the remaining time being divided between I/O and data updating. The parallel overhead does not appear to be a problem, being always less than 1% of the total time. This timing refers, as indicated above, to simulations stopped at roughly the time of maximum compression. A run with eight processors up to $t \simeq 2.5$, the time at which the system is almost completely virialized, took 3800 s. One of the most stringent requirement for a parallel code is its capability to distrib-

TABLE I

The Adiabatic Collapse test. Benchmarks for a run with 2×10^4 particles. Time refers to 50 time-steps.

$N_{\rm cpu}$	Total (s)	Data Up-date (s)	Parallel Overhead (s)	Neighbours (s)	SPH (s)	Gravity (s)	Miscellaneous (s)
1	120	0.47	0.00	40	36	40	3.53
2	69	0.22	0.60	23	19	25	1.18
4	42	0.27	1.70	14	9.5	15	1.53
8	23	0.13	3.20	5.5	5.4	5.3	3.47
16	17.3	0.13	3.40	3.4	3.0	3.8	3.60
32	11.5	0.09	3.00	2.6	1.3	3.2	1.31
64	7.5	0.05	2.90	0.33	1.1	1.9	1.23

ute the computational work equally among all the processors. This can be done by defining a suitable workload criterion, as discussed in Section 3.2. This is far from being an easy task (Davé *et al.*, 1997), and in practice some processors stand idle for some time waiting for those processors with the greatest computational load to accomplish their task. This is also true when an asynchronous communication scheme is adopted, as in our TreeSPH code. To evaluate the code load balance we adopted the same strategy as Davé *et al.* (1997) of measuring the fractional amount of time spent idle in a time-step while another processor performs computation:

$$L = \frac{1}{N_{\rm procs}} \sum_{j=1}^{N_{\rm procs}} 1 - \frac{(t_{\rm max - t_i})}{t_{\rm max}}. \tag{2}$$

Here $t_{\rm max}$ is the time spent by the slowest processor, while t_i is the time taken by the ith processors to perform computation. The results are shown in Figure 2, where we plot the load balance for simulations for an increasing number of processors, from 1 to 64. The load balance is always maintained above 80%, being close to 1 for up to eight processors. For the kind of simulation we are performing, the use of eight processors is particularly advantageous for reasons of symmetry. For increasing numbers of processors, a parallel code should ideally speed up linearly. In practice the increase in the number of processors causes an increase in the communications among processors and a degradation of the code performance. To test this, we used the same simulations discussed above, running the adiabatic collapse test with 2×10^4 particles for increasing processor numbers. We estimated how the code speed scales computing the wall-clock time per processor spent in executing a single time-step, averaged over 50 time-steps. In Figure 3 we plot the speed (in s^{-1} against the number of processors.

The code scalability stays very close to the ideal scalability up to eight processors, where it shows a minimum. This case is in fact the most symmetric one. The scalability then deviates significantly only when using more that 16 processors. Looking also at Figure 4, it is easy to recognize that the gravitational interaction code is mainly responsible for this deviation.

In the near future we are going to investigate whether the code's overall scalability might be improved by adding new physics (e.g. cooling and star formation), which is necessary to describe the evolution of real systems such as galaxies.

References

Barnes, J.E. and Hut, P.: 1986, *Nature* **324**, 446.
Carraro, G., Lia, C. and Chiosi, C.: 1998, *Mon. Not. R. Astron. Soc.* **297**, 1029.
Davé, R., Dubinski, J. and Hernquist, L.: 1997, *New Astron.* **2**, 277.
Hernquist, L. and Katz, N.: 1989, *Astrophys. J. Suppl.* **70**, 419.