UNIVERSITÀ DI PADOVA FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

Scuola di Dottorato di Ricerca in Ingegneria dell'Informazione

Indirizzo: Ingegneria Informatica ed Elettronica Industriali

XXI Ciclo

Oblivious Computations on Memory and Network Hierarchies

Direttore della Scuola: Ch.mo Prof. Matteo Bertocco [illegible]

Supervisore: Ch.mo Prof. Andrea Alberto Pietracaprina

[illegible]

Dottorando: Francesco Silvestri

[illegible]

Sommario

L'organizzazione gerarchica dei sistemi di memoria e di comunicazione e la disponibilità di molte unità di calcolo influiscono notevolmente sulle prestazioni di un algoritmo. Il loro efficiente utilizzo è limitato dalle differenti configurazioni che possono assumere. È cruciale, per motivi economici e di portabilità, che gli algoritmi si adattino allo spettro delle piattaforme esistenti e che la procedura di adattamento sia il più possibile automatizzata. L'adattività può essere raggiunta sia tramite algoritmi aware, che utilizzano esplicitamente opportuni parametri architetturali, sia tramite algoritmi oblivious, la cui sequenza di operazioni è indipendente dalle caratteristiche dell'architettura sottostante. Gli algoritmi oblivious esibiscono spesso prestazioni ottime su diverse architetture e sono attrattivi soprattutto nei contesti in cui i parametri architetturali sono difficili da stimare o non sono noti.

Questa tesi si focalizza sullo studio degli algoritmi oblivious con due obiettivi principali: l'indagine delle potenzialità e limitazioni delle computazioni oblivious e l'introduzione del concetto di algoritmo oblivious in un sistema parallelo.

Inizialmente, vengono affrontate varie problematiche legate all'esecuzione di algoritmi cache-oblivious per permutazioni razionali, le quali rappresentano un'importante classe di permutazioni che include la trasposizione di matrici e il bit-reversal di vettori. Si dimostra un lower bound, valido anche per algoritmi cache-aware, sul numero di operazioni macchina necessarie per eseguire una permutazione razionale assumendo un numero ottimo di accessi alla memoria. Quindi, si descrive un algoritmo cache-oblivious che esegue qualsiasi permutazione razionale e che richiede un numero ottimo di operazioni macchina e di accessi alla memoria, assumendo l'ipotesi di tall-cache. Infine, si dimostra che per certe famiglie di permutazioni razionali (tra cui la trasposizione e il bit-reversal) non può esistere un algoritmo cache-oblivious che richieda un numero ottimo di accessi alla memoria per tutti i valori dei parametri della cache, mentre esiste un algoritmo cache-aware con tali caratteristiche.

Nella tesi viene poi introdotto il framework network-oblivious per lo studio di algoritmi oblivious in un sistema parallelo. Il framework esplora lo sviluppo di algoritmi paralleli di tipo bulk-synchronous che, senza usare parametri dipendenti dalla macchina, hanno prestazioni efficienti su macchine parallele con differenti gradi di parallelismo e valori di banda/latenza. Vengono inoltre forniti algoritmi network-oblivious per alcuni problemi chiave (moltiplicazione e trasposizione di matrici, trasformata discreta di Fourier, ordinamento) e viene presentato un risultato di impossibilità sull'esecuzione di algoritmi network-oblivious per la trasposizione di matrici che ricorda quello ottenuto per le permutazioni razionali.

Infine, per mostrare ulteriormente le potenzialità del framework, vengono presentati algoritmi network-oblivious ottimi per eseguire un'ampia classe di computazioni risolvibili tramite il paradigma di programmazione ad eliminazione gaussiana, tra cui il calcolo dei cammini minimi in un grafo, l'eliminazione gaussiana senza pivoting e la moltiplicazione di matrici.

Abstract

The hierarchical organization of the memory and communication systems and the availability of numerous processing units play an important role in the performance of algorithms. Their actual exploitation is made hard by the different configurations they may assume. It is crucial, for economical and portability issues, that algorithms adapt to a wide spectrum of executing platforms, possibly in an automatic fashion. Adaptivity can be achieved through either aware algorithms, which make explicit use of suitable architectural parameters, or oblivious algorithms, whose sequence of operations is independent of the characteristics of the underlying architecture. Oblivious algorithms are more desirable in contexts where architectural parameters are unknown and hard to estimate, and, in some cases, they still exhibit optimal performance across different architectures.

This thesis focuses on the study of oblivious algorithms pursuing two main objectives: the investigation of the potentialities and intrinsic limitations of oblivious computations in comparison with aware ones, and the introduction of the notion of oblivious algorithm in the parallel setting.

We study various aspects concerning the execution of cache-oblivious algorithms for rational permutations, an important class of permutations including matrix transposition and the bit-reversal of a vector. We provide a lower bound, which is also valid for cache-aware algorithms, on the work complexity required by the execution of a rational permutation with an optimal cache complexity. Then, we develop a cache-oblivious algorithm performing any rational permutation, which exhibits optimal work and cache complexities under the tall-cache assumption. We then show that for certain families of rational permutations (including transposition and bit-reversal) no cache-oblivious algorithm can exhibit optimal cache complexity for all values of the cache parameters, while there exists a cache-aware algorithm with this property.

We introduce the network-oblivious framework for the study of oblivious algorithms in the parallel setting. This framework explores the design of bulk-synchronous parallel algorithms that, without resorting to parameters for tuning the performance on the target platform, can execute efficiently on parallel machines with different degree of parallelism and bandwidth/latency characteristics. We illustrate the framework by providing optimal network-oblivious algorithms for a few key problems (i.e., matrix multiplication and transposition, discrete Fourier transform and sorting) and by presenting an impossibility result on the execution of network-oblivious algorithms for matrix transposition, which is similar, in spirit, to the one provided for rational permutations.

Finally, we present a number of network-oblivious algorithms, which exhibit optimal communication and computation complexities, for solving a wide class of computations encompassed by the Gaussian Elimination Paradigm, including all-pairs shortest paths, Gaussian elimination without pivoting and matrix multiplication.

Ringraziamenti

Ecco, finalmente, la pagina che ho desiderato a lungo di scrivere e che sicuramente sarà la più letta. Negli anni trascorsi come dottorando nel Dipartimento di Ingegneria Informatica ho incontrato persone eccezionali con cui ho condiviso bellissimi momenti e che mi hanno aiutato a crescere professionalmente e umanamente. Sono riconoscente a tutte queste persone e ad alcune di loro voglio dedicare un pensiero.

Il primo grazie va ad Andrea. Mi ritengo fortunato ad aver avuto Andrea come advisor in questi anni: è sempre stato disponibile a consigliarmi e ad aiutarmi, facendo sempre più del suo dovere. Lo ringrazio soprattutto per aver cercato di migliorare, e sopportato, il mio modo non lineare di pensare accompagnato da un flusso impulsivo di parole. Ringrazio poi Geppino per il suo corso di Fondamenti di Informatica II, che mi ha introdotto all'algoritmica, e per essere stato un ottimo co-advisor.

Un ringraziamento speciale va alla vecchia guardia di dottorandi e assegnisti del laboratorio ACG: Alberto B., Fabio, Francesco, Marco e Paolo; con loro ho condiviso i momenti migliori e più allegri del mio dottorato e stretto bellissime amicizie. In particolare sono grato a Fabio per le lunghe chiacchierate e per avermi contagiato con la sua passione per la ricerca (prima o poi pubblicheremo qualcosa insieme!), e a Marco per avermi ricordato che le risorse della Terra sono finite. Sono poi riconoscente a Alberto P., Carlo, Enoch, Gianfranco, Mattia e Michele per le interessanti discussioni (quelle di Gianfranco anche particolarmente lunghe) e ad Alessandra, il cuore della Scuola di Dottorato, per la sua gentilezza e disponibilità.

Non posso dimenticare Bruno che con tutta la sua energia mi ha permesso di capire e apprezzare gli Stati Uniti D'America.

Un grazie speciale va ad Adriano per essere stato un vero amico in tutti questi anni e per tutte le birre bevute insieme. Ringrazio poi Alberto e Martina per aver condiviso l'appartamento, ma soprattutto per aver sopportato i miei goffi tentativi di far saltare la pasta in padella.

Infine, voglio dire grazie alle tre persone più importanti della mia vita, anche se non sono in grado di esprimere quello che sono stati e hanno fatto per me: senza di loro non avrei mai provato quei sentimenti di cui ora non posso più fare a meno. A mia mamma Fanny e a mio papà Giuliano: il vostro amore è stato un punto di riferimento e un aiuto in tutti questi anni; senza di voi, non sarei arrivato fin qui. E ad Elisa, la mia futura moglie: grazie per avermi sempre ascoltato, consigliato ed essere stata al mio fianco; grazie a te ora sono Felice.

Acknowledgements

It is very difficult for me to translate my feelings into English, but I would like to thank the few people that cannot understand my language. I'm grateful to Vijaya Ramachandran for inviting me to spend some months at the University of Texas at Austin and for the useful discussions. I also thank Rezaul Alam Chowdhury for his collaboration.

Contents

Chapter 1

Introduction

You have to be fast on your feet and
adaptive or else a strategy is useless.
(Charles de Gaulle)

Since the advent of computers, the request of computational power from applications has been continuously growing and has increased significantly in the last twenty years with the emergence of novel applications in computational sciences as well as in many other contexts. This phenomenon has posed formidable challenges to both computer architects and software/algorithm designers, and much progress has been made in the development of architectures, algorithms and compilers. Nevertheless, the potential power of actual platforms can hardly be fully exploited.

One of the main reasons which limit the efficiency of an application is the so-called *memory wall*. While the clock speed of functional units has been growing exponentially, at least until recent years when physical limits have been approached [ABC$^+$06], the advances in the realization of memory supports have been less impressive. This phenomenon has increased the gap in efficiency between functional units and memory supports, which is the main cause of inefficiency in sequential applications [Bac78]. Several architectural approaches have been adopted to mitigate the impact of this gap. The traditional approach consists in the introduction, between the main memory and the functional unit(s), of several levels of *caches*, which are fast yet small memories [HP06]. The main justification for this hierarchical organization of the memory system is that an algorithm typically reuses small sets of data within relatively short time intervals, a property referred to as *temporal locality of reference*. Thus, the time required for retrieving these data can be decreased if they are stored in faster caches. Another important aspect is the movement of data in batches of contiguous segments (*blocks*) between adjacent levels of the memory hierarchy. The motivation is that an algorithm usually requires data that reside in contiguous memory locations within a short time interval, a property referred to as *spatial locality of reference*. Since accessing all data in one shot is less expensive than performing several independent accesses, it is convenient to move data in blocks.

A number of compilers, prefetching techniques and runtime speculative analyses [KA02, HP06] have been introduced to reduce the programmers' effort when dealing with the memory hierarchy. Although, these approaches yield some performance improvements, more substantial optimizations can be made at the algorithm design level. For this reason, a number of computational models have been introduced to help programmers develop algorithms which perform efficiently in memory hierarchies (see [Vit01, MSS03] and references therein). These models aim at representing only the most important features of the hierarchy in order to simplify the design process. Examples are the *External Memory* model of Aggarwal and Vitter [AV88] and the *Ideal Cache* model of Frigo et al. [FLPR99]. The first represents a disk-RAM hierarchy [Vit01, GVW96, SN96], while the second a RAM-cache one [Dem02, ABF05].

As already noted, the clock speed cannot be increased indefinitely since the finite speed of light and thermodynamic laws impose physical limitations. A natural solution to increase the number of operations executed per time unit is the resort to *parallel* architectures. Parallelism has been studied for decades [JáJ92, Lei92, CSG98], and has been explored at the bit level, instruction level, and thread level. Many compilers [Wol96] and languages [MHS05] have been proposed to help programmers extract parallelism automatically from their applications, but they do not always provide optimal solutions. Indeed, the best approach is to expose parallelism explicitly at the algorithm design level. However, there is practical evidence that this is not an easy task. Furthermore, the situation is made more complex by the lack of a widely accepted model of computation which strikes a good balance between the conflicting requirements of *usability*, *effectiveness* and *portability*. In the sequential setting one such model has been constituted for many years by the Random Access Machine based on the Von Neumann organization [AHU74]. Unfortunately, in the parallel setting there is still no agreement on a common model for algorithm design and analysis, although some proposals have received considerable attention and a broad consensus. Among them we recall the *Bulk Synchronous Parallel* (*BSP*) model by Valiant [Val90] and successive extensions, like the *Decomposable-BSP* (*D-BSP*) model by De la Torre and Kruskal [DlTK96], which have been both extensively studied (see [BPP07] and references therein).

A major obstacle to the achievement of high performance in parallel architectures, is represented by the cost of *communication*. Since the relevance of this factor increases with the size of the system, communication will play an even greater role in future years. Thus, reducing the communication requirements of algorithms is of paramount importance, if they have to run efficiently. The cost of communication is in general dependent on the distance between the processors involved, and in typical parallel architectures processors communicate through a hierarchical interconnection featuring different bandwidth and latency characteristics at different levels. Thus programs run efficiently if they are able to confine data exchanges within small regions of the interconnection, a property referred to as *submachine locality*. This scenario is reflected in several models [Lei85, BP97, BP99, ACS90, CKP$^+$96], and constitute the fundamental grounds upon which the aforementioned D-BSP model, which effectively describes a very large and significant class of parallel platforms [BPP07], has been defined.

Therefore, it is evident that parallelism on the one hand and the hierarchical organization of the memory and of the interconnection on the other play a key role in the performance of applications, and their combination has become even more evident with the advent of *Chip Multi-Processors* [ONH$^+$96]. As we have already observed, optimizations based on these factors can be reached, to some extent, by automatic techniques, but the best results are obtained through a judicious algorithm design process. In aggressive optimizations, an algorithm is tuned to match almost perfectly the characteristics of the target machine. Although this approach is common in applications which require expensive or special purpose platforms (such as *IBM BlueGene/L* [Aea02] and *APE* computers for Quantum Chromo Dynamics [BPP$^+$05]), in general it is undesirable for economical and portability reasons. Moreover, an aggressive optimization could not be possible in scenarios, like grid environments [FK03], where software may be required to run, for availability or load management purposes, on widely different configurations, which may be revealed only at runtime. An alternative approach, which is becoming more and more popular, is provided by *adaptive algorithms*, that is, algorithms which adapt themselves to the target machine where they run. We can distinguish two different kinds of adaptive algorithms: *aware* and *oblivious*.

Aware adaptive algorithms use machine dependent parameters that are set at run time to match adaptively the structure of the actual platform. Some libraries that follow this approach are: *Fast Fourier Transform in The West* (*FFTW*) [FJ98] for the discrete Fourier transform (DFT), *Automatically Tuned Linear Algebra Software* (*ATLAS*) [WPD01] for linear algebra problems, *Finite-Elements Multifrontal Solver* (*FEMS*) [Ber08] for applications based on the Finite-Elements method. The FFTW library computes a *plan* for the machine where it is running, which provides information on how the DFT can be efficiently computed on that machine. If the machine does not change, the plan can be reused for many transforms, amortizing the cost of plan building. ATLAS is an implementation of a style of performance software production called *Automated Empirical Optimization of Software* (*AEOS*) [WPD01]; in an AEOS-enabled library, many different ways of performing a given kernel operation are supplied, and timers are used to empirically determine which implementation is best for a given architecture. FEMS is a solver designed for distributed-memory parallel machines and arranges the computation based on the organization of the memory system and on the subdivision of processors into clusters.

Oblivious adaptive algorithms are algorithms where no parameters dependent on machine characteristics need to be tuned to achieve efficiency, and still exhibit high performance on several machines. This tape of algorithms has recently been introduced by Frigo et al. in [FLPR99] through the notion of *cache-oblivious algorithms*. These algorithms are designed for the Random Access Machine, but are analyzed on the Ideal Cache model. Cache-oblivious algorithms are desirable because if optimal in the Ideal Cache model they can be proved, under some assumptions, to be optimal also in multi-level memory hierarchies. Cache-oblivious algorithms have been proposed for many problems (see [Dem02, ABF05] and references therein), and they have also been adopted in the commercial software *Tokutek* [Tok]. Intuitively, a key paradigm for cache-oblivious algorithms is *divide and conquer*: problems are recursively decomposed into smaller problems which, at some point of the recursion, will fit in the cache of the machine. However, it must be said that some problems can be solved optimally by non-recursive cache-oblivious algorithms.

Although the aware approach in practical settings reaches better results than the oblivious one [YRP$^+$07], we think that the latter embodies valuable ideas and opportunities. As noted in [Fri99], with reference to cache-oblivious computations, since divide and conquer is advantageous for portable high-performance programs, architectural innovations and compiler techniques to reduce the cost of procedure calls are to be sought. An improvement in this direction will reduce the practical gap between oblivious and aware algorithms, and make oblivious algorithms more appealing for their higher portability. Another advantage of obliviousness over awareness is that it requires no parameters to be tuned. Hence the executing platform can be dynamically changed without requiring a parameter recalibration to adapt the algorithm to the new machine, a property very desirable in grid or global computing environments [Aeo, FK03].

This thesis focuses on the study of oblivious algorithms pursuing two main objectives. The first objective is to further investigate the potentialities and intrinsic limitations of oblivious versus aware algorithms. The second objective is to introduce the notion of oblivious computation in the parallel setting.

In principle, aware algorithms are able to use the target platforms more effectively than oblivious algorithms, since they have more knowledge about the system on which they are running. Nevertheless, for several notable problems the oblivious approach is asymptotically equivalent to the aware one [FLPR99, Dem02, ABF05]. In [FLPR99], the authors ask if there exists a separation in asymptotic complexity between cache-aware and cache-oblivious algorithms. Few works have recently made contributions in this direction. Bilardi and Peserico [BP01] provided a first study of this gap in the context of DAG computations on the *Hierarchical Memory* model [AACS87], and, recently, Brodal and Fagerberg [BF03] have proved that cache-oblivious algorithms for sorting and permuting cannot be optimal for a full range of values of the cache parameters.

Inspired by these works, we have looked for other problems where the aware and oblivious approaches exhibit an asymptotical separation. Specifically, in the thesis we consider an important class of permutations. Permuting a vector is a fundamental primitive that arises in many problems; in particular the so-called *rational permutations* are widely used. A permutation is rational if it is defined by a permutation of the bits of the binary representations of the vector indices. Matrix transposition, bit-reversal, and some permutations implemented in the *Data Encryption Standard* (*DES*) [FIP99] are notable examples of rational permutations. We study the execution of these permutations in cache hierarchies, with particular emphasis on the cache-oblivious settings. We first prove a lower bound on the work needed to execute a rational permutation with an optimal cache complexity. Then, we develop a cache-oblivious algorithm to perform any rational permutation, which exhibits optimal work and cache complexities in those cache that satisfy the tall-cache assumption [FLPR99], that is where the cache size is at least the square of the block length. We finally show that for certain families of rational permutations (including matrix transposition and bit-reversal) no cache-oblivious algorithm can exhibit optimal cache complexity for all values of the cache parameters, while this is attainable through a cache-aware algorithm. Our results and the ones in [BF03] provide evidence that, in some cases, the tall-cache assumption is required in order to achieve cache optimality in an oblivious fashion. While caches in modern architectures are usually tall, this in general is not the case for the *Translation Lookaside Buffer* (*TLB*) which can be regarded as a type of cache [HP06, Kum03]. Hence these results may shed some light on the interaction that algorithms have with the TLB, highlighting some non-optimal behaviors.

To the best of our knowledge, cache-oblivious algorithms constitute the only example of oblivious adaptiveness studied in the algorithmic field. A prominent scenario where an oblivious approach is desirable is that of parallel computations. Attempts to introduce the notion of oblivious computation in the parallel setting have recently been made in [BFGK05, FS06, CR08]. However, most parallel algorithms adapt in an aware fashion to parameters that relate to parallelism in the architecture, such as, for example, processor number and bandwidth/latency of the interconnection. In this thesis, we define an environment for the design, analysis and execution of oblivious parallel algorithms, which we refer to as the *network-oblivious framework*. This framework has been inspired by the following vision of cache-oblivious algorithms. Cache-oblivious algorithms can be studied by means of three models: the *specification model* (the Random Access Machine), where the only parameter is the input size; the *evaluation model* (the Ideal Cache), which consists of a two-level memory hierarchy and specifies as parameters the cache size and the block length; the *execution machine model* which features a multi-level hierarchy and represents more accurately actual architectures. A cache-oblivious algorithm is designed in the specification model; its cache complexity is assessed in the evaluation model and optimality in the evaluation model translates into optimality in the execution model, under certain assumptions.

In a similar fashion, the network-oblivious framework is based on three distinct models: the specification model where algorithms are developed, consisting of a clique of processor/memory pairs, whose number is function exclusively of the input size and which support a bulk-synchronous programming style; the evaluation model where algorithms are analyzed, which is similar to the specification model but provides two parameters, namely processor number and communication block-size, which capture parallelism and granularity of communication, respectively; the execution model where algorithms are eventually executed, which is a block-variant of the D-BSP model. We prove that, for a wide class of network-oblivious algorithms, optimality in the evaluation model implies optimality in the execution model. We substantiate our study by providing optimal network-oblivious algorithms for some fundamental computational kernels, namely matrix multiplication and transposition, discrete Fourier transform and sorting. We also prove a result on the separation between the oblivious and the aware approaches: specifically, we show that no network-oblivious algorithm for matrix transposition can be optimal for all values of the evaluation model parameters, while this is attainable through an aware parallel approach.

In this thesis, we further show the potential of the network-oblivious approach for an important class of applications. Specifically, as argued in [CR06], notable problems such as all-pairs shortest paths, Gaussian elimination without pivoting and matrix multiplication, can be solved by a generic algorithmic paradigm referred to as the *Gaussian Elimination Paradigm* (*GEP*). Building on previous parallel cache-oblivious algorithms developed in [CR07, CR08], which correctly implement several GEP computations, we derive three network-oblivious algorithms, named *N-GEP*, *ϵN-GEP* and *PN-GEP*, which correctly solve a wide class of GEP computations, included the aforementioned ones. All of these algorithms exhibit optimal performance in the evaluation model and, furthermore, we show that one of them, namely N-GEP, performs optimally also on a D-BSP model for certain significant ranges of parameters.

The remaining part of the thesis is structured a follows. Chapter 2 presents an overview of models for memory and communication hierarchies and describes the three computational models that will be used in the thesis, namely the External Memory, the Ideal Cache and the Decomposable-BSP models. Chapter 3 presents the results regarding the execution of rational permutations in cache hierarchies. Chapter 4 introduces the network-oblivious framework and network-oblivious algorithms for the aforementioned computational kernels. Chapter 5 provides the three network-oblivious algorithms for GEP computations. Finally, Chapter 6 concludes with a brief summary and some open problems.

Part of the results contributed by the thesis have been published in the open literature. Results described in Chapter 3 appeared as a preliminary version in [Sil06] and, in the current form, in [Sil08]. The contents of Chapter 4 are based on a joint work with Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci, which appeared in [BPPS07]. Part of the results in Chapter 5 are a joint work with Rezaul Alam Chowdhury and Vijaya Ramachandran. The research activities of the thesis have been supported by the EU/IST Project "*AEOLUS*", by MIUR of Italy under the project "*MAINSTREAM*" and the fellowship "*Legge 170*". The research that led to the results in Chapter 5 was also supported by NSF Grants *CCF-0514876* and *CCF-0850775* and carried out in part while the candidate was visiting the Department of Computer Sciences of the University of Texas at Austin.

Chapter 2

Models for Memory and Communication Hierarchies

Martin's Law of Communication. The inevitable result of improved and enlarged communication between different levels in a hierarchy is a vastly increased area of misunderstanding.
(Thomas L. Martin, Jr.)

As seen in Chapter 1, the idea behind memory and communication hierarchies is that good (sequential and parallel) algorithms exhibit high *locality* that can be exploited by the hardware in the following way:

- An algorithm usually refers to the same data within relatively small time intervals (*temporal locality of reference*), hence it is convenient to store this data in the fastest levels (which are also the smallest ones) to decrease the access time; moreover, contiguous memory locations are involved in consecutive operations (*spatial locality of reference*), thus moving segments of consecutive data between memory levels can decrease latency.

- Communications performed by a parallel algorithm are usually enclosed in small areas of the network (*submachine locality*): by means of a hierarchical interconnection, communications within small areas can exploit high bandwidth and low latency.

Algorithms, data structures and models for memory and communication hierarchies have been studied for many years, resulting in a large body of works appeared in the literature. The aim of this chapter is not to give a survey on the argument, but to focus on the models that will be used in this dissertation. Section 2.1 describes two models for memory hierarchy: the External Memory model of Aggarwal and Vitter [AV88] in Section 2.1.1; the Ideal Cache model and the cache-oblivious framework of Frigo et al. [FLPR99] in Section 2.1.2. Section 2.2 focuses on the model for communication hierarchy originally introduced by De la Torre and Kruskal [DlTK95, BPP07] which is called Decomposable Bulk Synchronous Parallel model.

In this section, the models for memory and communication hierarchies are analyzed separately, as it is common in the literature. However, it deserves to be noticed that a number of works in the literature (e.g., [PPS06, Sil05, FPP06, DHMD99, SK97]) have explored the relations between locality of reference and submachine locality. This thesis does not cover this topic, but some remarks will be made in Chapter 4, where we describe the network-oblivious framework, which embodies a suitable adaptation of the ideas at the core of the cache-oblivious one.

2.1 Models for memory hierarchy

The typical memory hierarchy of a uniprocessor consists of many levels: it is common to find two or three levels of caches, an internal memory (the traditional *Random Access Memory* [AHU74]), and an external memory (an hard disk or a tape). Due to technological constraints, the levels which are close to the CPU have small access times, but small capacities as well. Furthermore, for latency hiding, data movements between adjacent memory levels involve *blocks* of consecutive memory locations, a feature usually referred to as *block transfer*. Thus, an algorithm must expose temporal and spatial locality of reference in order to take advantage of such a hierarchical organization.

Many models have been proposed to explicitly account for the hierarchical nature of the memory system. We can divide these works basically in three classes: models that represent a disk-RAM hierarchy, models that represent a RAM-cache hierarchy, and models that feature a multilevel hierarchy (i.e., more than two levels).

According to Vitter [Vit01] the study of relations between disk and memory started with Demuth's Ph.D. on sorting [Dem56, Dem85] and the Knuth's extensive study [Knu98] of sorting on magnetic tapes and disks. At about the same time, Floyd defined a model which consists of a memory of M words and blocks of length $M/2$ [Flo72]. Hong and Kung [HK81] defined a pebbling game for proving lower bounds in a model without the block transfer feature, which was later taken into account by Savage and Vitter [SV87]. One of the most famous models is the *External Memory* (EM) model by Aggarwal and Vitter [AV88], which features a two-level disk-RAM hierarchy where P blocks can be transferred in parallel between the disk and the RAM; they also proposed tight lower and upper bounds for sorting, matrix transposition and the Fast Fourier Transform (FFT). This model was improved by the Vitter's *Parallel Disk* model [VS94], which allows P concurrent block transfers from P disks. Surveys on the disk-RAM models appeared in [Arg04, Vit01, GVW96, SN96].

The RAM-cache hierarchy is similar to the disk-RAM one, except that data movements cannot be controlled explicitly by the algorithm but are orchestrated by some automatic policy. Frigo et al. [FLPR99] proposed the *Ideal Cache* (IC) model, a two-level hierarchy composed of a memory and a fully-associative cache with an optimal replacement policy [Bel66]. In [FLPR99], the authors pioneer the study of a class of algorithms called *cache-oblivious algorithms*, which aim at obtaining optimal performance on different caches without explicitly using the cache parameters. Other models which extend the IC model taking into account the limited associativity of caches were proposed by Mehlhorn and Sanders [MS00] and by Sen et al. [SCD02].

Finally, some models feature a complex memory hierarchy. In the *Hierarchical Memory* (HM) model of Aggarwal et al. [AACS87], the access time to location x in memory requires time $f(x)$, where f is a non decreasing function. The block transfer feature was added to the HM model in [ACS87]. These two models can be considered a continuous version of memory hierarchy. Alpern et al. [ACFS94] defined the *Uniform Memory Hierarchy* where the parameters of a multilevel memory (i.e., bandwidth, memory size and block length) grow at uniform rates.

In the following sections we will use matrix multiplication as an example for understanding the EM and IC models: we denote with n-MM the multiplication of two $n \times n$ matrices using only semiring operations. We suppose all matrices to be stored in the slowest level according with a row-major layout (other layouts, like the column-major and the bit-interleaved, can be adopted as well).

2.1.1 External Memory model

The *External Memory* model ($\mathsf{EM}(M, B)$) of Aggarwal and Vitter [AV88] features two levels of memory: a (fast) RAM memory of M words and an arbitrarily large (slow) disk. The processor can only reference words that reside in RAM. The disk is divided into blocks of B adjacent words called *B-blocks*, or simply *blocks* if B is clear from the context. An *input operation* moves a B-block of the disk into B words of the RAM, and an *output operation* moves B words of the RAM into a B-block of the disk. The input/output operations (*I/Os*) are explicitly controlled by the algorithm and the *I/O complexity* of an EM algorithm is the number of I/Os performed by the algorithm.

Note that the above definition is slightly different from the one given in [AV88] for two reasons. Firstly, in the described model the starting position of a B-block in the disk is fixed[1], while in the original model a block can start in any position; however, as noted in [AV88], the two versions are equivalent from the point of view of asymptotic analysis. Secondly, in the original model at most P I/Os are performed concurrently but, as noted by Vitter in [Vit01] this kind of parallelism is unrealistic in that the P I/Os are allowed to take place in the same disk (the Parallel Disk model improves in some sense the original specification of the EM by introducing concurrent disks).

As mentioned above, we describe an EM algorithm for the n-MM problem.

Theorem 2.1. *There exists an optimal EM algorithm for the n-MM problem whose I/O complexity on an $\mathsf{EM}(M,B)$ is*

$$\Theta\left(\frac{n^3}{B\sqrt{M}}+\frac{n^2}{B}+1\right).$$

Proof. Let A and B be the two input matrices and let C be the output matrix, which we suppose to be initially set to zero. For simplicity, we suppose $n^2 > M$ and $M > B^2$ (for the remaining cases we refer to [GVL96]). Consider A (resp., B and C) divided into submatrices of size $s \times s$ where $s = \alpha\sqrt{M}$, with α a suitable constant in $(0,1)$, and denote each submatrix with $A_{i,j}$ (resp., $B_{i,j}$ and $C_{i,j}$), for $0 \leq i,j < n/s$. The algorithm is composed of $(n/s)^3$ steps, and in each one the algorithm fetches $A_{i,k}$, $B_{k,j}$ and $C_{i,j}$ from the disk, for suitable values of i, j and k, and computes $C_{i,j} \leftarrow C_{i,j} + A_{i,k} \cdot B_{k,j}$ with the traditional $\Theta(s^3)$ iterative algorithm; then data are written into the disk. The correctness of the algorithm is straightforward and the I/O complexity of the algorithm is $O\left(n^3/B\sqrt{M}\right)$, since $O\left(s^2/B\right) = O\left(M/B\right)$ I/Os are required for loading and writting submatrices from and to the disk. Optimality descends from [HK81]. □

EM algorithms for matrix transposition, sorting and FFT are given in [AV88], while [Vit01] provides a complete survey.

2.1.2 Ideal Cache model

The *Ideal Cache* model ($\mathsf{IC}(M,B)$) was introduced by Frigo et al. in [FLPR99] and consists of an arbitrarily large main memory and a (data) cache of M words. The

[1] Loosely speaking, the initial position of the B-block containing memory location x is $x \mod B$.

main memory is organized into blocks of B adjacent words called *B-blocks*, or simply *blocks* if B is clear from the context. The cache is fully associative and organized in $M/B > 1$ lines of B words each. At any time during the execution of an algorithm, a line is either empty or it contains a B-block of the memory. The processor can only reference words that reside in cache: if a referenced word belongs to a block in a cache line, a *cache hit* occurs; otherwise there is a *cache miss* and the block has to be copied into a line, replacing the line's previous content. The model adopts an optimal off-line replacement policy, that is, it replaces the block whose next access is furthest in the future [Bel66]. It is easy to see that an IC algorithm is formulated as a traditional RAM algorithm. We denote as *work complexity* of an algorithm the number of (elementary) operations it performs, and as *cache complexity* the number of misses it incurs. Note that the work complexity is not defined in the EM model since an I/O is a very slow operation, and the execution time of an algorithm is generally determined by disk accesses.

There is a natural correspondence between I/Os in the EM model and cache misses in the IC model: a miss requires the fetching of a B-block from memory and the eviction of a B-block from cache if there is no empty line; hence, a miss corresponds to at most two I/Os, and for these reasons we will intentionally mix the two terms. Lower bounds on the cache complexity on $\mathsf{IC}(M, B)$ naturally translate into lower bounds on the I/O complexity on $\mathsf{EM}(M, B)$. Furthermore, an EM algorithm translates into an IC algorithm by removing I/O operations, and its cache-complexity is not bigger than the I/O complexity of the original algorithm, if the same values are used for corresponding parameters.

One of the most restrictive assumptions of the IC model is the optimal off-line replacement policy, however Frigo et al. proved that the cache complexity of a wide class of IC algorithms does not change asymptotically on an IC model with a *Last Recently Used* (*LRU*) replacement policy.

Corollary 2.2. ***[FLPR99, Corollary 13]*** *Consider an algorithm $\mathcal{A}$ with input size n whose cache complexity is $Q(n, M, B)$ on an $\mathsf{IC}(M, B)$. If $Q(n, M, B)$ satisfies the regularity condition*

$$Q(n, M, B) \in O\left(Q(n, 2M, B)\right), \tag{2.1}$$

then the cache complexity of $\mathcal{A}$ on an $\mathsf{IC}(M, B)$ with an LRU replacement policy is $\Theta\left(Q(n, M, B)\right)$.

Cache-oblivious algorithms

As defined in [FLPR99], a *cache-oblivious* (resp., *cache-aware*) algorithm is an IC algorithm whose specification is independent of (resp., dependent on) the cache parameters M and B. A *cache-optimal* (resp., *work-optimal*) cache-oblivious algorithm denotes a cache-oblivious algorithm which reaches the best cache (resp., work) complexity on an $\mathsf{IC}(M, B)$ for all values of M and B; if an algorithm is both cache and work-optimal, it is said *optimal*. A number of cache-oblivious algorithms proposed in literature are cache-optimal only under the *tall-cache assumption*, that is, when $M \geq B^2$.

In [FLPR99] it is also proved that optimal cache-oblivious algorithms exhibit, under certain circumstances, optimal performance on platforms with multi-level memory hierarchies.

Lemma 2.3. ***[FLPR99, Lemma 15]*** *A cache-optimal cache-oblivious algorithm whose cache complexity satisfies the regularity condition 2.1 incurs an optimal number of cache misses on each level of a multilevel hierarchy of IC caches with an LRU replacement policy.*

A number of optimal cache-oblivious algorithms [FLPR99, Dem02] and data structures [ABF05] have been proposed in literature for important problems. As we will see in details in Chapter 3, potentialities and intrinsic limitations of oblivious versus aware algorithms has been studied by Bilardi and Peserico [BP01] and by Brodal and Fagerberg [BF03].

Cache-oblivious algorithms are attractive especially in a global computing environment [FK03], which provides dependable and cost-effective access to a number of platforms of varying computational capabilities, irrespective of their physical locations or access points. Indeed, the actual platform(s) onto which an application is ultimately run, may not be known at the time when the application is designed.

As in the EM model, we describe a cache-oblivious algorithm for the n-MM problem.

Theorem 2.4. ***[FLPR99, Theorem 1]*** *There exists a cache-oblivious algorithm for the n-MM problem that requires optimal work*

$$W(n, M, B) \in \Theta\left(n^3\right),$$

and, under the tall-cache assumption, optimal cache complexity

$$Q(n, M, B) \in \Theta\left(\frac{n^3}{B\sqrt{M}} + \frac{n^2}{B} + 1\right). \tag{2.2}$$

Proof. This is a sketch of the original proof. Let A and B be the two $n \times n$ input matrices and let C be the output matrix, which we suppose to be initially set to zero. The algorithm divides each matrix into four $n/2 \times n/2$ submatrices, which we denote by $A_{i,j}$ and $B_{i,j}$, for $i, j \in \{0, 1\}$. Then, eight subproblems $C_{i,j} \leftarrow C_{i,j} + A_{i,k} \cdot B_{k,j}$ are computed recursively for all values of i, j and k, with $i, j, k \in \{0, 1\}$. The recursion stops when the subproblem size is a constant. It easy to see that the cache complexity is given by the following recurrence, where the base case is reached when the subproblem is entirely contained in cache ($\alpha \in (0, 1)$ is a suitable constant):

$$Q(n, M, B) \leq \begin{cases} 8Q\left(\frac{n}{2}, M, B\right) + O\left(\frac{n^2}{B}\right) & \text{if } n^2 > \alpha M \\ O\left(\frac{n^2}{B} + 1\right) & \text{if } n^2 \leq \alpha M \end{cases}$$

which yields Equation 2.2. Optimality follows from [HK81]. The correctness of the algorithm and its work complexity can be easily derived. □

Note that the cache-oblivious algorithm for n-MM exhibits the traditional divide and conquer structure of cache-oblivious algorithms: the problem is recursively decomposed into smaller subproblems, until the subproblem fits in cache. Since the cache size cannot be used in the algorithm, the recursion finishes when the subproblem has a constant size.

As we will see in Chapter 4, it is convenient to study the definition and analysis of cache-oblivious algorithms with reference to the following three distinct models: the *specification* model, which is a Random Access Machine; the *evaluation* model, which is an $\mathsf{IC}(M, B)$; the *execution machine* model which is a hierarchy of many ICs with an LRU replacement policy. A cache-oblivious algorithm is an algorithm defined in the specification model whose cache and work complexities are measured in the evaluation model (clearly, a so defined algorithm is independent of IC parameters) and which is carried out in the execution model.

2.2 Models for communication hierarchy

The definition of the amount of communication performed by an algorithm presents many obstacles: indeed, communication is defined only with respect to a specific mapping of a computation onto a specific machine structure. Furthermore, the impact of communication on performance depends on the latency and bandwidth properties of the channels connecting different parts of the target machine. In this scenario, algorithm design, optimization, and analysis can become highly machine dependent, which is undesirable from the economical perspective of developing efficient and portable software, and it is not clear whether a single model of computation can adequately capture the communication requirements of an algorithm for all machines. This problem has been widely acknowledged and a number of approaches have been proposed to tackle it.

We can organize the existing parallel models of computation in a spectrum and outline three classes. On one end of the spectrum, we have the *parallel slackness* approaches, based on the assumption that, if a sufficient amount of parallelism is provided by algorithms, then general and automatic latency-hiding techniques can be deployed to achieve an efficient execution. Broadly speaking, the required algorithmic parallelism must be at most proportional to the product of the number of processing units by the worst-case latency of the target machine [Val90]. Assuming that this amount of parallelism is typically available in computations of practical interest, algorithm design can dispense altogether with communication concerns and focus on the maximization of parallelism. The functional/data-flow [Arv81] and the *Parallel RAM* (*PRAM*) [KR90, FW78, Gol78] models of computation have often been supported with similar arguments. Unfortunately, as argued in [BP97, BP99], latency hiding is not a scalable technique, due to fundamental physical constraints. Hence, parallel slackness does not really solve the communication problem.

On the other end of the spectrum, we could place the *universality* approaches, whose objective is the development of machines (nearly) as efficient as any other machine of (nearly) the same cost (e.g., [Lei85, BBP99]). To the extent that a universal machine with very small performance and cost gaps could be identified, one could adopt a model of computation sufficiently descriptive of such a machine, and focus most of the algorithmic effort on this model. As technology approaches the inherent physical limitations to information processing, storage, and transfer, the emergence of a universal architecture becomes more likely. Economy of scale can also be a force favoring convergence in the space of commercial machines. While this appears as a perspective worthy of investigation, at this stage, neither the known theoretical results nor the trends of commercially available platforms indicate an imminent convergence.

In the middle of the spectrum, a variety of models proposed in the literature can be viewed as variants of an approach aiming at realizing an *efficiency/portability/design-complexity tradeoff*. Well-known examples of these models are the *Local PRAM* [ACS90], *Bulk Synchronous Parallel* (*BSP*) [Val90] and its refinements (e.g., [DlTK96, BPP07, BDP99]), *LogP* [CKP+96], *Queuing Shared Memory* (*QSM*) [GMR99], and several others. These models aim at capturing by means of parameters relevant features common to most (reasonable) machines, while ignoring less relevant features, specific of the various architectures. The hope is that performance of real machines are largely determined by the modeled features, so that optimal algorithms in the model translate into near optimal ones on real machines.

One of the most prominent models is the BSP model of Valiant [Val90]. It is composed of P *processing elements*, each one equipped with a CPU and a local memory, which communicate through a network. An algorithm is composed of a sequence of supersteps. During a superstep a processor executes operations on data residing in the local memory, sends/receives messages to/from other processors, and, at the end, performs a global synchronization instruction. A message sent during a superstep becomes available to the receiver only at the beginning of the next superstep. If during a superstep the maximum number of operations performed by a processor on local data is τ and each processor sends and receives at most h messages (*h-relation*), then the *superstep time* is defined to be $\tau + gh + l$, where parameters g and l account for the inverse of the bandwidth and for latency/synchronization costs, respectively.

The BSP provides an elegant and simple way to deal with some of the most important aspects of actual platforms: granularity of memory, non uniformity of memory-access time, communication and synchronization's costs. For these reasons the BSP has been regarded an optimal *bridging* model, that is, a model which combines the following contrastant properties: usability, effectiveness, portability. However, the BSP model makes a restrictive assumption on the interconnection network: specifically, the cost of an h-relation depends only on the value of h and not on the communication pattern. This assumption is not true in most interconnection networks (e.g., multidimensional arrays and fat tree) where it is cheaper to move information between near neighbors than distant nodes. The *Decomposable-BSP* (*D-BSP*) model [DlTK96, BPP07] has been motivated by this consideration.

2.2.1 D-BSP model

The Decomposable-BSP was introduced by [DlTK96] and successively studied in many papers (e.g., [BPP07, BPP99, BFPP01, FPP06, FPP03, Fan03]). It is an extension of the BSP aiming at accounting for submachine locality. The D-BSP described in this section is a variant of the one presented in the aforementioned papers because it features a minimum block size for message exchanges. (A similar feature was introduced in another extension of the BSP called BSP* [BDP99, BDadH98].)

The $\mathsf{D\text{-}BSP}(P, \mathbf{g}, \mathbf{B})$, where $\mathbf{g} = (g_0, g_1, \ldots g_{\log P-1})$ and $\mathbf{B} = (B_0, B_1, \ldots B_{\log P-1})$, consists of P *processing elements*, $P_0, \ldots, P_{P-1}$, each one equipped with a CPU and a local memory, which communicate through a network. Specifically, for $0 \leq i \leq \log P$, the P processors are partitioned into 2^i fixed and disjoint *i-clusters* $C_0^i, C_1^i \ldots C_{2^i-1}^i$ of $P/2^i$ processors each, where the processors of a cluster are capable of communicating among themeselves independently of the other clusters. The clusters form a hierarchical, binary decomposition tree of the D-BSP machine, in the sense that $C_j^{\log P} = \{P_j\}$, for $0 \leq j < P$, and $C_j^i = C_{2j}^{i+1} \cup C_{2j+1}^{i+1}$, for $0 \leq i < \log P$ and $0 \leq j < 2^i$.

A D-BSP program consists of a sequence of *labeled supersteps*. In an *i-superstep*, with $0 \leq i < \log P$, each processor executes internal computation on locally held data and sends/receives messages exclusively to/from processors within its i-cluster. The superstep ends with a barrier, which synchronizes processors independently within each i-cluster. It is assumed that each message contains a constant number of words, and that messages sent in one superstep become available at the destinations only at the beginning of the following superstep. Consider the execution of an i-superstep s and let $\tau^s \geq 0$ be the maximum number of operations (*computation time*) performed by a processor during the local computation, and let w_{jk}^s be the number of words P_j sends to P_k, with $0 \leq j, k < P$. Words exchanged between two processors in an i-superstep can be envisioned as traveling within *blocks* of fixed size B_i (in words). The *block-degree* h_s of the i-superstep s is

$$h_s = \max_{0 \leq j < P} \left\{ \max \left(\sum_{k=0}^{P-1} \lceil w_{jk}^s / B_i \rceil, \sum_{k=0}^{P-1} \lceil w_{kj}^s / B_i \rceil \right) \right\},$$

while its *communication time* is $g_i h_s$. The communication time (resp., computation time) of an algorithm is the sum of the communication times (resp., computation times) of its supersteps. Hence, the model rewards batched over fine-grained communications.

For simplicity, we take P and all of the B_i's to be powers of two. Also, we assume that both the B_i's and the ratios g_i/B_i are non increasing for $0 \leq i < \log P$. It is indeed reasonable that, in smaller submachines, smaller block sizes are used and blocks can be dispatched at a faster rate.

Previous versions of D-BSP [DlTK96, BPP99] do not model blocked communication but include a latency parameter vector $(l_0, l_1, \ldots l_{\log P-1})$, so that the communication time of an i-superstep s, in which each processor sends or receives at most h_s words, is $h_s g_i + l_i$. The introduction of blocks makes the model more descriptive of actual platforms and also compensates for the absence of the latency parameters. Furthermore, in the general formulation of the D-BSP model [DlTK96], processors can be aggregated into an arbitrary collection of clusters which can change dynamically: this feature makes it possible to incorporate the fine details of point-to-point network interconnections. However, the simple binary decomposition presented above (referred as *recursive* D-BSP in [DlTK96]) makes algorithm design easier while it effectively abstracts many prominent interconnections [BPP07].

As example, we describe a simple and optimal D-BSP algorithm for multiplying, using only semiring operations, two $n \times n$ matrices distributed according with the row-major layout among the processors[2] (i.e., the n-MM problem).

Theorem 2.5. *There exists an optimal D-BSP algorithm for the n-MM problem whose communication and computation times on D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$ are respectively:*

$$D(n, P, \boldsymbol{g}, \boldsymbol{B}) \in \Theta\left(\frac{n^2}{P}\sum_{i=0}^{\log P-1} 2^{i/2}\frac{g_i}{B_i}\right), \tag{2.3}$$

$$T(n, P, \boldsymbol{g}, \boldsymbol{B}) \in \Theta\left(\frac{n^3}{P}\right), \tag{2.4}$$

when $P \leq n^2$ and $B_i \in O\left(n^2/P\right)$ for each i with $0 \leq i < \log P$.

Proof. Let A and B be the two input matrices and let C be the output matrix, which we suppose to be initially set to zero. The D-BSP algorithm is similar to the cache-oblivious one described in Theorem 2.4: each matrix is divided into four $n/2 \times n/2$ submatrices, denoted by $A_{i,j}$ and $B_{i,j}$, with $0 \leq i, j \leq 1$; then eight subproblems $C_{i,j} \leftarrow A_{i,k} \cdot B_{k,j}$ are recursively solved in parallel in two rounds by the four 2-clusters. In order to keep memory requirements at minimum, subproblems are solved in the following order:

- First round: $A_{0,0} \cdot B_{0,0}$, $A_{0,1} \cdot B_{1,1}$, $A_{1,1} \cdot B_{1,0}$, $A_{1,0} \cdot B_{0,1}$;
- Second round: $A_{0,1} \cdot B_{1,0}$, $A_{0,0} \cdot B_{0,1}$, $A_{1,0} \cdot B_{0,0}$, $A_{1,1} \cdot B_{1,1}$.

In each round, each 2-cluster solves one subproblem. The recursion stops when the subproblem size is $n/\sqrt{P} \times n/\sqrt{P}$, that is, when the subproblem is solved by a $\log P$-cluster (a singleton processor); in this case a simple sequential algorithm is used.

The value for $D(n, P, \mathbf{g}, \mathbf{B})$ given in Equation 2.3 is obtained by solving the following recurrence (for notational convenience, we leave P, $\mathbf{g}$ and $\mathbf{B}$ out from

[2]An $n \times n$ matrix A is distributed according with the row-major layout among P processors $P_0, P_1, \ldots P_{P-1}$ as follows: A is divided into P submatrices $A_{i,j}$ of size $\lceil n/p \rceil \times \min\{n, n^2/p\}$, with $0 \leq i < n/\lceil n/p \rceil$ and $0 \leq j < n/\min\{n, n^2/p\}$; $A_{i,j}$ is contained in processor P_k, where $k = i \cdot n/\min\{n, n^2/p\} + j$.

$D(n,i)$):

$$D(n,i) \leq \begin{cases} 2D\left(\frac{n}{2}, i+2\right) + O\left(\frac{n^2 g_i}{PB_i}\right) & \text{if } i < \log P \\ 0 & \text{if } i \geq \log P \end{cases}$$

Hence, we get

$$D(n,i) \in O\left(\frac{n^2}{P} \sum_{j=0}^{\lceil (\log P - i)/2 \rceil - 1} 2^j \frac{g_{i+2j}}{B_{i+2j}}\right),$$

from which Equation 2.3 follows by setting $i = 0$. The optimality of the communication time is a consequence of Corollary 4.10, and we refer to it for more details. The correctness and the optimality of computation time can be easily derived. □

Chapter 3

Limits of Cache-Oblivious Rational Permutations

A likely impossibility is always preferable
to an unconvincing possibility.
(Aristotle)

A number of optimal cache-oblivious algorithms [FLPR99, Dem02] and data structures [ABF05] have been proposed in literature. However, in several cases, optimality is attained under the so-called *tall-cache assumption* which requires the cache size in words to be at least the square of the cache line size in words. Recently, Brodal and Fagerberg [BF03] have proved that a cache-oblivious algorithm for sorting cannot be optimal for every set of the values of the cache parameters. Moreover, they have shown that no cache-oblivious algorithm for permuting can exhibit optimal cache complexity for all values of the cache parameters, even under the tall-cache assumption. Impossibility results of a similar flavor have been proved by Bilardi and Peserico [BP01] in the context of DAG computations on the Hierarchical Memory model [AACS87], which does not account for the spatial locality of reference. This kind of results gives important theoretical insights on the inherent trade-off between efficiency and portability among different memory hierarchies of cache-oblivious algorithms, and on the exploitation of the *Translation Lookaside Buffer* (*TLB)*, as explained below.

The TLB is a memory used by platforms which support *virtual memory* [Rah02, HP06]. Virtual memory means that each program has an unique *logical address space* which is independent of the *physical address space* assigned to it at the execution time. Both the logical and physical address spaces are divided into contiguous *pages* of B words and, during the execution of a program, logical pages are randomly mapped by the platform to physical pages. When a program refers to a logical address x contained in the logical page y, the platform translates y into the physical page address where the datum is actually stored. The translation is done by looking up the *page table*, a data structure in main memory which contains the map between logical and physical pages. An access to this structure for each memory request would lead to an unacceptable slowdown: for this reason, the TLB is used to speed up address translation. A TLB is a fast associative memory which holds ℓ translations of recently-accessed logical pages (different replacement policies can be adopted): if an access to a logical address results in a translation contained in the buffer (*TLB hit*), there is no delay, otherwise a slow access to the page table (*TLB miss*) is required. It is easy to see that the rationale behind a TLB is the exploitation of the locality of reference, as in standard data caches. Furthermore, a TLB can be modelled as an Ideal Cache $\mathsf{IC}(B\ell, B)$ [Kum03], where a cache line contains all the words within the same virtual page, that is, by the B words that benefit from the same translation stored in the TLB. Programs are usually oblivious of TLB parameters, however a TLB does generally not satisfy the tall-cache assumption[1]: indeed, typical values of the parameters are $\ell \sim 128$ and $B \sim 4KB$. Hence, impossibility results on cache-obliviousness give also deep insights on TLB exploitation of algorithms (even if cache-aware).

In this chapter we focus the attention on lower and upper bounds for performing an important class of permutations in an oblivious fashion. Permuting a vector is a fundamental primitive for many problems and, in particular, the so-called *rational permutations* are widely used. A permutation is rational if it is defined by a permutation of the bits of the binary representations of the vector indices. Matrix transposition, bit-reversal and some permutations implemented in the Data Encryption Standard (DES) [FIP99] are notable examples of rational permutations. There are some works in literature which deal with the efficient implementation of specific rational permutations in a memory hierarchy: for example, Frigo et al. [FLPR99] propose a cache-oblivious algorithm for matrix transposition which is optimal under the tall-cache assumption; Carter and Kang [CG98] give an optimal cache-aware algorithm for the bit-reversal of a vector. To the best of our knowledge, the only works in literature which propose a general approach to rational permutations are [ACS87, ACFS94, Cor93b, Cor93a]. The first two papers propose efficient algorithms for performing any rational permutation in the blocked Hierarchical Memory

[1]More exactly, the unit of the IC parameters (B and M) is really the size of the data element of an algorithm (e.g., double precision floating-point numbers)[Fri08]. Hence a TLB can be modelled as an $\mathsf{IC}(B\ell/s, B/s)$, where s denotes the size in words of a data element. If s is big (e.g., when a data element consists of 3D vector fields of pressure and velocity, and some scalars such as temperature and voltage), a TLB (barely) satisfies the tall-cache assumption. On the other hand, if a data element requires few words, the TLB is generally not tall.

[ACS87] and in the Uniform Memory Hierarchy [ACFS94] models, respectively. In [Cor93b, Cor93a] a lower bound on the number of disk accesses and an optimal algorithm for performing rational permutations are given for the Parallel Disk model [VS94] (which is similar to the EM model of [AV88]).

In this chapter we first present a lower bound on the work needed to execute any family of rational permutations in the IC model with an optimal cache complexity. The result requires a technical lemma which employs an adaptation of the argument used in [AV88] to bound from below the number of disk accesses of an algorithm for matrix transposition in the EM model. Then, we propose a new algorithm for performing any rational permutation: this algorithm, differently from the one in [Cor93b], is cache-oblivious, and exhibits optimal cache and work complexities under the tall-cache assumption. Finally, we show that for certain families of rational permutations (including matrix transposition and bit-reversal) there is no cache-oblivious algorithm which achieves optimality for every set of the values of the cache parameters. To this purpose we follow a similar approach to the one employed in [BF03]. Specifically, let $\mathcal{A}$ be a cache-oblivious algorithm for a specific class of rational permutations and consider the two sequences of misses generated by its executions on two different ICs, where one model satisfies a the tall-cache assumption while the other does not. We simulate these two executions in the EM model and obtain a new EM algorithm solving the same problem as $\mathcal{A}$. By adapting the technical lemma given in the argument for bounding from below the work complexity, we conclude that $\mathcal{A}$ cannot be optimal in both ICs.

The chapter is organized as follows. In Section 3.1 we give a formal definition of rational permutation. In Section 3.2 we present the lower bound on the work complexity and the aforementioned technical lemma. In Section 3.3 we describe the cache-oblivious algorithm for performing rational permutations. In Section 3.4 we present the simulation technique and apply it to prove the limits of any cache-oblivious algorithm performing certain families of rational permutations. An overview of the EM and IC models has been given in Chapter 2.

The results presented in this chapter were published in a preliminary version in [Sil06] and in the final form in [Sil08].

3.1 Rational permutations

An *N-permutation* Π_N is a bijective function from and to the set $\{0, \ldots, N-1\}$. This paper focuses on *rational permutations*, defined as follows. Let $N = 2^n$ and denote with σ an n-permutation and with $(a_{i,n-1}, \ldots, a_{i,0})$ the binary representation of the value i, where $0 \leq i < N$ and $a_{i,0}$ denotes the least significant bit (LSB). The rational N-permutation Π_N^σ maps each value i, with $0 \leq i < N$ to the value whose binary representation is $(a_{i,\sigma(n-1)}, \ldots, a_{i,\sigma(0)})$. We refer to σ as the bit-permutation defining Π_N^σ, and denote with σ^{-1} its inverse. Note that the inverse of Π_N^σ is $\Pi_N^{(\sigma^{-1})}$. In the remaining part of this section we define some concepts related to rational permutations that are used later in the chapter.

Given an n-permutation σ and an index j, with $0 \leq j < n$, we define the following sets of bit positions which, in some sense, provide a measure of how far bits are permuted by σ:

- *j-outgoing set*: $\mathcal{OUT}(j, \sigma) = \{k : (k < j) \wedge (\sigma^{-1}(k) \geq j)\}$;
- *j-incoming set*: $\mathcal{IN}(j, \sigma) = \{k : (k \geq j) \wedge (\sigma^{-1}(k) < j)\}$.

For convenience, we impose $\mathcal{OUT}(j, \sigma) = \mathcal{IN}(j, \sigma) = \emptyset$ when $j \geq n$. We call a bit position in $\mathcal{OUT}(j, \sigma)$ (resp., $\mathcal{IN}(j, \sigma)$) *j-outgoing bit position* (resp., *j-incoming bit position*). Observe that $\sigma^{-1}(k)$ denotes the position where k is permuted, then $\mathcal{OUT}(j, \sigma)$ contains the positions of the j LSBs that are permuted by σ in positions bigger or equal than j, while $\mathcal{IN}(j, \sigma)$ contains the positions of the j most significant bits (MSBs) that are permuted in positions smaller than j. It is evident that, if $\mathcal{OUT}(j, \sigma)$ positions of the j LSBs are permuted into the $(n - j)$ MSBs, then $\mathcal{OUT}(j, \sigma)$ positions of the $(n - j)$ MSBs must be permuted into the j LSBs: hence $|\mathcal{OUT}(j, \sigma)| = |\mathcal{IN}(j, \sigma)|$. Thus, we define the *$j$-cardinality* $\zeta(j, \sigma)$ as the cardinality of $\mathcal{OUT}(j, \sigma)$ (or $\mathcal{IN}(j, \sigma)$ equivalently).

Let $V = V[0], V[1], \ldots, V[N-1]$ be a vector of N entries. An algorithm performs the N-permutation Π_N on V if it returns a vector U, which does not overlap with V, such that $U[i] = V[\Pi_N(i)]$ for each i, with $0 \leq i < N$. Note that $V[i]$ is permuted into $U[\Pi_N^{-1}(i)]$, where Π_N^{-1} is the inverse of Π_N. We suppose that a vector entry requires one machine word and a word suffices to represent the index of a vector entry. We also assume that the entries of any vector are stored in consecutive memory locations, sorted by indices, and that the first entry of a vector stored in the memory (resp., disk) of the $\mathsf{IC}(M, B)$ (resp., $\mathsf{EM}(M, B)$) model is aligned with a B-block.

Let Σ denote an infinite set of permutations which contains at most one[2] n-permutation for each $n \in \mathbb{N}$. An algorithm performs the rational permutations defined by Σ if, when given in input an n-permutation $\sigma \in \Sigma$ and a vector V of $N = 2^n$ entries, it performs Π_N^σ on V. For each $N = 2^n$ such that there exists an

[2]The results in this chapter can be extended to the case where Σ contains an arbitrary number of n-permutations for any value of $n \in \mathbb{N}$. However, for simplicity, we assume that there is at most one n-permutation in Σ for each value of $n \in \mathbb{N}$.

n-permutation $\sigma \in \Sigma$, we denote by $\zeta_\Sigma(j, N)$ the j-cardinality $\zeta(j, \sigma)$. The above definition allows us to use the asymptotic notation in the results given in the subsequent sections, when the asymptotics are made with respect to N, hence n.

Let us see some notable examples of rational permutations. Let V be a vector representing a $\sqrt{N} \times \sqrt{N}$ matrix stored in a row-major layout, with $N = 2^n$ and n even. Transposing the matrix stored in V is equivalent to performing the rational permutations defined by $\Sigma_T = \{\tau_n : \forall\, n > 1 \text{ and } n \text{ even}\}$, where

$$\tau_n(j) = \left(j + \frac{n}{2}\right) \bmod n. \tag{3.1}$$

Since the j-outgoing and j-incoming sets of τ_n are

$$\mathcal{OUT}(j, \tau_n) = \begin{cases} \emptyset & \text{if } j = 0 \\ \{0, \dots, j-1\} & \text{if } 0 < j \leq \frac{n}{2} \\ \{j - \frac{n}{2}, \dots \frac{n}{2} - 1\} & \text{if } \frac{n}{2} < j < n \end{cases},$$

$$\mathcal{IN}(j, \tau_n) = \begin{cases} \emptyset & \text{if } j = 0 \\ \{\frac{n}{2}, \dots, \frac{n}{2} + j - 1\} & \text{if } 0 < j \leq \frac{n}{2} \\ \{j, \dots n-1\} & \text{if } \frac{n}{2} < j < n \end{cases}, \tag{3.2}$$

the j-cardinality of Σ_T is

$$\zeta_{\Sigma_T}(j, 2^n) = \min\{j, n - j\}. \tag{3.3}$$

Another interesting example is provided by the bit-reversal of a vector, which arises from the Fast Fourier Transform [CLRS01]. The bit-reversal is equivalent to the rational permutations defined by $\Sigma_R = \{\rho_n, \forall\, n \geq 1\}$, where

$$\rho_n(j) = n - j - 1.$$

The j-outgoing and j-incoming sets of a bit-permutation ρ_n are

$$\mathcal{OUT}(j, \rho_n) = \begin{cases} \emptyset & \text{if } j = 0 \\ \{0, \dots, j-1\} & \text{if } 0 < j \leq \lfloor \frac{n}{2} \rfloor \\ \{0, \dots, n-j-1\} & \text{if } \lfloor \frac{n}{2} \rfloor < j < n \end{cases},$$

$$\mathcal{IN}(j,\rho_n) = \begin{cases} \emptyset & \text{if } j = 0 \\ \{n-j, \dots n-1\} & \text{if } 0 < j \leq \left\lfloor \frac{n}{2} \right\rfloor \\ \{j, \dots, n-1\} & \text{if } \left\lfloor \frac{n}{2} \right\rfloor < j < n \end{cases} .$$

Hence, the j-cardinality of Σ_R is

$$\zeta_{\Sigma_R}(j, 2^n) = \min\{j, n-j\}. \tag{3.4}$$

3.2 Lower bounds

In this section we derive the $\Omega\left(N + N\zeta_\Sigma(\log B, N)/\log(1+M/B)\right)$ lower bound on the work complexity of any algorithm which performs a given family Σ of rational permutations with optimal cache complexity. To this purpose, we define a potential function and prove a technical lemma that provides an upper bound on the increase of the potential due to a miss. This result is an adaptation of a technical result given in [AV88] for bounding from below the number of disk accesses of an algorithm for matrix transposition in the EM model. The lower bound on the cache complexity given in [Cor93b] can be proved as a corollary of this technical lemma. Finally, we prove that the cache-aware algorithm obtained by adapting the one given in [Cor93a] for the EM model exhibits optimal cache and work complexities when executed on an $\mathsf{IC}(M,B)$ for each M and B.

Let Σ be an infinite set of n-permutations defined as in Section 3.1, and consider an algorithm for $\mathsf{IC}(M,B)$ which is able to perform any rational N-permutation defined by Σ on a vector of $N = 2^n$ entries. We denote with $Q_\Sigma(N,M,B)$ the cache complexity of this algorithm, and with V and U the input and output vectors, respectively. (Distinct areas of the RAM in the $\mathsf{IC}(M,B)$ machine are reserved for storing vectors V and U.)

We remind that the output vector U consists of N/B B-blocks and the i-th B-block, with $0 \leq i < N/B$, contains entries $U[iB, \dots, (i+1)B-1]$. We define the i-th *target group*, with $0 \leq i < N/B$, to be the set of entries of V which will be mapped by the permutation into the i-th B-block of U, that is, into $U[iB, \dots, (i+1)B-1]$. We define the following convex function:

$$f(x) = \begin{cases} x \log x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases} . \tag{3.5}$$

Let γ be a B-block of the memory (if there is a copy of the block in cache, we refer to that copy). The *togetherness rating of* γ $(C_\gamma(q))$ and the *potential function*

$(POT(q))$ after q misses are defined as:

$$C_\gamma(q) = \sum_{i=0}^{N/B-1} f(x_{\gamma,i}), \qquad POT(q) = \sum_{\forall B\text{-block } \gamma} C_\gamma(q),$$

where $x_{\gamma,i}$ denotes the number of entries in γ belonging to the i-th target group just before the $(q+1)$-st miss[3]. As proved in [Cor93b], the values of the potential function at the beginning and at the end of the algorithm are given by the following equations:

$$POT(0) = N \log\left(\frac{B}{2^{\zeta_\Sigma(\log B, N)}}\right), \qquad POT(Q_\Sigma(N,M,B)) = N \log B. \tag{3.6}$$

Let $\Delta POT(q)$ denote the increase in potential due to the q-th miss, with $1 \leq q \leq Q_\Sigma(N,M,B)$, that is $\Delta POT(q) = POT(q) - POT(q-1)$. The following lemma provides an upper bound on $\Delta POT(q)$, that is, the maximum increase due to a rearrangement of the entries in cache after the q-th miss.

Lemma 3.1. *Let γ be the block fetched into the cache as a consequence of the q-th miss, and C be the set of at most $M/B-1$ blocks residing in cache with γ. Denote with W the number of entries that are in γ just before the q-th miss and are in a block in C when γ is evicted from the cache, or vice versa. Then,*[4]

$$\Delta POT(q) \leq B + W \log \frac{2eM}{W} \tag{3.7}$$

for each q, with $1 \leq q \leq Q_\Sigma(N,M,B)$.

Proof. If there is no empty cache line when γ is fetched, then a block is evicted from the cache, but this operation does not affect the potential function. Block γ exchanges entries with blocks in C without incurring any miss: then, at most M/B blocks can increase their togetherness ratings before the next miss. We consider only data exchanged between γ and blocks in C, while we ignore the increase in the potential function due to rearrangements between two blocks α and β in C since it was considered when β was fetched in cache (if we suppose that β was fetched after α). Indeed, data movements after a miss can be reorganized in two rounds as follows: in the first round, data are moved within blocks in C; in the second round, data are moved between γ and blocks in C. Since we are considering the maximum increase in the potential function after each miss, the growth of the potential during

[3]If there is no $(q+1)$-st miss, we consider the end of the algorithm.

[4]We denote with e the Napier's constant.

the first round has already been considered (otherwise, there would exist a better organization of data in cache which increases the potential). We use the following notation:

- $m_{\alpha,i}$: number of entries in block α belonging to the i-th target group just before the q-th miss, with $\alpha \in C \cup \{\gamma\}$ and $0 \leq i < N/B$;
- $w_{\alpha,\beta,i}$: number of entries belonging to the i-th target group which are in block α just before the q-th miss, and are in block β just before γ is evicted, with $\alpha, \beta \in C \cup \{\gamma\}$, $\alpha \neq \beta$ and and $0 \leq i < N/B$. (Actually, we are interested only in $w_{\gamma,\alpha,i}$ and $w_{\alpha,\gamma,i}$, with $\alpha \in C$.)
- $w_{\alpha,i} = w_{\alpha,\gamma,i} - w_{\gamma,\alpha,i}$, with $\alpha \in C$.

By the definition of potential function, the increase in potential is

$$\begin{aligned}\Delta POT(q) &= \sum_{\alpha \in C \cup \{\gamma\}} \left(C_\alpha(q) - C_\alpha(q-1)\right) \\ &= \sum_{i=0}^{N/B-1} \left(f\left(m_{\gamma,i} + \sum_{\alpha \in C} w_{\alpha,i}\right) - f\left(m_{\gamma,i}\right) + \sum_{\alpha \in C} f\left(m_{\alpha,i} - w_{\alpha,i}\right) - f\left(m_{\alpha,i}\right)\right).\end{aligned}$$

We partition the target groups into two sets P and R: the i-th target group belongs to P if and only if $\sum_{\alpha \in C} w_{\alpha,i} \geq 0$, while it belongs to R otherwise. Let:

$$W_P = \sum_{i \in P} \sum_{\alpha \in C} w_{\alpha,i} \qquad W_R = -\sum_{i \in R} \sum_{\alpha \in C} w_{\alpha,i}.$$

Note that $W_P + W_R \leq W$. We can assume without loss of generality that, for each $\alpha \in C$, $w_{\alpha,i} \geq 0$ if $i \in P$ and $w_{\alpha,i} \leq 0$ if $i \in R$. Indeed, since $\sum_{\alpha \in C} w_{\alpha,i} \geq 0$, entries in C can be reorganized in such a way that each $w_{\alpha,i}$ is non negative. Similarly, we can assume $w_{\alpha,i} \leq 0$ if $i \in R$. Observe that the $\Delta POT(q)$ computed in such a way is not smaller that the one yielded in the general case.

The $m_{\alpha,i}$ values are limited by the constraints below:

$$\sum_{i \in P} m_{\gamma,i} \leq B - W_P, \quad \sum_{\alpha \in C} \sum_{i \in R} m_{\alpha,i} \leq M - W_R. \tag{3.8}$$

Then, by defining $\Delta POT_{\mathcal{T}}(q)$, with $\mathcal{T} \in \{P, R\}$, as

$$\Delta POT_{\mathcal{T}}(q) = \sum_{i \in \mathcal{T}} \left(f\left(m_{\gamma,i} + \sum_{\alpha \in C} w_{\alpha,i}\right) - f\left(m_{\gamma,i}\right) + \sum_{\alpha \in C} f\left(m_{\alpha,i} - w_{\alpha,i}\right) - f\left(m_{\alpha,i}\right)\right)$$

we have that $\Delta POT(q) = \Delta POT_P(q) + \Delta POT_R(q)$.

When $i \in P$, we have $w_{\alpha,i} \geq 0$ and, by Lemma A.1, $f\left(m_{\alpha,i} - w_{\alpha,i}\right) - f\left(m_{\alpha,i}\right) \leq -f\left(w_{\alpha,i}\right)$. Then,

$$\Delta POT_P(q) \leq \sum_{i \in P} \left[f\left(m_{\gamma,i} + \sum_{\alpha \in C} w_{\alpha,i} \right) - f\left(m_{\gamma,i}\right) - \sum_{\alpha \in C} f\left(w_{\alpha,i}\right) \right].$$

By Inequalities 3.8 and Corollary A.4, an upper bound on $\Delta POT_P(q)$ is obtained by plugging $m_{\gamma,i} = (B - W_P)/|P|$ and $w_{\alpha,i} = W_P/(|P||C|)$ into the previous equation:

$$\begin{aligned} \Delta POT_P(q) &\leq B \log \frac{B}{|P|} - (B - W_P) \log \frac{B - W_P}{|P|} - W_P \log \frac{W_P}{|P||C|} \\ &\leq (B - W_P) \log \frac{B}{B - W_P} + W_P \log \frac{B|C|}{W_P} \\ &\leq B + W_P \log \frac{M}{W_P}, \end{aligned} \tag{3.9}$$

since $|C| < M/B$.

When $i \in R$, we have $w_{\alpha,i} \leq 0$ and, by Lemma A.1, $f\left(m_{\gamma,i} + \sum_{\alpha \in C} w_{\alpha,i}\right) - f\left(m_{\gamma,i}\right) \leq -\sum_{\alpha \in C} f\left(-w_{\alpha,i}\right)$. Therefore, we have:

$$\Delta POT_R(q) \leq \sum_{i \in R} \sum_{\alpha \in C} \left[f\left(m_{\alpha,i} - w_{\alpha,i}\right) - f\left(m_{\alpha,i}\right) - f\left(-w_{\alpha,i}\right) \right].$$

By Lemma A.3 an upper bound on $\Delta POT_R(q)$ is obtained by setting $m_{\alpha,i} = (M - W_R)/(|R||C|)$ and $w_{\alpha,i} = -W_R/(|R||C|)$. Then:

$$\begin{aligned} \Delta POT_R(q) &\leq M \log \frac{M}{|R||C|} - (M - W_R) \log \frac{M - W_R}{|R||C|} - W_R \log \frac{W_R}{|R||C|} \\ &\leq (M - W_R) \log \left(1 + \frac{W_R}{M - W_R}\right) + W_R \log \frac{M}{W_R} \\ &\leq W_R \log \frac{eM}{W_R}, \end{aligned} \tag{3.10}$$

since $(1 + 1/x)^x \leq e$ if $x \geq 1$.

By Equations 3.10 and 3.9, and the fact that $W_R + W_P \leq W$, we derive the following upper bound:

$$\Delta POT(q) \leq B + W_P \log \frac{M}{W_P} + W_R \log \frac{eM}{W_R} \leq B + W \log \frac{2eM}{W}.$$

$\square$

Corollary 3.2. *The increase in the potential function due to the q-th miss, with*

$1 \leq q \leq N/B$, *is upper bounded by* $2B \log \frac{2eM}{B}$.

Proof. When a block γ is fetched into the cache, at most $2B$ entries are exchanged between γ and the other blocks residing in cache. The corollary follows from Lemma 3.1 by setting $W = 2B$. □

The following theorem restats the lower bound proved in [Cor93b, Theorem 2.23].

Theorem 3.3. *Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$. An algorithm which performs any rational N-permutation Π_N^σ defined by an n-permutation $\sigma \in \Sigma$ requires*

$$\Omega\left(\frac{N\zeta_\Sigma(\log B, N)}{B\log(1+M/B)} + \frac{N}{B} + 1\right) \tag{3.11}$$

misses on an IC(M, B), for each value of M and B. [5]

Proof. The $\Omega\,(N/B + 1)$ lower bound is straightforward because the input and output vectors are distinct. The first term of Equation 3.11 follows from Corollary 3.2 and Equations 3.6 since

$$\sum_{q=1}^{Q_\Sigma(N,M,B)} \Delta POT(q) \geq POT(Q_\Sigma(N,M,B)) - POT(0)$$
$$2B\log\frac{2eM}{B} Q_\Sigma(N,M,B) \geq N\zeta_\Sigma(\log B, N),$$

which yields

$$Q_\Sigma(N,M,B) \geq \frac{N\zeta_\Sigma(\log B, N)}{2B\log\frac{2eM}{B}}.$$

□

An obvious lower bound on the work complexity of an algorithm performing rational permutations is $\Omega\,(N)$ since N entries have to be moved from the input vector V to the output vector U. Moreover, this work complexity is attained by the naïve algorithm which reads V sequentially and moves each entry $V[i]$ directly into $U[\Pi_N^{\sigma^{-1}}(i)]$ (we remind that $\Pi_N^{\sigma^{-1}}$ is the inverse of Π_N^σ), but this algorithm is not cache-optimal. We wonder whether there is a cache-optimal IC algorithm whose

[5]Note that the lower bound given in [Cor93b] is $\Omega\left(\frac{N\max\{\zeta_\Sigma(\log M,N),\zeta_\Sigma(\log B,N)\}}{B\log(M/B)} + \frac{N}{B} + 1\right)$, but it easy to see that it is asymptotically equivalent to Equation 3.11.

work complexity is $\Theta(N)$ for each value of IC parameters. The following theorem states that such an algorithm cannot exist.

Theorem 3.4. *Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$. Consider an algorithm which performs any rational N-permutation Π_N^σ defined by an n-permutation $\sigma \in \Sigma$ and whose cache complexity is*

$$Q_\Sigma(N, M, B) \in \Theta\left(\frac{N\zeta_\Sigma(\log B, N)}{B\log(1 + M/B)} + \frac{N}{B} + 1\right),$$

on an IC(M, B), for each value of M and B. Then its work complexity is

$$W_\Sigma(N, M, B) \in \Omega\left(\frac{N\zeta_\Sigma(\log B, N)}{\log(1 + M/B)} + N\right) \tag{3.12}$$

when $M/B > b$, for a suitable constant $b > 1$.

Proof. If $\zeta_\Sigma(\log B, N) \leq \log(1 + M/B)$, Equation 3.12 becomes the $\Omega(N)$ lower bound. Suppose $\zeta_\Sigma(\log B, N) > \log(1 + M/B)$, and let c and d be two suitable constants such that

$$c\frac{N\zeta_\Sigma(\log B, N)}{B\log(1 + M/B)} \leq Q_\Sigma(N, M, B) \leq d\frac{N\zeta_\Sigma(\log B, N)}{B\log(1 + M/B)}. \tag{3.13}$$

Denote with $Q'_\Sigma(N, M, B)$ the number of misses each of which increases the potential by at least $(B/2d)\log(1 + M/B)$. We claim that $Q'_\Sigma(N, M, B) = \Theta(Q_\Sigma(N, M, B))$. Let ΔPOT be the upper bound given in Corollary 3.2 on the increase in the potential function due to a miss, and let $\Delta POT_1 = (B/2d)\log(1 + M/B)$. Then,

$$\begin{aligned} POT(Q) - POT(0) \leq & \\ &\leq (Q_\Sigma(N, M, B) - Q'_\Sigma(N, M, B))\,\Delta POT_1 + Q'_\Sigma(N, M, B)\Delta POT \\ &\leq Q_\Sigma(N, M, B)\Delta POT_1 + Q'_\Sigma(N, M, B)\Delta POT \end{aligned}$$

From Equations 3.6 and Inequality 3.13, we derive

$$Q'_\Sigma(N, M, B)\left(2B\log\left(\frac{2eM}{B}\right)\right) \geq N\zeta_\Sigma(\log B, N) - \frac{dN\zeta_\Sigma(\log B, N)}{2d},$$

which implies

$$Q'_\Sigma(N, M, B) \in \Omega\left(\frac{N\zeta_\Sigma(\log B, N)}{B\log(1 + M/B)}\right).$$

Let q be a miss which increases the potential by at least ΔPOT_1, and let γ be the block fetched into the cache in the q-th miss. By Lemma 3.1, if at most W entries are exchanged between γ and the other blocks resident in cache with γ, the potential increases by at most $(B+W\log(2eM/W))$. If $M/B \geq b$ for a suitable constant $b > 1$ and $W < B/4d$, then $(B + W\log(2eM/W)) < \Delta POT_1$, which is a contradiction. Then $W = \Theta(B)$. Since an IC operation moves only a constant number of words between blocks, there are at least $\Omega(B)$ operations per miss. The theorem follows. □

Corollary 3.5. *Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$. Each rational N-permutation Π_N^σ, defined by an n-permutation $\sigma \in \Sigma$, can be performed by an optimal cache-aware algorithm with work complexity*

$$W(N, M, B) \in \Theta\left(\frac{N\zeta(\log B, \sigma)}{\log(1 + M/B)} + N\right),$$

and cache complexity

$$Q(N, M, B) \in \Theta\left(\frac{N\zeta(\log B, \sigma)}{B\log(1 + M/B)} + \frac{N}{B} + 1\right),$$

on an $\mathsf{IC}(M, B)$, for each value of M and B.

Proof. An optimal algorithm for performing a rational permutation Π_N^σ in the Parallel Disk model [VS94] with p disks is given in [Cor93a]. By setting $p = 1$, this algorithm translates automatically into an EM algorithm; then, by removing all I/O operations, the algorithm becomes a cache-aware IC algorithm. Clearly, the number of block transfers performed by the optimal off-line policy of the IC model cannot be bigger than the number of disk accesses performed in the Parallel Disk model with $p = 1$. This cache-aware algorithm is composed of $\zeta(\log B, \sigma)/(B\log(1 + M/B))$ phases, each of which requires $\Theta(N)$ work and $\Theta(N/B)$ misses. By Theorems 3.3 and 3.4, this algorithm exhibits optimal cache and work complexities on an $\mathsf{IC}(M, B)$, for each value of M and B. □

Corollary 3.6. *Matrix transposition and bit-reversal can be performed by an optimal cache-aware algorithm with work complexity*

$$W(N, M, B) \in \Theta\left(\frac{N\log\min\{B, N/B\}}{\log(1 + M/B)} + N\right),$$

and cache complexity[6]

$$Q(N, M, B) \in \Theta\left(\frac{N \log \min\{B, N/B\}}{B \log(1 + M/B)} + \frac{N}{B} + 1\right), \tag{3.14}$$

on an $\mathsf{IC}(M, B)$*, for each value of* M *and* B*.*

Proof. The statement follows by Corollary 3.5 and Equations 3.3 and 3.4. □

3.3 Cache-oblivious algorithm for rational permutations

In this section we propose an optimal cache-oblivious algorithm which performs each rational permutation Π_N^σ on a vector V of $N = 2^n$ entries, where σ denotes an n-permutation. It incurs $\Theta(N/B)$ misses and requires $\Theta(N)$ work on an $\mathsf{IC}(M, B)$ that satisfies the tall-cache assumption. We also describe an efficient cache-oblivious algorithm for computing all the values $\Pi_N^\sigma(i)$, with $0 \leq i < N$, since the computation of any such value cannot be considered as an elementary operation.

3.3.1 Computing the values of a rational permutation

Let $N = 2^n$ and let Π_N^σ be the rational permutation defined by the n-permutation σ. In this subsection we describe an algorithm which computes a vector P of N entries, where $P[i] = \Pi_N^\sigma(i)$ for each i, with $0 \leq i < N$. The algorithm derives $\Pi_N^\sigma(i)$ from $\Pi_N^\sigma(i-1)$ (note that $\Pi_N^\sigma(0) = 0$ for each σ), comparing the binary representations of i and $i - 1$.

Specifically, the algorithm uses four vectors:

- S where $S[j] = \sigma(j)$ for each j, with $0 \leq j < n$;
- I where $I[j] = \sigma^{-1}(j)$ for each j, with $0 \leq j < n$;
- P where, at the end of the algorithm, $P[i] = \Pi_N^\sigma(i)$ for each i, with $0 \leq i < N$;
- A where $A[j]$ stores the j-th bit of the binary representation of the current index i, with $0 \leq j < n$ and $0 \leq i < N$ ($A[0]$ is the LSB).

[6] In [AV88] a lower bound on the I/O complexity of matrix transposition in the EM model is provided, which is $\Omega\left(\frac{N \log \min\{M, \sqrt{N}, N/B\}}{B \log(1+M/B)} + \frac{N}{B} + 1\right)$. This is also a valid lower bound on the cache complexity of matrix transposition in the IC model, and coincides with Equation 3.14.

```
INPUT: a vector $S$ of $n$ entries which represents the $n$-permutation $\sigma$;
OUTPUT: a vector $P$ of $N$ entries, with $N = 2^n$, where $P[i] = \Pi_N^\sigma(i)$ for each
    $i$, with $0 \leq i < N$;
 1: Compute $I$ from $S$ through Mergesort;
 2: Set all entries of $A$ to 0;
 3: $P[0] \leftarrow 0$;
 4: $N \leftarrow 2^n$;
 5: for $i = 1$ to $N - 1$ do
 6:     $P[i] \leftarrow P[i-1]$;
 7:     $j \leftarrow 0$;
 8:     while $A[j] = 1$ do
 9:         $A[j] \leftarrow 0$; // The $j$-th bit of $i$ is set to 0
10:         $P[i] \leftarrow P[i] - 2^{I[j]}$; // The $I[j]$-th bit of $P[i]$ is set to 0
11:         $j \leftarrow j + 1$;
12:     $A[j] \leftarrow 1$; // The $j$-th bit of $i$ is set to 1
13:     $P[i] \leftarrow P[i] + 2^{I[j]}$; // The $I[j]$-th bit of $P[i]$ is set to 1
```

Figure 3.1: Cache-oblivious algorithm for computing the values of a bit-permutation σ.

More succinct data structures can be adopted, but we maintain the ones above for the sake of simplicity. The input of the algorithm is S (i.e., the bit-permutation σ), while the output is P. Note that I can be computed from S through sorting. The algorithm for computing P is divided into $N-1$ stages: in the i-th stage, with $0 < i < N$, the algorithm adds 1 (modulo N) to the binary representation of $i-1$ stored in A, and derives $P[i]$ from $P[i-1]$ according to the differences between the binary representations of i and $i-1$. The pseudocode for the algorithm is reported in Figure 3.1. Note that the algorithm is cache-oblivious and based on the binary counter [CLRS01].

Theorem 3.7. *The work and cache complexities of the algorithm in Figure 3.1 are:*

$$W(N, M, B) \in \Theta(N), \tag{3.15}$$

$$Q(N, M, B) \in \Theta\left(\frac{N}{B}\right) \tag{3.16}$$

on an IC(M, B), for each value of M and B such that $M/B \geq 4$.

Proof. The vector I can be efficiently computed through Mergesort with work complexity $o(N)$ and cache complexity $o(N/B)$ [BF03].

In order to bound the cache complexity of the **for** loop (Steps 5-13), we describe a particular replacement policy for the cache and compute the cache complexity using this policy: since the IC model adopts an optimal off-line replacement policy, the actual cache complexity cannot be larger than the one achieved by the policy we describe. We suppose the cache to have at least four lines, and we associate the vectors I, P and A with three distinct cache lines: that is, there is exactly one cache line for all the constituent blocks of each vector. The fourth line is used for support variables. Since the entries of P are required in sequential order, each constituent block of P is fetched only once into the line associated with P. Therefore, the number of misses due to P is $\Theta(N/B)$.

Let α_i be the memory block which contains entries $A[iB], \dots, A[(i+1)B-1]$, with $0 \le i < \lceil n/B \rceil$. When an entry $A[iB+k]$, with $0 \le i < \lceil n/B \rceil$ and $1 \le k < B$, is required, the corresponding block α_i is in cache since the previous required A's entry was $A[iB+k-1]$, which also belongs to α_i. On the other hand, when $A[iB]$ is referenced, block α_i is not in cache and a miss occurs. Since $A[j]$ flips $N/2^j$ times, with $0 \le j < n$, during the course of the algorithm [CLRS01], each block α_i is fetched into the cache $N/2^{iB}$ times. Therefore, the number of misses due to A is $\Theta(N/2^B)$. Since $I[j]$ is read only after $A[j]$ for each j, with $0 \le j < n$, the upper bound for A is also valid for I. Then, the number of misses due to I is $\Theta(N/2^B)$. Equation 3.16 follows. Since there are $\Theta(B)$ operations for each block, Equation 3.15 follows as well. □

3.3.2 Performing a rational permutation

In this subsection we present a cache-oblivious algorithm which performs any rational permutation Π_N^σ on a vector V of $N = 2^n$ entries, where σ and V are given as input. As usual, U denotes the output vector.

Before describing the algorithm, note that the recursive cache-oblivious algorithm for matrix transposition described in [FLPR99] moves each entry of the input matrix to the corresponding entry of the output matrix in an order based on the Z-Morton layout [CLPT99]. This particular *access pattern* to V minimizes the cache complexity of the algorithm under the tall-cache assumption. In the same fashion, our algorithm first derives an efficient access pattern for V from σ, and then it moves each entry of V, in the order specified by the pattern, into the right entry of U. The access pattern to V is defined by the n-permutation π_σ: the i-th accessed element is $V[j]$ where $j = \Pi_N^{\pi_\sigma}(i)$ for each i, with $0 \le i < N$. The n-permutation π_σ is computed by the algorithm in Figure 3.2.[7]

The algorithm for performing the rational permutation Π_N^σ on V is divided into N steps: in the i-th step, the entry $V[\Pi_N^{\pi_\sigma}(i)]$ is moved into $U[\Pi_N^{(\sigma^{-1})}(\Pi_N^{\pi_\sigma}(i))]$, with $0 \le i < N$. The pseudocode of the algorithm is given in Figure 3.3.

[7]For simplifying algorithms in Figures 3.2 and 3.3, we consider π_σ, π_σ^{-1}, σ, σ^{-1} as vectors.

```
INPUT: an $n$-permutation $\sigma$;
OUTPUT: the $n$-permutation $\pi_\sigma$;
1: Compute $\sigma^{-1}$ from $\sigma$ through Mergesort;
2: $i = 0$; $j = 0$;
3: while $j < n$ do
4:     if $\sigma^{-1}(i) \geq i$ then $\{\pi_\sigma^{-1}(j) = i;\ j = j + 1;\}$
5:     if $\sigma(i) > i$ then $\{\pi_\sigma^{-1}(j) = \sigma(i);\ j = j + 1;\}$
6:     $i = i + 1$;
7: Compute $\pi_\sigma$ from $\pi_\sigma^{-1}$ through Mergesort;
```

Figure 3.2: Cache-oblivious algorithm for computing the values of a bit-permutation π_σ.

```
INPUT: an $n$-permutation $\sigma$, and a vector $V$ of $N = 2^n$ entries;
OUTPUT: a vector $U$ of $N$ entries, where $U[\Pi_N^{\sigma^{-1}}(i)] = V[i]$ for each $i$, with
    $0 \leq i < N$;
1: Compute $\sigma^{-1}$ from $\sigma$ through Mergesort;
2: Compute $\pi_\sigma$ through algorithm in Figure 3.2;
3: Compute the values of the bit-permutations $\Pi_N^{\pi_\sigma}$ and $\Pi_N^{(\sigma^{-1})}$ through algorithm
   in Figure 3.1;
4: for $i = 0$ to $N - 1$ do
5:     $U[\Pi_N^{(\sigma^{-1})}(\Pi_N^{\pi_\sigma}(i))] = V[\Pi_N^{\pi_\sigma}(i)]$;
```

Figure 3.3: Cache-oblivious algorithm for performing a rational permutation Π_N^σ.

Let k be an arbitrary power of two, with $0 \leq k \leq N$, and let l be an arbitrary value with $0 \leq l < N/k$. Observe that, intuitively, the access pattern defined by π_σ guarantees that the k entries of V and U which are read and written during the $kl, \ldots, k(l+1) - 1$-st accesses, differ in at most $(\log k)/2$ of the $n - (\log k)/2$ MSBs. This implies that $\Theta\left(\sqrt{k}\right)$ cache lines of size $\Theta\left(\sqrt{k}\right)$ suffice to permute the k entries incurring $O\left(k/B + \sqrt{k}\right)$ misses. An example of π_σ is given later in Equation 3.17.

In order to prove the correctness and to evaluate the cache and work complexities of the algorithm in Figure 3.3, we introduce the following two lemmas. The first one shows a property of π_σ and that the algorithm in Figure 3.2 is well defined, while the second lemma proves that π_σ is actually an n-permutation.

Lemma 3.8. *Let σ be an n-permutation, $\pi_\sigma^{-1}(k)$ be the function computed in the algorithm in Figure 3.2, and let $\varrho_{i,\sigma} = \{\pi_\sigma^{-1}(k) : 0 \leq k \leq i + \zeta(i+1, \sigma)\}$. The algorithm in Figure 3.2 is well defined and we have*

$$\varrho_{i,\sigma} = \{0, \ldots, i\} \cup \mathcal{IN}(i+1, \sigma).$$

Proof. In order to prove the lemma, we show by induction on i, with $0 \leq i < n - 1$, that at the end of the i-th iteration of the algorithm in Figure 3.2, we have $j =$

$i + \zeta(i+1, \sigma) + 1$ (which also implies that $\varrho_{i,\sigma}$ is well defined at the end of the i-th iteration) and $\varrho_{i,\sigma} = \{0, \ldots, i\} \cup \mathcal{IN}(i+1, \sigma)$. If $i = 0$ the claim is clearly true. Let $i > 0$. Denote with $\tilde{j}$ the value of j at the beginning of the i-th iteration, that is $\tilde{j} = (i-1) + \zeta(i, \sigma) + 1$ by the inductive hypothesis. If i is assigned to $\pi_\sigma^{-1}(\tilde{j})$ in Step 4, then $i \notin \varrho_{i-1,\sigma}$: otherwise $i \in \varrho_{i-1,\sigma}$ and, in particular, $i \in \mathcal{IN}(i, \sigma)$. If $\sigma(i)$ is assigned to $\pi_\sigma^{-1}(\tilde{j})$ or $\pi_\sigma^{-1}(\tilde{j}+1)$ in Step 5, then either $\sigma(i) \in \mathcal{IN}(i+1, \sigma) - \mathcal{IN}(i, \sigma)$, or $\sigma(i) \in \mathcal{IN}(i, \sigma)$. A simple case analysis shows that at the end of the i-th iteration $j = i + \zeta(i+1, \sigma) + 1$ and $\varrho_{i,\sigma} = \{0, \ldots, i\} \cup \mathcal{IN}(i+1, \sigma)$.

Since $i < n$ in each iteration, $\sigma(i)$ and $\sigma^{-1}(i)$ in Lines 4 and 5 do exist and, therefore, the algorithm is well defined. □

Lemma 3.9. *Let σ be an n-permutation. Then the function π_σ defined by the algorithm in Figure 3.2 is an n-permutation.*

Proof. We claim that π_σ^{-1} (hence π_σ) is a permutation. Suppose, for the sake of contradiction, that there are two values j' and j'', $0 \leq j' < j'' < n$ such that $\pi_\sigma^{-1}(j') = \pi_\sigma^{-1}(j'') = p$. Clearly, p cannot be assigned to both $\pi_\sigma^{-1}(j')$ and $\pi_\sigma^{-1}(j'')$ by two steps of the same kind. Then, suppose that p is assigned to $\pi_\sigma^{-1}(j')$ in Step 4 and to $\pi_\sigma^{-1}(j'')$ in Step 5: by the **if** statements in Steps 4 and 5, it follows that $\sigma^{-1}(p) \geq p$ and $p > \sigma^{-1}(p)$, respectively, but this is a contradiction. In the same fashion, it can be proved that p cannot be assigned to $\pi_\sigma^{-1}(j')$ in Step 5 and to $\pi_\sigma^{-1}(j'')$ in Step 4. Therefore, π_σ^{-1} is a permutation since there are n values and no duplicates. □

As example, consider the bit-permutation τ_n associated with the transposition of a $2^{n/2} \times 2^{n/2}$ matrix (Equation 3.1). Then $\pi_{\tau_n}^{-1}$ is so defined:

$$\pi_{\tau_n}^{-1}(i) = \begin{cases} \frac{i}{2} & \text{if } i \text{ even and } 0 \leq i < n \\ \\ \frac{n}{2} + \frac{i-1}{2} & \text{if } i \text{ odd and } 0 \leq i < n \end{cases} . \tag{3.17}$$

The access pattern defined by $\pi_{\tau_n}^{-1}$ coincides with the Z-Morton layout, that is, with the access pattern used in the cache-oblivious algorithm for matrix transposition given in [FLPR99]. According with Lemma 3.8 and Equation 3.2, it is easy to see that:

$$\varrho_{i,\tau_n} = \begin{cases} \{0, \ldots, i\} \cup \{\frac{n}{2} \ldots \frac{n}{2} + i\} & \text{if } 0 \leq i < \frac{n}{2} \\ \\ \{0, \ldots, n-1\} & \text{if } \frac{n}{2} \leq i < n-1 \end{cases} .$$

Theorem 3.10. *Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$. The cache-oblivious algorithm in Figure 3.3 performs each rational permutation Π_N^σ defined by an n-permutation $\sigma \in \Sigma$, and requires*

$$W(N, M, B) \in \Theta(N) \tag{3.18}$$

work and

$$Q(N, M, B) \in \begin{cases} O\left(\dfrac{N}{B}\right) & \text{if } \frac{M}{B} \geq 2^{1+\zeta(\log B, \sigma)} \\ \\ O\left(\dfrac{NB}{M}\right) & \text{if } \frac{M}{B} < 2^{1+\zeta(\log B, \sigma)} \end{cases} \tag{3.19}$$

misses on an $\mathit{IC}(M, B)$, for each value of M and B such that $M/B > 4$.

Proof. The correctness of the algorithm in Figure 3.3 follows from the fact that π_σ and $\Pi_N^{\pi_\sigma}$ are permutations.

We now analyze the work and cache complexities of the algorithm in Figure 3.3. Recall that $\zeta(\log B, \sigma)$ is the cardinality of $\mathcal{OUT}(j, \sigma)$ (or $\mathcal{IN}(j, \sigma)$ equivalently). For simplifying the notation, we denote $\zeta(\log B, \sigma)$ with ζ.

We now show that Steps 1, 2 and 3 require $\Theta(N)$ work and $\Theta(N/B)$ misses. As argued in the proof of Theorem 3.7, the computation of the inverse of an n-permutation requires $o(N)$ work and $o(N/B)$ misses. Thus, the computation of σ^{-1} and of the algorithm in Figure 3.2 (Steps 1-2) can be performed in $o(N)$ operations and $o(N/B)$ misses. The computation of the values of $\Pi_N^{\pi_\sigma}$ and $\Pi_N^{(\sigma^{-1})}$ (Step 3) requires linear work and $\Theta(N/B)$ misses (by Theorem 3.7).

We now upper bound the cache complexity of Steps 4-5, in which all the entries of V are permuted into U. We distinguish between two cases: when the number M/B of cache lines is bigger or smaller than $2^{1+\zeta}$. In both cases we analyze the number of distinct blocks touched by a consecutive sequence (whose length is different in each case) of accesses to V and U in the order defined by π_σ. We show that, in the first case, $\Theta\left(2^\zeta\right)$ blocks of V and U are touched by $B2^\zeta$ consecutive accesses while, in the second case, $O(M/B)$ blocks are touched by $2^\varphi M/(2B)$ accesses (φ is a suitable value defined later).

Suppose $\frac{M}{B} \geq 2^{1+\zeta}$ and partition the sequence of N accesses to V into $N/(B2^\zeta)$ segments. Let $\mathcal{F}^i = \{\Pi_N^{\pi_\sigma}(iB2^\zeta), \ldots, \Pi_N^{\pi_\sigma}((i+1)B2^\zeta - 1)\}$, with $0 \leq i < N/B2^\zeta$, be the set of the indices of the entries accessed in the i-th segment. By Lemma 3.8, the binary representations of the values in $\mathcal{F}^i$ differ on $(\log B + \zeta)$ bit positions, and ζ of these are the $(\log B)$-incoming bit positions of σ, which are among the $\log(N/B)$

MSBs by definition. Then, the $B2^{\zeta}$ entries of V with indices in $\mathcal{F}^i$ are distributed among 2^{ζ} blocks. Moreover, in the $(\log B + \zeta)$ bit positions there are also ζ $(\log B)$-outgoing bit positions of σ. Then, by the definition of outgoing bit position, the $B2^{\zeta}$ entries are permuted into 2^{ζ} blocks of the output vector U. Since there are at least $2^{1+\zeta}$ cache lines, the permutation of entries indexed by the values in $\mathcal{F}_i$ requires $\Theta\left(2^{\zeta}\right)$ misses, and the permutation of the whole vector V requires $\Theta\left(N/B\right)$ misses.

Suppose $\frac{M}{B} < 2^{1+\zeta}$. Let φ be the maximum integer in $[0, \log B)$ such that $|\mathcal{OUT}(\log B, \sigma) \cap \mathcal{OUT}(\varphi, \sigma)| = \log(M/2B)$, that is φ denotes the bigger bit position such that exactly $\log(M/2B)$ $(\log B)$-incoming bit positions are permuted into positions smaller than φ. Note that φ is well defined since $|\mathcal{OUT}(\log B, \sigma)| = \zeta > \log(M/(2B))$. We use the previous argument, except for the segment length. Specifically, partition the sequence of N accesses to V into $N/(2^{\varphi}M/(2B))$ segments and let $\mathcal{F}^i = \{\Pi_N^{\pi_\sigma}(i2^{\varphi}M/(2B)), \ldots, \Pi_N^{\pi_\sigma}((i+1)2^{\varphi}M/(2B) - 1\}$, with $0 \leq i < N/(2^{\varphi}M/(2B))$, be the set of the indices of the entries required in the i-th segment. The binary representations of the values in $\mathcal{F}^i$ differ on $\varphi + \log(M/(2B))$ bit positions, and $(\log(M/2B))$ of these are $(\log B)$-incoming bit positions of σ. Then the $2^{\varphi}M/(2B)$ entries of V with indices in $\mathcal{F}^i$ are distributed among $M/(2B)$ blocks. An argument similar to the one used above proves that these $2^{\varphi}M/(2B)$ entries are permuted into at most $M/(2B)$ blocks of the output vector U. Therefore, the permutation steps requires $O\left(N/2^{\varphi}\right) = O\left(NB/M\right)$ misses, since $\varphi \geq \log(M/(2B))$, and Equation 3.19 follows. The proof of Equation 3.18 is straightforward. □

By Theorem 3.10 and the lower bounds on the work and cache complexities given in Section 3.2, the cache-oblivious algorithm in Figure 3.3 is optimal when $M/B \geq 2^{1+\zeta(\log B, \sigma)}$. Since $\zeta(\log B, \sigma) \leq \log B$, the tall-cache assumption (i.e., $M \geq B^2$) is sufficient to guarantee cache and work optimality of the cache-oblivious algorithm for each rational permutation. Recall that by Corollary 3.5, there exists a cache-aware algorithm for performing rational permutations which exhibits optimal cache and work complexities for all values of the IC parameters. In the next section, we will show that an optimal cache-oblivious algorithm for *all* values of the IC parameters cannot exist.

3.4 Limits of cache-oblivious rational permutations

Theorem 3.4 proves that the work complexity of a cache-optimal algorithm is $\omega(N)$ when $M/B \in o\left(2^{\zeta_\Sigma(\log B, N)}\right)$, and $\Theta\left(N\right)$ otherwise. Clearly, the work complexity of a cache-oblivious algorithm is independent of the cache parameters (this is not the case, in general, for cache complexity). Hence, a cache-oblivious algorithm cannot have optimal work complexity for each value of M and B. One can wonder whether there exists a cache-oblivious algorithm which is cache-optimal for each M and B, regardless of the work complexity. In this section we will prove that such an algorithm cannot exist. To this purpose we follow a similar approach to the one employed in [BF03].

Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$ and let $\mathcal{A}$ be a cache-oblivious algorithm which performs any rational N-permutation defined by an n-permutation $\sigma \in \Sigma$ on a vector of N entries. Consider the two sequences of misses generated by the executions of $\mathcal{A}$ in two different ICs, where one model satisfies a particular assumption we will define, while the other does not. We simulate these two executions in the EM model and obtain a new EM algorithm solving the same problem as $\mathcal{A}$. By adapting the argument described in Subsection 3.2 to bound from below the number of disk accesses, we conclude that $\mathcal{A}$ cannot be optimal in both ICs.

3.4.1 The simulation technique

In this subsection we describe a technique for obtaining an EM algorithm from two executions of a cache-oblivious algorithm in two different IC models. The technique is presented in a general form and is a formalization of the *ad-hoc* one employed in [BF03] for proving the impossibility result for general permutations.

Consider two models $\mathcal{C}_1$=$\mathsf{IC}(M, B_1)$ and $\mathcal{C}_2 = \mathsf{IC}(M, B_2)$, where $B_1 < B_2$. For convenience, we assume B_2 to be a multiple of B_1. Let $\mathcal{A}$ be a cache-oblivious algorithm for an *arbitrary* problem and let Q_1 and Q_2 be its cache complexities in the two models, respectively. We define an algorithm $\mathcal{A}'$ for $\mathsf{EM}(2M, B_2)$ which emulates in parallel the executions of $\mathcal{A}$ in both $\mathcal{C}_1$ and $\mathcal{C}_2$ and solves the same problem as $\mathcal{A}$.

Let us regard the memory in $\mathsf{EM}(2M, B_2)$ as partitioned into two contiguous portions of size M each, which we refer to as $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. In turn, portion $\mathcal{M}_1$ is subdivided into blocks of B_1 words (which we call B_1-*rows*), and portion $\mathcal{M}_2$ is subdivided into blocks of B_2 words (which we call B_2-*rows*), so that we can establish a one-to-one mapping between the cache lines of $\mathcal{C}_1$ and the B_1-rows of $\mathcal{M}_1$, and a one-to-one mapping between the cache lines of $\mathcal{C}_2$ and the B_2-rows of $\mathcal{M}_2$. Algorithm $\mathcal{A}'$ is organized so that its I/Os coincide (except for some slight reordering) with the I/Os performed by $\mathcal{A}$ in $\mathcal{C}_2$, and occur exclusively between the disk and $\mathcal{M}_2$. On the other hand, $\mathcal{A}'$ executes all operations prescribed by $\mathcal{A}$ on data in $\mathcal{M}_1$. [8] Since there are no I/Os between $\mathcal{M}_1$ and the disk, data are inserted into $\mathcal{M}_1$ by means of transfers of B_1-rows between $\mathcal{M}_1$ and $\mathcal{M}_2$, which coincide with the I/Os performed by $\mathcal{A}$ in $\mathcal{C}_1$.

Let us now see in detail how the execution of $\mathcal{A}'$ in the $\mathsf{EM}(2M, B_2)$ develops. Initially all the words in $\mathcal{M}_1$ and $\mathcal{M}_2$ are empty, that is filled with NIL values, and the EM disk contains the same data as the memory of $\mathcal{C}_2$ (or $\mathcal{C}_1$ indistinguishably) with the same layout (a one-to-one relation between the B_2-blocks of $\mathcal{C}_2$ and the B_2-blocks of the disk can be simply realized). Let o_i be the i-th operation of $\mathcal{A}$, $i = 1 \ldots h$. The execution of $\mathcal{A}$ in C_i, $1 \leq i \leq 2$, can be seen as a sequence $\mathcal{L}_i$ of operations interleaved with I/Os. Since operations in $\mathcal{L}_1$ and $\mathcal{L}_2$ are the same, we build a new sequence $\mathcal{L}$=$\Gamma^2_1\Gamma^1_1 o_1 \ldots \Gamma^2_j\Gamma^1_j o_j \ldots \Gamma^2_h\Gamma^1_h o_h \Gamma^2_{h+1}\Gamma^1_{h+1}$. Each Γ^i_j, with $1 \leq j \leq h+1$ and $1 \leq i \leq 2$, is defined as follows:

- Γ^i_1 is the sequence of I/Os that precede o_1 in $\mathcal{L}_i$.
- Γ^i_j, $1 < j \leq h$, is the sequence of I/Os which are enclosed between o_{j-1} and o_j in $\mathcal{L}_i$.
- Γ^i_{h+1} is the sequence of I/Os performed after o_h in $\mathcal{L}_i$.

Note that a Γ^i_j can be empty. The length of $\mathcal{L}$, denoted as $|\mathcal{L}|$, is the sum of the number h of operations and the size of all Γ^i_j, with $1 \leq j \leq h+1$ and $1 \leq i \leq 2$. Let $\mathcal{A}'$ be divided into $|\mathcal{L}|$ phases. The behavior of the j-th phase is determined by the j-th entry l_j of $\mathcal{L}$:

1. *l_j is an operation:* $\mathcal{A}'$ executes the same operation in $\mathcal{M}_1$.

2. *l_j is an input of a B_2-block (i.e., an input of $\mathcal{L}_2$):* $\mathcal{A}'$ fetches the same B_2-block from the disk into the B_2-row of $\mathcal{M}_2$ associated with the line used in $\mathcal{C}_2$.

3. *l_j is an input of a B_1-block (i.e., an input of $\mathcal{L}_1$):* let γ be such a B_1-block and γ' be the B_2-block containing γ. Since there is no prefetch in the IC model, the next operation of $\mathcal{A}$ requires an entry in γ; thus γ' must be in the cache of $\mathcal{C}_2$, too. For this reason, we can assume that γ' was, or has just been, fetched into a B_2-row of $\mathcal{M}_2$. $\mathcal{A}'$ copies γ in the right B_1-row of $\mathcal{M}_1$ and replaces the copy of γ in $\mathcal{M}_2$ with B_1 NIL values.

4. *l_j is an output of a B_2-block (i.e., an output of $\mathcal{L}_2$):* $\mathcal{A}'$ moves the respective B_2-row of $\mathcal{M}_2$ to the disk, replacing it with B_2 NIL values.

[8]Note that the operations of $\mathcal{A}$ do not include I/Os since block transfers are automatically controlled by the machine. Moreover, $\mathcal{A}$'s operations are the same no matter whether execution is in $\mathcal{C}_1$ or $\mathcal{C}_2$.

5. *l_j is an output of a B_1-block (i.e., an output of $\mathcal{L}_1$):* let γ be such a B_1-block and γ' be the B_2-block containing γ. If γ' is still in $\mathcal{M}_2$, then $\mathcal{A}'$ copies γ from $\mathcal{M}_1$ into γ' and replaces γ's row with B_1 NIL values. The second possibility (i.e., γ' is not in $\mathcal{M}_2$) can be avoided since no operations are executed between the evictions of γ' and γ. If some operations were executed, both blocks γ and γ' would be kept in cache (and so in $\mathcal{M}_1$ and $\mathcal{M}_2$). Therefore, we can suppose γ was removed just prior to the eviction of γ'.

It is easy to see that every operation of $\mathcal{A}$ can be executed by $\mathcal{A}'$ in $\mathcal{M}_1$, since there is a one to one relation between the cache lines of C_1 and the rows of $\mathcal{M}_1$ (except for the B_1-blocks whose evictions from cache were anticipated, see fifth point). $\mathcal{M}_2$ is a quasi-mirror of C_2, in the sense that it contains the same B_2-blocks of C_2 while $\mathcal{A}$ is being executed, except for those sub B_1-blocks which are also in $\mathcal{M}_1$. By rules 2 and 4, the I/O complexity of $\mathcal{A}'$ is at most $2Q_2$ (note that a miss in the IC model is equivalent to at most two I/Os in the EM model).

Let $K = Q_1B_1/Q_2$; it is easy to see that $K \leq B_2$. Indeed, if K were greater than B_2, a replacement policy for $\mathcal{C}_1$ which requires $Q_2B_2/B_1 < Q_1$ misses would be built from the execution of $\mathcal{A}$ in $\mathcal{C}_2$; but this is a contradiction since the replacement policy of the IC model is optimal. $\mathcal{A}'$ can be adjusted so that there are at most K words exchanged between $\mathcal{M}_1$ and a B_2-block in $\mathcal{M}_2$ before this block is removed from the memory: it is sufficient to insert some dummy I/Os. This increases the I/O complexity of $\mathcal{A}'$ from $2Q_2$ to at most $2Q_2 + 2Q_1B_1/K = 4Q_2$ I/Os. In particular, there are at most $2Q_2$ inputs and $2Q_2$ outputs of B_2-blocks.

We define the *working set* $\mathcal{W}(q)$ after q I/Os as the content of $\mathcal{M}_1$ plus the words in the corresponding B_2-blocks of $\mathcal{M}_2$ that will be used by $\mathcal{A}'$ (moved to $\mathcal{M}_1$) before the B_2-blocks are evicted. When $\mathcal{A}'$ fetches a B_2-block from the disk, we can suppose that the at most K entries which will be moved between $\mathcal{M}_1$ and the block are immediately included in the working set.

Note that the EM algorithm $\mathcal{A}'$ could not be implementable in practice; however, since we are interested in lower bounding its I/O complexity, this is not relevant except in the case where the arguments used for bounding suppose the implementability of $\mathcal{A}'$. The argument used in the following section is based on a potential function and does not require the algorithm to be implementable.

3.4.2 Impossibility result for rational permutations

In this subsection we prove that a cache-oblivious algorithm which performs the rational permutations defined by a set Σ cannot be optimal for each value of the cache parameters.

Theorem 3.11. *Let Σ be an infinite set of permutations which contains at most one n-permutation for each $n \in \mathbb{N}$, and N range over $\{2^n : \exists$ an n-permutation $\sigma \in \Sigma\}$. Consider a cache-oblivious algorithm $\mathcal{A}$ which performs any rational N-permutation defined by an n-permutation $\sigma \in \Sigma$. If there exists a function $g(N) \leq \log(\delta N)$, with $\delta \in (0,1)$, such that $\zeta_\Sigma(g(N), N) \in \omega(1)$, then $\mathcal{A}$ cannot be cache-optimal for each value of the M and B parameters.*

Proof. We begin by asserting that a lower bound on the cache complexity in the IC model translates into a lower bound on the I/O complexity in the EM model, and vice versa, since the IC model adopts an optimal off-line replacement policy [SCD02]. Moreover, the lower bound provided in Theorem 3.3 is tight since it can be matched by an aware algorithm, as established in Corollary 3.5. Assume, for the sake of contradiction, that $\mathcal{A}$ attains optimal cache complexity for each value of M and B. In particular, consider two models $\mathcal{C}_1$=$\mathsf{IC}(M, B_1)$ and $\mathcal{C}_2 = \mathsf{IC}(M, B_2)$ where B_2 is a multiple of B_1, and let Q_1 and Q_2 be the cache complexities of $\mathcal{A}$ in the two models, respectively. We will show that B_1 and B_2 can be suitably chosen so that Q_1 and Q_2 cannot be both optimal, thus reaching a contradiction. To achieve this goal, we apply the simulation technique described in the previous subsection to $\mathcal{A}$, and obtain an algorithm $\mathcal{A}'$ for the $\mathsf{EM}(2M, B_2)$ solving the same problem as $\mathcal{A}$. We then apply an adaptation of Lemma 3.1 (which is based on a technical result given in [AV88] for bounding from below the number of disk accesses of matrix transposition in the EM model) to $\mathcal{A}'$, and we prove the impossibility of the simultaneous optimality of $\mathcal{A}$ in the two IC models. We denote with Q and Q_I the I/O complexity and the number of inputs of B_2-blocks, respectively, of $\mathcal{A}'$; recall that $Q \leq 4Q_2$ and $Q_I \leq 2Q_2$.

As in Section 3.2, we define the i-th target group, with $0 \leq i < N/B_2$, to be the set of entries that will ultimately be in the i-th B_2-block of the output vector (remember that it must be entirely in the disk at the end of $\mathcal{A}'$). Let γ be a B_2-block of the disk or a B_2-row of $\mathcal{M}_2$; the *togetherness rating of γ after q I/Os* is defined as:

$$C_\gamma(q) = \sum_{i=0}^{N/B_2-1} f(x_{\gamma,i}),$$

where $x_{\gamma,i}$ denotes the number of entries in γ belonging to the i-th target group just before the $(q+1)$-st I/O, and f is the convex function given in Equation 3.5. These entries are not included in the working set $\mathcal{W}(q)$ and are not NIL symbol. We also define the *togetherness rating for the working set* $\mathcal{W}(q)$ as:

$$C_{\mathcal{W}}(q) = \sum_{i=0}^{N/B_2-1} f(s_i),$$

where s_i is the number of entries in the working set $\mathcal{W}(q)$ which belong to the i-th target group just before the $(q+1)$-st I/O. The *potential function* of $\mathcal{A}'$ after q I/Os is defined as:

$$POT(q) = C_{\mathcal{W}}(q) + \sum_{\gamma \in disk} C_\gamma(q) + \sum_{\gamma \in \mathcal{M}_2} C_\gamma(q).$$

At the beginning and at the end of the algorithm the above definition is equivalent to the one given in Section 3.2. Then by Equations 3.6,

$$POT(0) = N \log(B_2/2^{\zeta_\Sigma(\log B_2, N)}), \qquad POT(Q) = N \log B_2.$$

Hence, $POT(Q) - POT(0) = N\zeta_\Sigma(\log B_2, N)$.

We now bound the increase in the potential function due to the input of a B_2-block since the eviction of a block from the memory does not increase the potential. Suppose that the q-th I/O is an input and a B_2-block γ is fetched into a B_2-row of $\mathcal{M}_2$. Before the q-th input, the intersection between γ and the working set $\mathcal{W}(q-1)$ was empty; after the input, at most $K = Q_1B_1/Q_2$ entries of γ are inserted into $\mathcal{W}(q-1)$. We use the following notation:

- s_i: number of entries in the working set $\mathcal{W}(q-1)$ belonging to the i-th target group;
- k_i: number of entries in γ belonging to the i-th target group just before the q-th miss;
- w_i: number of entries in the (at most) K words, inserted in $\mathcal{W}(q-1)$, belonging to the i-th target group.

The s_i, k_i and w_i values are limited by the following constraints:

$$\sum_{i=0}^{N/B_2-1} s_i \leq 2M - K, \qquad \sum_{i=0}^{N/B_2-1} k_i \leq B_2, \qquad \sum_{i=0}^{N/B_2-1} w_i \leq K.$$

The increase in the potential function due to the q-th miss ($\Delta POT(q)$) is:

$$\Delta POT(q) = \sum_{i=0}^{N/B_2-1} \left[f(s_i + w_i) + f(k_i - w_i) - f(s_i) - f(k_i) \right].$$

By Lemma A.1, we have that $f(k_i - w_i) - f(k_i) \leq -f(w_i)$. Thus,

$$\Delta POT(q) \leq \sum_{i=0}^{N/B_2-1} \left[f(s_i + w_i) - f(s_i) - f(w_i) \right]. \tag{3.20}$$

According with Corollary A.4 an upper bound on $\Delta POT(q)$ is obtained by setting $s_i = (2M - K)/(N/B_2)$ and $w_i = K/(N/B_2)$ in Inequality 3.20:

$$\begin{aligned}\Delta POT(q) &\leq \sum_{i=0}^{N/B_2-1} \left[s_i \log \frac{s_i + w_i}{s_i} + w_i \log \frac{s_i + w_i}{w_i} \right] \\ &\leq K \log e + K \log \frac{2M}{K} = K \log \frac{2eM}{K},\end{aligned}$$

since $(1 + 1/x)^x \leq e$ if $x \geq 1$. Let $\mathcal{C}_1$ be a cache with more than $2^{\zeta_\Sigma(\log B_1, N)}$ lines, while $\mathcal{C}_2$ be a cache with less than $2^{\zeta_\Sigma(\log B_2, N)}$ lines. By Theorem 3.3, $cN/B_1 \leq Q_1 \leq dN/B_1$ for two suitable positive constants c and d. Since the number of input operations is $Q_I \leq 2Q_2$ (remember that the output of a block does not increase the potential and that $K = Q_1 B_1 / Q_2$), we have that

$$POT(Q) - POT(0) \leq \sum_{q=1}^{Q_I} \Delta POT(q) \leq 2Q_2 K \log \frac{2eM}{K} \leq 2dN \log \frac{2eMQ_2}{cN}.$$

By recalling that $POT(Q) - POT(0) = N \log 2^{\zeta_\Sigma(\log B_2, N)}$,

$$N \log 2^{\zeta_\Sigma(\log B_2, N)} \leq 2dN \log \frac{2eMQ_2}{cN}.$$

Hence,

$$Q_2 \in \Omega\left(N \frac{2^{\frac{\zeta_\Sigma(\log B_2, N)}{2d}}}{M} \right). \tag{3.21}$$

By setting $B_2 = \delta' M$ and $M = 2^{g(N)}/\delta'$, for a constant δ, with $0 < \delta' < \delta$, we have

$$\zeta_\Sigma(\log B_2, N) = \zeta_\Sigma(g(N), N) \in \omega(1),$$

which yields (the asymptotic is on N)

$$Q_2 \in \omega\left(N\frac{\zeta_\Sigma(g(N),N)}{M}\right).$$

However, by optimality of $\mathcal{A}$ and Theorem 3.3, Q_2 must be $\Theta\left(N\frac{\zeta_\Sigma(g(N),N)}{M}\right)$ when $B_2 = \delta' M$, which yields a contradiction. □

Corollary 3.12. *There cannot exists a cache-oblivious algorithm for matrix transposition or bit-reversal that yields optimality for all values of the IC parameters.*

Proof. Matrix transposition and bit-reversal are examples of rational permutations which, by Equation 3.3 and 3.4, satisfy the hypothesis of Theorem 3.11 by setting $g(N) = \lceil \log \sqrt{N} \rceil$. Thus, Theorem 3.11 implies that cache-oblivious algorithms for matrix transposition or the bit-reversal of a vector cannot exhibit optimal cache complexity for all values of the cache parameters. □

Note that Theorem 3.11 do not rule out the existence of an optimal cache-oblivious algorithm for some particular ranges of the cache parameters. Indeed by Theorem 3.10, there exists an optimal cache-oblivious algorithm under the tall-cache assumption.

Chapter 4

Network-Oblivious Algorithms

I'm totally, totally oblivious. Usually.
(Paul Hewitt)

As seen in Section 2.2, a number of parallel models aim at realizing an efficiency/ portability/ design-complexity tradeoff by capturing features common to most machines through a number of parameters. One parameter present in virtually all models is the number of processors, and most models also exhibit parameters describing the time required to route certain communication patterns. Increasing the number of parameters, from just a small constant to logarithmically many in the number of processors (like in the D-BSP), can considerably increase the effectiveness of the model with respect to realistic architectures, such as point-to-point networks, as extensively discussed in [BPP07]. However, a price is paid in the increased complexity of algorithm design necessary to gain greater efficiency across a larger class of machines.

It is natural to wonder whether, at least for some problems, algorithms can be designed that, while independent of any machine/model parameters, are nevertheless efficient for a wide range of such parameters. In other words, we are interested in exploring the world of efficient *network-oblivious* algorithms, in the same spirit as the exploration of efficient *cache-oblivious* algorithms proposed in [FLPR99].

Of course, the first step is to develop a framework where the concept of network-obliviousness and of algorithmic efficiency are precisely defined. The framework we propose is based on three models of computation, each with a different role, as briefly outlined next.

- *Specification model.* This model, denoted by $\mathsf{M}(n)$, is a set of n CPU/memory nodes, called processing elements (PEs), computing in supersteps, and able to exchange messages. Network-oblivious algorithms will be formulated in this model. The number of PEs is chosen by the algorithm designer exclusively as a function of the input size n. (Reasonably, n reflects the amount of parallelism of the algorithm at hand).

- *Evaluation model.* This model, denoted by $\mathsf{M}(p, B)$, has two parameters: the number of processors, p, and a block size, B, which models the fixed payload size of any message exchanged by two processors. As for $\mathsf{M}(n)$, the computation is organized in supersteps. A cost function is defined, called the block degree of a superstep, which, when summed over all supersteps of an algorithm, gives the *communication complexity* of the algorithm. An $\mathsf{M}(n)$ algorithm will execute on an $\mathsf{M}(p, B)$, with $p \leq n$, by letting each $\mathsf{M}(p, B)$ processor carry out the work of a pre-specified set of n/p PEs of $\mathsf{M}(n)$.

  The quality of a network-oblivious algorithm $\mathcal{A}$, with input size n, is defined with respect to the communication complexity $H_{\mathcal{A}}(n, p, B)$ of its execution on $\mathsf{M}(p, B)$, by measuring how close $H_{\mathcal{A}}(n, p, B)$ comes to the minimum communication complexity $H^*(n, p, B)$ achievable by any $\mathsf{M}(p, B)$ algorithm solving the same problem as $\mathcal{A}$. Algorithm $\mathcal{A}$ is optimal if $H_{\mathcal{A}}(n, p, B) = O\left(H^*(n, p, B)\right)$, for a suitably large range of (p, B) values.

- *Execution machine model.* This model aims at describing the set of platforms on which we expect the network-oblivious algorithm to be actually executed. Technically, we adopt for this role the block-based variant of the Decomposable Bulk Synchronous Parallel model, $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$, [BPP07] described in Section 2.2.1, where $\boldsymbol{g}$ and $\boldsymbol{B}$ are vectors of length $\log P$.

  Fortunately, as shown in the next section, for a wide and interesting class of network-oblivious algorithms, optimality with respect to the $\mathsf{M}(p, B)$ model, for suitable ranges of (p, B), translates into optimality with respect to the $\mathsf{D\text{-}BSP}(p, \boldsymbol{g}, \boldsymbol{B})$, for suitable ranges of $\boldsymbol{g}$ and $\boldsymbol{B}$. It is this circumstance that motivates the introduction of the evaluation model, as a tool to substantially simplify the performance analysis of oblivious algorithms.

To help placing our network-oblivious framework in perspective, it may be useful to compare it with the well established cache-oblivious framework [FLPR99]. In the latter, the algorithm formulation model is the Random Access Machine; the algorithm evaluation model is the Ideal Cache model $\mathsf{IC}(M, B)$, a machine with only one level of cache of size M and line length B; and the machine execution model is a machine with a hierarchy of caches, each with its own size and line length. In the cache-oblivious context, the simplification in the analysis arises from the fact that, under certain conditions, optimality on $\mathsf{IC}(M, B)$ for all values of M and B translates into optimality on multilevel hierarchies.

The structure of this chapter is as follows. In Section 4.1, we define rigorously the three models relevant to the framework and establish the key relations among them. In Section 4.2, we illustrate the framework by deriving positive and negative results on network-oblivious algorithms for key problems, such as matrix multiplication and transposition, FFT, and sorting. We also show that matrix transposition does not admit network-oblivious algorithms, for some range of parameters. The results presented in this chapter were published in [BPPS07].

4.1 The framework

In this section, we introduce the models of computation for the formulation and analysis of network-oblivious algorithms, and develop some key relations between these models, which provide the justification for the framework.

Specification model. Let Π be a given computational problem and let n (for simplicity, a power of two) be a suitable function of the input size. A *network-oblivious* algorithm $\mathcal{A}$ for Π is designed for a complete network $\mathsf{M}(n)$ of n *Processing Elements* (PEs), $\mathrm{PE}_0, \ldots, \mathrm{PE}_{n-1}$, each consisting of a CPU and an unbounded local memory. $\mathcal{A}$ consists of a sequence of *labeled supersteps*[1], with labels in the integer range $[0, \log n)$. For $0 \leq i < \log n$ and $0 \leq j < n$, in an i-superstep, PE_j can perform operations on locally held data, and send words of data only to any PE_k whose index k agrees with j in the i most significant bits, that is, $\lfloor j2^i/n \rfloor \leq k < \lfloor j2^i/n \rfloor + n/2^i$. The superstep ends with a global synchronization.

Evaluation model. In order to analyze $\mathcal{A}$'s communication complexity on different machines, we introduce the machine model $\mathsf{M}(p, B)$, where the parameters p and B are positive integers (for simplicity, powers of two). $\mathsf{M}(p, B)$ is essentially an $\mathsf{M}(p)$ with a communication cost function parameterized by B, whose processing elements are called *processors* and denoted as p_j, with $0 \leq j < p$, to distinguish them from those of $\mathsf{M}(n)$. Words exchanged between two processors in a superstep can be envisioned as traveling within *blocks* of fixed size B (in words). For each superstep we define the *block-degree* as the maximum number of blocks sent/received by a single processor in that superstep. More formally, the block-degree of a superstep s where

[1]The results would hold even if, in the various models considered, supersteps were not explicitly labeled. However, explicit labels can help reduce synchronization costs; they become crucial for efficient simulation of algorithms on point-to-point networks, especially those of large diameter.

processor p_j sends w^s_{jk} words to p_k, with $0 \leq j, k < p$, is defined as

$$h^s(p, B) = \max_{0 \leq j < p} \left\{ \max \left(\sum_{k=0}^{p-1} \lceil w^s_{jk}/B \rceil, \sum_{k=0}^{p-1} \lceil w^s_{kj}/B \rceil \right) \right\}.$$

The *communication complexity* of an algorithm is the sum of the block-degrees of its supersteps. Hence, the model rewards batched over fine-grained communication. The quantity $h^s = h^s(p, 1)$ is also called the *word-degree* of superstep s. Clearly, $\lceil h^s/B \rceil \leq h^s(p, B) \leq h^s$. Although the network-oblivious framework is primarily concerned with communication, we define a cost function, called *computation complexity*, which measures, in some sense, the degree of parallelism. The computation complexity of an algorithm is the sum over all supersteps of the maximum number of operations performed by a processor on locally held data during a superstep.

A network-oblivious algorithm $\mathcal{A}$ formulated for $\mathsf{M}(n)$ can be naturally executed on an $\mathsf{M}(p, B)$ machine, for every $1 \leq p \leq n$ and for every B, by stipulating that processor p_j, with $0 \leq j < p$, of $\mathsf{M}(p, B)$ will carry out the operations of the n/p consecutively numbered processing elements of $\mathsf{M}(n)$ starting with $\mathrm{PE}_{(n/p)j}$. Supersteps with a label $i < \log p$ on $\mathsf{M}(n)$ become supersteps with the same label on $\mathsf{M}(p, B)$; supersteps with label $i \geq \log p$ become local computation. Let us number the supersteps of $\mathcal{A}$ from 1 to S, where S is the number of supersteps executed by the algorithm on $\mathsf{M}(p, B)$, and let $h^s(n, p, B)$ be the block-degree of the execution of superstep s. The central quantity in our analysis is the communication complexity

$$H_{\mathcal{A}}(n, p, B) = \sum_{s=1}^{S} h^s(n, p, B),$$

of $\mathcal{A}$ on an $\mathsf{M}(p, B)$, for varying p and B.

With regard to the computation complexity of a network-oblivious algorithm for $\mathsf{M}(n)$, we observe that the simulation of a superstep s on $\mathsf{M}(p, B)$ requires $O\left((n/p)(\tau^s + h^s)\right)$ operations, where τ^s is the maximum number of operations executed by a PE on locally held data and h^s is the maximum number of words sent/received by a PE (i.e, the word-degree on $\mathsf{M}(n, 1)$); $O\left((n/p)h^s\right)$ is the cost of a naïve message dispatching. Without loss of generality we assume $h^s \in O\left(\tau^s\right)$, that is, $\Omega\left(1\right)$ operations are performed for each sent or received message. We denote with $T_{\mathcal{A}}(n, p, B)$ the computation complexity of a network-oblivious algorithm $\mathcal{A}$ with input size n on $\mathsf{M}(p, B)$.

As a cache-oblivious algorithm "ignores", hence cannot explicitly use, cache size and line length, so does a network-oblivious algorithm ignore, hence cannot explicitly use, the actual number of processors that will carry out the computation, and the block size of the communication.

Definition 4.1. *A network-oblivious algorithm $\mathcal{A}$ for a problem Π is optimal if for every p and B, with $1 < p \leq n$ and $B \geq 1$, the execution of $\mathcal{A}$ on an $\mathsf{M}(p, B)$ machine yields an algorithm with asymptotically minimum communication complexity among all $\mathsf{M}(p, B)$ algorithms for Π.*

Execution machine model. To substantiate the usefulness of the above definition, we now show that, under certain assumptions, an optimal network-oblivious algorithm can run optimally on an wide class of parallel machines, whose underlying interconnection network exhibits a hierarchical structure with respect to its bandwidth characteristics. To model machines in this class, we use the block variant of the Decomposable BSP (D-BSP) model described in Section 2.2.1, denoted as a $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$, where $\boldsymbol{g} = (g_0, g_1, \ldots g_{\log P-1})$ and $\boldsymbol{B} = (B_0, B_1, \ldots B_{\log P-1})$. Note that a $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ is essentially an $\mathsf{M}(P)$ machine, where the *communication time* of superstep s is defined to be $h^s(P, B_i)g_i$, where i is the label of the superstep and $h^s(P, B_i)$ denotes the block-degree of the superstep.

In the reminder of this section, we show that an optimal network-oblivious algorithm $\mathcal{A}$ translates into an optimal D-BSP algorithm under some reasonable assumptions on the communication pattern employed by the algorithm and on the machine parameters. We begin with the following technical lemma.

Lemma 4.2. *For $m \geq 1$, let $\langle X_0, X_1, \ldots, X_{m-1}\rangle$ and $\langle Y_0, Y_1, \ldots, Y_{m-1}\rangle$ be two arbitrary sequences of nonnegative integers, and let $\langle f_0, f_1, \ldots, f_{m-1}\rangle$ be a nonincreasing sequence of nonnegative real values. If $\sum_{j=0}^{i} X_j \leq \sum_{j=0}^{i} Y_j$, for every $0 \leq i < m$, then*

$$\sum_{j=0}^{m-1} X_j f_j \leq \sum_{j=0}^{m-1} Y_j f_j.$$

Proof. By defining $S_{-1} = 0$ and $S_j = \sum_{i=0}^{j}(Y_i - X_i) \geq 0$, for $0 \leq j \leq m-1$, we have:

$$\sum_{j=0}^{m-1} f_j(Y_j - X_j) = \sum_{j=0}^{m-1} f_j(S_j - S_{j-1}) = \sum_{j=0}^{m-1} f_j S_j - \sum_{j=1}^{m-1} f_j S_{j-1} \geq$$
$$\geq \sum_{j=0}^{m-1} f_j S_j - \sum_{j=1}^{m-1} f_{j-1} S_{j-1} = f_{m-1} S_{m-1} \geq 0.$$

$\square$

In the next definitions, we introduce some useful parameters and properties of network-oblivious algorithms.

Definition 4.3. *Given an algorithm $\mathcal{A}$ for $\mathsf{M}(n)$, we define i-granularity b_i, for $0 < i \leq \log n$, the minimum number of words ever exchanged by two communicating PEs in any superstep of the execution of $\mathcal{A}$ on an $\mathsf{M}(2^i, 1)$.*

In other words, when executing $\mathcal{A}$ on $\mathsf{M}(2^i, 1)$, in any superstep, if p_j sends any words to p_k, then it sends at least b_i words to it.

Definition 4.4. *Let $\alpha > 0$ be constant. An algorithm $\mathcal{A}$ for $\mathsf{M}(n)$ is said to be (α, P)-wise if, for any i with $1 < 2^i \leq P$, we have*

$$H_{\mathcal{A}}(n, 2^i, 1) \geq \alpha \frac{n}{2^i} \sum_{s \in L_i} h^s(n, n, 1).$$

where L_i is the set of indices of the supersteps with labels $j < i$.

To put the above definition into perspective, we observe that an algorithm where for each j-superstep and for every $i > j$ there is always at least one segment of $n/2^i$ consecutively numbered PEs each communicating the maximum amount of words for that superstep to PEs outside the segment, is surely an (α, P)-wise algorithm. However, (α, P)-wiseness holds even if the aforementioned communication scenario is realized only in an average sense.

Many algorithms are likely to exhibit a good level of granularity and can be arranged to be (α, P)-wise. Indeed, this is the case for all network-oblivious algorithms presented in this chapter. Quite interestingly, these algorithms achieve optimal performance on D-BSP, as better established in the following theorem.

Theorem 4.5. *Let $\mathcal{A}$ be an (α, P^*)-wise optimal network-oblivious algorithm for a problem Π with input size n, specified for the $\mathsf{M}(n)$ model, with i-granularity b_i, for $0 \leq i < \log P^*$. Then, $\mathcal{A}$ exhibits asymptotically optimal communication time when executed on any $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$, with $P \leq P^*$ and $B_i \leq b_{\log P}$, for $0 \leq i < \log P$.*

Proof. Let $D_{\mathcal{A}}(i)$, with $0 \leq i < \log P$, be the sum of block-degrees of all i-supersteps when $\mathcal{A}$ is executed on $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$. From the hypothesis on the granularity of $\mathcal{A}$, we have that the minimum amount of words ever exchanged by two communicating PEs in any superstep is at least $b_{\log P} \geq B_i$, for every $0 \leq i < \log P$. Hence,

$$D_{\mathcal{A}}(i) \leq 2\frac{n}{P} \sum_{s \in L_{i+1} \setminus L_i} \frac{h^s(n, n, 1)}{B_i}, \qquad \forall 0 \leq i < \log P.$$

Since $\mathcal{A}$ is (α, P^*)-wise, we have that

$$H_{\mathcal{A}}(n, 2^i, B_i) \geq \frac{H_{\mathcal{A}}(n, 2^i, 1)}{B_i} \geq \alpha \sum_{s \in L_i} \frac{n}{2^i} \frac{h^s(n, n, 1)}{B_i}$$
$$\geq \alpha \sum_{j=0}^{i-1} \frac{n}{2^i B_i} \sum_{s \in L_{j+1} \setminus L_j} h^s(n, n, 1) \geq \frac{\alpha}{2} \sum_{j=0}^{i-1} \frac{P}{2^i} D_{\mathcal{A}}(j) \frac{B_j}{B_i}.$$

By definition, the overall communication time of $\mathcal{A}$ on $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ is $H = \sum_{j=0}^{\log P - 1} D_{\mathcal{A}}(j) g_j$. Suppose $\mathcal{A}'$ were an asymptotically faster $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ algorithm for Π. Then, for every constant $\epsilon > 0$ and sufficiently large input size n, $\mathcal{A}'$ would exhibit communication time $H' < \epsilon H$, so that, with obvious notation,

$$\sum_{j=0}^{\log P - 1} D_{\mathcal{A}'}(j) g_j < \epsilon \sum_{j=0}^{\log P - 1} D_{\mathcal{A}}(j) g_j.$$

The above relation can be rewritten as

$$\sum_{j=0}^{\log P - 1} D_{\mathcal{A}'}(j) B_j \frac{g_j}{B_j} < \epsilon \sum_{j=0}^{\log P - 1} D_{\mathcal{A}}(j) B_j \frac{g_j}{B_j}.$$

Recalling from Section 2.2.1 that the ratios g_i/B_i are assumed to be non-increasing, we can apply Lemma 4.2, with $m = \log P$, $f_j = g_j/B_j$, $X_j = \epsilon D_{\mathcal{A}}(j) B_j$, and $Y_j = D_{\mathcal{A}'}(j) B_j$, to show that there exists an $i \leq \log P$ such that

$$\sum_{j=0}^{i-1} D_{\mathcal{A}'}(j) B_j < \epsilon \sum_{j=0}^{i-1} D_{\mathcal{A}}(j) B_j.$$

Now, we can naturally interpret $\mathcal{A}'$ as an $\mathsf{M}(2^i, B_i)$ algorithm, whose communication complexity satisfies

$$H_{\mathcal{A}'}(n, 2^i, B_i) \leq \sum_{j=0}^{i-1} \frac{P}{2^i} D_{\mathcal{A}'}(j) \frac{B_j}{B_i} < \epsilon \sum_{j=0}^{i-1} \frac{P}{2^i} D_{\mathcal{A}}(j) \frac{B_j}{B_i} \leq \frac{2\epsilon}{\alpha} H_{\mathcal{A}}(n, 2^i, B_i),$$

which is a contradiction, since $2\epsilon/\alpha$ is an arbitrary value and, by definition, $\mathcal{A}$ is asymptotically optimal for $\mathsf{M}(2^i, B_i)$. (Note that in the above inequalities we used the fact that the B_j's are powers of two and non-increasing.) $\square$

As a final remark, observe that by setting all block sizes equal to 1, the above framework can be specialized to the case where the block transfer feature is not accounted for.

4.2 Algorithms for key problems

In this section we present optimal network-oblivious algorithms for a number of relevant computational problems, namely matrix multiplication, matrix transposition, FFT and sorting. In some cases, optimality requires additional constraints on the relative values of some machine and input parameters and, in one case, we will prove that these constraints are necessary to obtain network-oblivious optimality. The correctness of the algorithms will not be discussed since they are straightforward (e.g., matrix transposition) or already proved in the literature (e.g., matrix multiplication, FFT, sorting).

This section is organized as follows. In Sections 4.2.1 and 4.2.2 we describe the network-oblivious algorithms for matrix multiplication and transposition, respectively. In Section 4.2.3 we show that there cannot exist a network-oblivious algorithm for matrix transposition which is optimal for all values of the parameters. Then, in Sections 4.2.4 and 4.2.5, we provide network-oblivious algorithms for FFT and sorting, respectively.

4.2.1 Matrix multiplication

The *n-MM problem* requires to multiply two $n \times n$ matrices using only semiring operations [Ker70]. In this section we propose two network-oblivious algorithms which both yield optimal communication complexities, while they exhibit different memory requirements. We first establish lower bounds on the communication complexity of any $\mathsf{M}(p, B)$ algorithm for this problem.

Theorem 4.6. *Let $\mathcal{A}$ be any algorithm solving the n-MM problem on an $\mathsf{M}(p, B)$, with $1 < p \le n^2$ and $B \ge 1$. If initially the inputs are evenly distributed among the p processors, then the communication complexity of the algorithm is*

$$\Omega\left(\frac{n^2}{Bp^{2/3}}\right). \tag{4.1}$$

Moreover, if each processor cannot contain more than $O\left(n^2/p\right)$ entries of the input and output matrices in each superstep, then the communication complexity is:

$$\Omega\left(\frac{n^2}{B\sqrt{p}}\right). \tag{4.2}$$

Proof. The lower bound in Equation 4.2 is a consequence of [ITT04, Theorem 4.1]: this theorem requires the local memory of each processor to be upper bounded by $O(n^2/p)$, thus making no distinction between the local memory used for storing input and output entries (and their possible copies) and the local memory used for others data structures (e.g., the space required for simulating network-oblivious algorithms on an $\mathsf{M}(p, B)$). Nevertheless, the proof remains valid when imposing the $O(n^2/p)$ limit only to the number of input and output entries in a processor.

Irony et al. [ITT04, Lemma 5.1] provided the lower bound in Equation 4.1 assuming a $O\left(n^2/p^{2/3}\right)$ upper bound on the local memory of each processor. However, the same lower bound was proved, without the assumption, in [Pie95, CFSV95] for $\mathsf{M}(p, 1)$. □

Next, we describe an optimal network-oblivious algorithm, which we name *N-MM* (*Network-oblivious Matrix Multiplication*), for the n-MM problem whose communication complexity matches the lower bound in Equation 4.1. The algorithm is specified on $\mathsf{M}(n^2)$. Let A, B and C denote the two input matrices and the output matrix, respectively, and suppose that their entries are evenly distributed among the PEs. For $0 \leq i, j < n$, we denote with $P(i, j)$ the processing element PE_{in+j} of $\mathsf{M}(n^2)$, and require that such a PE holds $A[i, j]$, $B[i, j]$, and $C[i, j]$. We denote a quadrant of matrix E ($E \in \{A, B, C\}$) with E_{hk}, with $h, k \in \{0, 1\}$.[2] Let $\ell \in \{0, 1\}$, define $M_{hk\ell} = A_{h\ell} \cdot B_{\ell k}$, whence $C_{hk} = M_{hk0} + M_{hk1}$. The algorithm is based on the following simple recursive strategy, where we denote with m and q the number of rows/columns of the (sub)matrices involved and the number of assigned PEs, respectively (initially, $m = n$ and $q = n^2$).

1. Regard the q PEs as partitioned into *eight* segments $S_{hk\ell}$, with $h, k, \ell \in \{0, 1\}$, of $q/8$ PEs each. Replicate and distribute the inputs so that the entries of $A_{h\ell}$ and $B_{\ell k}$ are evenly spread among the PEs in $S_{hk\ell}$. (Note that each $S_{hk\ell}$ is an $\mathsf{M}(q/8)$ machine.)

2. For $h, k, \ell \in \{0, 1\}$ in parallel, compute recursively the product $M_{hk\ell}$ within $S_{hk\ell}$.

3. For $h, k \in \{0, 1\}$ in parallel, M_{hk0} and M_{hk1} are evenly distributed among PEs in S_{hk0} and S_{hk1} is such a way that $M_{hk0}[i, j]$ and $M_{hk1}[i, j]$ are contained by the same PE for each $0 \leq i, j < m/2$.

[2]Quadrants are the following: E_{00} (top-left), E_{01} (top-right), E_{10} (bottom-left), E_{11} (bottom-right).

4. For $0 \leq i, j < m$ in parallel, $C[i,j]$ is computed by adding $M_{hk0}[i,j]$ and $M_{hk1}[i,j]$.

The recurrence stops when only one PE is assigned to a subproblem (i.e., $q = 1$). The parameter m decreases by a factor two, while the number q of PEs assigned to each subproblem by a factor eight. Since, at the beginning of the algorithm, $m = n$ and $q = n^2$, each PE, in the base case, must multiply sequentially two matrices of size $n^{1/3} \times n^{1/3}$.

Theorem 4.7. *The communication and computation complexities of the network-oblivious algorithm N-MM for the n-MM problem, when executed on an $\mathsf{M}(p,B)$ machine, with $1 < p \leq n^2$ and $1 \leq B \leq n^2/p$, are*

$$H_{\text{N-MM}}(n,p,B) \in \Theta\left(\frac{n^2}{Bp^{2/3}}\right), \tag{4.3}$$

$$T_{\text{N-MM}}(n,p,B) \in \Theta\left(\frac{n^3}{p}\right), \tag{4.4}$$

which are optimal for all values of p and B in the specified ranges. The algorithm requires a $O\left(n^{2/3}\right)$ memory blow-up per processor.

Proof. Consider the execution of a recursive call with input size m and q assigned PEs, and let r be the number of $\mathsf{M}(p,B)$ processors that simulate the q PEs ($r \leq q$). Let $H(m,r)$ be the communication complexity of a recursive call to N-MM. Since there is no communication when the q PEs are simulated by the same processor, it follows that (for simplicity B and q are omitted from $H(m,r)$):

$$H(m,r) \leq \begin{cases} H\left(\frac{m}{2}, \frac{r}{8}\right) + O\left(\frac{m^2}{Br}\right) & \text{if } r > 1 \\ 0 & \text{if } r \leq 1 \end{cases}$$

which yields

$$H(m,r) \in O\left(\frac{m^2}{Br^{2/3}}\right).$$

Equation 4.3 follows by setting $m = n$, $q = n^2$ and $r = p$; its optimality descends from Theorem 4.6. Let $T(m,q,r)$ denote the computation complexity of a recursive call to N-MM, where m, q and r are defined as in $H(m,r)$; clearly, $T_{\text{N-MM}}(n,p,B) =$

$T(n, n^2, p)$. $T(m, q, r)$ is upper bounded by the following recurrence:

$$T(m,q,r) \leq \begin{cases} T\left(\frac{m}{2}, \frac{q}{8}, \frac{r}{8}\right) + O\left(\frac{m^2}{r}\right) & \text{if } r > 1 \\ 8T\left(\frac{m}{2}, \frac{q}{8}, 1\right) + O\left(m^2\right) & \text{if } r \leq 1 \text{ and } q > 1 \\ O\left(m^3\right) & \text{if } r \leq 1 \text{ and } q \leq 1 \end{cases}$$

which yields

$$T(m,q,r) \in O\left(\frac{m^3}{r}\right).$$

Note that the inequality $q > 1$ in the recurrence is equivalent to $m > n^{1/3}$. Each PE in $\mathsf{M}(n^2)$ requires $S(n, n^2)$ space, where $S(m, q)$ denotes the space required by a PE for executing a recursive call of N-MM with parameters m and q. $S(m, q)$ is upper bounded by the following equation:

$$S(m,q) \leq \begin{cases} S\left(\frac{m}{2}, \frac{q}{8}\right) + O\left(\frac{m^2}{q}\right) & \text{if } q > 1 \\ O\left(m^2\right) & \text{if } q \leq 1 \end{cases}$$

from which it follows that

$$S(m,q) \in O\left(\frac{m^2}{q^{2/3}}\right).$$

Each $\mathsf{M}(p, B)$ processor requires $O\left(S(n, n^2)n^2/p\right)$ space, hence the algorithm incurs a $O\left(n^{2/3}\right)$ memory blow-up. □

N-MM solves the eight recursive subproblems $M_{hk\ell}$ in parallel: the parallelism decreases the communication complexity of the algorithm, but it causes a non-constant memory blow-up. By solving the subproblems in two rounds, during which only four subproblems are solved in parallel, it is possible to define a network-oblivious algorithm with a constant memory blow-up, but communication complexity $\Theta\left(n^2/B\sqrt{p}\right)$. The algorithm, which we denote with *SN-MM* (*Succinct Network-oblivious Matrix Multiplication*), is optimal for Theorem 4.6 because each processor holds at most $\Theta\left(n^2/p\right)$ entries of the input and output matrices.

SN-MM is defined in $\mathsf{M}(n^2)$ and computes $C + A \cdot B$, where A, B and C are three $n \times n$ matrices. Suppose the three matrices to be distributes among the PEs as in N-MM. As before, we refer to a quadrant of a matrix E ($E \in \{A, B, C\}$) as E_{hk}, with $h, k \in \{0, 1\}$. The algorithm is based on the following simple recursive strategy, where parameters m and q are employed with the same meaning as before (initially, $m = n$ and $q = n^2$):

1. Regard the q PEs as partitioned into *four* segments S_{hk}, with $h, k \in \{0, 1\}$, of $q/4$ PEs each. (Note that each S_{hk} is an $\mathsf{M}(q/4)$ machine.)
2. Redistribute the inputs so that the entries of A_{h0} and B_{0h} are evenly spread among the PEs in S_{hh}, and the entries of A_{h1} and B_{1k} evenly spread among the PEs in S_{hk}, for $h, k \in \{0, 1\}$ and $h \neq k$.
3. For $h, k \in \{0, 1\}$ and $h \neq k$ in parallel, compute recursively $C_{hh} + A_{h0} \cdot B_{0h}$ within S_{hh}. and $C_{hk} + A_{h1} \cdot B_{1k}$ within S_{hk}.
4. Redistribute the inputs so that the entries of A_{h1} and B_{1h} are evenly spread among the PEs in S_{hh}, and the entries of A_{h0} and B_{0k} evenly spread among the PEs in S_{hk}, for $h, k \in \{0, 1\}$ and $h \neq k$.
5. For $h, k \in \{0, 1\}$ and $h \neq k$ in parallel, compute recursively $C_{hh} + A_{h1} \cdot B_{1h}$ within S_{hh}. and $C_{hk} + A_{h0} \cdot B_{0k}$ within S_{hk}.
6. Redistribute the inputs so that the entries of A_{hk} and B_{hk} are evenly spread among the PEs in S_{hk}, for $h, k \in \{0, 1\}$.

Since the subproblem size m^2 and the number of assigned PEs decrease by a factor four, it follows that in each recursive call with input size m there are m^2 PEs. The recurrence stops when a subproblem is solved by an unique PE, that is, when the subproblem size is one: in this case, $C + A \cdot B$ is computed locally. SN-MM is equivalent to the D-BSP algorithm given in Theorem 2.5 when $P = n^2$, where P denotes the number of D-BSP processors.

Theorem 4.8. *The communication and computation complexities of the network-oblivious algorithm SN-MM for the n-MM problem, when executed on an* $\mathsf{M}(p, B)$ *machine, with* $1 < p \leq n^2$ *and* $1 \leq B \leq n^2/p$, *are*

$$H_{\mathrm{SN-MM}}(n, p, B) \in \Theta\left(\frac{n^2}{B\sqrt{p}}\right), \tag{4.5}$$

$$T_{\mathrm{SN-MM}}(n, p, B) \in \Theta\left(\frac{n^3}{p}\right), \tag{4.6}$$

which are optimal for all values of p and B in the specified ranges. The algorithm incurs a $O(\log n)$ *memory blow-up per processor.*

Proof. Observe that any recursive call with input size m is solved by $q = m^2$ PEs. Let $H(m, r)$ denote the communication complexity of a recursive call to SN-MM, where m and r denote the input size and the number of assigned $\mathsf{M}(p, B)$ processors, respectively. It follows that:

$$H(m,r) \leq \begin{cases} 2H\left(\frac{m}{2}, \frac{r}{4}\right) + O\left(\frac{m^2}{Br}\right) & \text{if } r > 1 \\ 0 & \text{if } r \leq 1 \end{cases}$$

which yields

$$H(m,r) \in O\left(\frac{m^2}{B\sqrt{r}}\right).$$

Equation 4.5 follows because $H_{\text{SN-MM}}(n, p, B) = H(n, p)$. Let $T(m, r)$ denote the computation complexity of a recursive call to SN-MM, where m and r are defined as usual.

$$T(m,r) \leq \begin{cases} 2T\left(\frac{m}{2}, \frac{r}{4}\right) + O\left(\frac{m^2}{r}\right) & \text{if } r > 1 \\ 8T\left(\frac{m}{2}, 1\right) + O\left(m^2\right) & \text{if } r \leq 1 \text{ and } m > 1 \\ O\left(1\right) & \text{if } r \leq 1 \text{ and } m \leq 1 \end{cases}$$

which yields

$$T(m,r) \in O\left(\frac{m^3}{q}\right).$$

Since the input and output matrices are evenly distributed among the PEs and their entries are not replicated, the optimality of the communication complexity follows from Equation 4.2 of Theorem 4.6. The memory blow-up is due to the machine stack size of each PE, which has size $O(\log n)$ because each PE performs $\log n$ recursive and nested calls to SN-MM. The execution of SN-MM on an $\mathsf{M}(p, B)$ exhibits the same memory blow-up. □

The next lemma states that SN-MM can be transformed into an iterative network-oblivious algorithm without increasing its communication and computation complexities.

Lemma 4.9. *The network-oblivious algorithm SN-MM can be transformed into an iterative network-oblivious algorithm. Its communication and computation complexities on an* $\mathsf{M}(p, B)$*, with* $1 < p \leq n^2$ *and* $1 \leq B \leq n^2/p$*, do not change, while no* *additional space for stack is required.*

Proof. We show that the stack of a PE can be simulated by a constant number of words. In a recursive call of SN-MM, each PE must know the size of the subproblem and on which submatrices of A, B and C is working. However, the size can be represented in a global variable m which is divided by two before a recursive call and multiplied by two when a call finishes. Furthermore, the submatrices of A, B and C on which a PE is working is uniquely determined by the size of the subproblem and the PE's ID.

Thus, the stack is required only for storing the return address of a recursive call. However, in SN-MM there are only two possible return addresses (Lines 3 and 5), thus only one bit is sufficient for storing them. Since there are at most $\log n$ nested calls, it follows that the machine stack can be simulated by $\log n$ bits. Since on an $\mathsf{M}(n^2)$ there are n^2 PEs, we can suppose the $\log n$ bits to be stored in a constant number of words. □

The additional memory due to the stack is also required by the network-oblivious algorithms for FFT and sorting, but we will not discus this issue further since the approach used for SN-MM applies to these algorithms as well.

We now apply Theorem 4.5 to show that N-MM and SN-MM are both optimal in a D-BSP. (Remember that the D-BSP model assumes that both the B_i's and the ratios g_i/B_i are non increasing.)

Corollary 4.10. *SN-MM and N-MM perform optimally on a* $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ *where* $1 < P \leq n^2$ *and* $1 \leq B_i \leq n^2/P$ *for each* $0 \leq i < \log P$*. In particular, the communication time of SN-MM is:*

$$D_{\mathrm{SN-MM}}(n, P, \boldsymbol{g}, \boldsymbol{B}) \in \Theta\left(\frac{n^2}{P}\sum_{i=0}^{\log P-1} 2^{i/2}\frac{g_i}{B_i}\right). \tag{4.7}$$

Proof. Consider an i-superstep s of SN-MM or N-MM and let h^s be the maximum number of words sent or received by a PE. It is easy to see that there exists a set of $q = n^2/2^i$ consecutive numbered PEs where almost all PEs in the first half of $q/2$ PEs send $\Theta(h^s)$ messages to PEs in the second half. Hence, the word-degree of s on an $\mathsf{M}(2^j, 1)$, with $j > i$, is $\Theta(h^s n^2/2^j)$ and algorithms SN-MM and N-MM are (α, n^2)-wise, for a suitable constant α. Since the i-granularity of SN-MM is $\Theta(n^2/2^i)$, the corollary follows from Theorem 4.5. Equation 4.7 can be derived as Equation 2.3 in Theorem 2.5. □

In conclusion, N-MM attains a better communication time than SN-MM, however the former requires a $O\left(n^{2/3}\right)$ memory blow-up. If a constant memory blow-up is desired, the (iterative) SN-MM algorithm must be used; however, communication time will increase [ITT04]. For the optimality of SN-MM in the D-BSP, it follows that Equation 4.7 provides also a lower bound on the communication time for the n-MM problem on a $\mathsf{D\text{-}BSP}(P, \mathbf{g}, \mathbf{B})$ where $B_i \leq n^2/P$ for each i, with $0 \leq i < \log P$.

4.2.2 Matrix transposition

The *n-MT problem* consists of transposing an $n \times n$ matrix. To completely specify the problem in the parallel setting, we require that, initially (resp., finally), the entries of the matrix are evenly distributed among the available PEs according to a row-major (resp., column-major) ordering. While the problem is trivially solved on any $\mathsf{M}(p, 1)$ machine, as we will see, it becomes harder for larger block sizes. The following theorem establishes a lower bound on the communication complexity of the n-MT problem.

Theorem 4.11. *Let $\mathcal{A}$ be an algorithm solving the n-MT problem on an $\mathsf{M}(p, B)$, with $1 < p \leq n^2$ and $1 \leq B \leq n^2/p$. The communication complexity of the algorithm is*

$$\Omega\left(\frac{n^2}{Bp}\left(1 + \frac{\log(\min\{(n^2/p), p\})}{\log(1 + n^2/(Bp))}\right)\right).$$

Proof. We use an argument similar to the one employed in [AV88] (see also Section 3.2) to bound from below the I/O complexity of transposition in the EM model. For $0 \leq i < p$, we define the *i-th target group* as the set of entries that will be in processor p_i at the end of the algorithm. Let H be the communication complexity of the algorithm and $H' \leq Hp$ be the overall number of blocks exchanged by the processors during the entire execution. Let us index the blocks communicated among the processors from 1 to H', so that the indices assigned to blocks communicated in one superstep are smaller than those assigned to blocks communicated in any subsequent superstep. For $0 \leq t \leq H'$, define $x_{i,j}(t)$ as the number of entries of the i-th target group held by p_j after block t has been communicated ($x_{i,j}(0)$ reflects the initial condition). We define the *potential of $\mathcal{A}$* after the block of index t has been communicated as

$$\mathrm{POT}(t) = \sum_{i=0}^{p-1}\sum_{j=0}^{p-1} f(x_{i,j}(t)),$$

where $f(x) = x \log x$, for $x > 0$, and $f(0) = 0$. We now bound POT(0): if $p \geq n$, each processor contains one entry for each of n^2/p target groups, hence $POT(0) = 0$; if $p < n$, each processor contains n/p rows of the input matrix, hence it contains n^2/p^2

entries of p target groups and $POT(0) \leq n^2\log(n^2/p^2)$. It follows that $\mathrm{POT}(0) \leq n^2\log(\lceil (n/p)^2 \rceil)$ for each $1 \leq p \leq n^2$; clearly, $\mathrm{POT}(H') = n^2\log(n^2/p)$. By reasoning as in [AV88], it can be shown that, for a suitable constant $c > 0$, the block of index t increases the potential by the quantity[3]

$$\mathrm{POT}(t) - \mathrm{POT}(t-1) \leq cB\log\left(1 + \frac{n^2}{Bp}\right) \stackrel{\text{def}}{=} \nabla\mathrm{POT}. \tag{4.8}$$

Therefore,

$$p \cdot H \cdot \nabla\mathrm{POT} \geq \mathrm{POT}(H') - \mathrm{POT}(0) = n^2\log\left(\frac{n^2}{p}\right) - n^2\log\left(\left\lceil\left(\frac{n}{p}\right)^2\right\rceil\right),$$

and the theorem follows. □

We now describe a network-oblivious algorithm, named *N-MT*, for the n-MT problem on $\mathsf{M}(n^2)$. For a nonnegative integer i, let $\mathcal{B}(i)$ denote the binary representation of i, and let $\mathcal{B}^{-1}(\cdot)$ be such that $\mathcal{B}^{-1}(\mathcal{B}(i)) = i$. Given two binary strings $u = (u_{d-1}\dots u_0)$ and $v = (v_{d-1}\dots v_0)$ we let $u \bowtie v$ denote their bitwise interleaving, that is, $u \bowtie v = u_{d-1}v_{d-1}\dots u_0v_0$. Let A be the $n \times n$ input matrix and let $P(i,j)$ denote the processing element PE_{in+j}, which initially holds $A[i,j]$ and at the end will hold $A^T[i,j]$, with $0 \leq i,j < n$. The algorithm consists of a 1-superstep followed by a 0-superstep.

1. For $0 \leq i,j < n$, $P(i,j)$ sends $A[i,j]$ to the PE_q, where $q = \mathcal{B}^{-1}(\mathcal{B}(i) \bowtie \mathcal{B}(j))$;

2. For $0 \leq q < n^2$, if the PE_q has received entry $A[i,j]$ in the previous substep, then it forwards it to $P(j,i)$.

We observe that the first superstep rearranges matrix entries according the Z-Morton permutation defined in [CLPT02]. The correctness of the algorithm is evident. The communication complexity of the above algorithm, whose correctness is immediate, is established in the following theorem.

Theorem 4.12. *The communication and computation complexities of the network-oblivious algorithm N-MT when executed on an $\mathsf{M}(p,B)$ machine, with $1 < p \leq n^2$*

[3]Equation 4.8 can, loosely speaking, be derived from Lemma 3.1 by setting $M = n^2/p$ and $W = B$.

and $1 \leq B \leq n/\sqrt{p}$, *are*

$$H_{\text{N-MT}}(n,p,B) \in \Theta\left(\frac{n^2}{Bp}\right) \tag{4.9}$$

$$T_{\text{N-MT}}(n,p,B) \in \Theta\left(\frac{n^2}{p}\right) \tag{4.10}$$

respectively, which are optimal for the specified ranges of p *and* B.

Proof. Suppose B, p and n are powers of two and $1 \leq B \leq n/\sqrt{p}$. Let B_k, with $0 \leq k < n^2/B$, denotes the k-th segment of B consecutive PEs of $\mathsf{M}(n^2)$, that is, $\text{PE}_{kB} \dots \text{PE}_{(k+1)B-1}$. Note that PEs within the same B_k hold A's entries that are in the same row and belong to B different columns whose addresses differ in the $\log B$ least significant bits, assuming $B \leq n/\sqrt{p}$. Hence, in the first superstep PEs in every B_k send their entries to PEs belonging to a segment of size at most $B^2/2 \leq n^2/p$. In the second superstep, $P(j,i)$ receives an entry from PE_q, where $q = \mathcal{B}^{-1}(\mathcal{B}(i) \bowtie \mathcal{B}(j))$; thus PEs in every B_k receives all their data from PEs belonging to a segment of size B^2. Therefore, since a segment of $B^2 \leq 2n^2/p$ PEs is contained in at most four $\mathsf{M}(p,B)$ processors, the block-degree of the communication involved in each superstep is $O\left(n^2/(Bp)\right)$. Optimality follows from Theorem 4.11. If B, n or p are not powers of two, then the algorithm and the proof work by considering the nearest powers of two (which are a multiplicative factor two apart). $\square$

The following corollary is a consequence of Theorems 4.5 and 4.12:

Corollary 4.13. *The network-oblivious algorithm N-MT performs optimally on a* $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ *where* $1 < P \leq n^2$ *and* $1 \leq B_i \leq n/\sqrt{p}$, *for each* $0 \leq i < \log P$.

Proof. It is easy to see that in each i-superstep, with $i \in \{0,1\}$ and for each $i < j < \log n^2$, there exists a segment of $\Theta\left(n^2/2^j\right)$ consecutive numbered PEs each communicating the maximum amount of words (that is, $O\left(n^2/2^j\right)$) for that superstep to PEs outside the segment; then the algorithm is (α, n^2)-wise. From the proof of Theorem 4.12, it follows that two communicating processors of $\mathsf{M}(2^i, B)$, with $0 < i \leq \log n^2$ and $B \leq n/2^{i/2}$, exchange at least $\Omega\left(B\right)$ words; hence, the i-granularity is $\Theta\left(n/2^{i/2}\right)$. The corollary follows from Theorem 4.5. $\square$

4.2.3 Impossibility result for matrix transposition

The constraint on B in the above theorem is what we call *small-block assumption* and is reminiscent of the tall-cache assumption made in the context of cache-oblivious algorithms [FLPR99]. We will now prove that, under reasonable constraints, the small-block assumption is necessary to achieve network-oblivious optimality for the n-MT problem, just as the tall-cache assumption was shown to be necessary to achieve cache-oblivious optimality for n-MT (see Section 3). We say that an $\mathsf{M}(p, B)$ algorithm is *full* if in each of its supersteps all processors send/receive the same number of blocks, within constants, and each block contains $\Theta(B)$ data words.

Theorem 4.14. *There cannot exist a network-oblivious algorithm $\mathcal{A}$ for the n-MT problem such that, for every $1 < p \leq n^2$ and $1 \leq B \leq n^2/p$, its execution on $\mathsf{M}(p, B)$ yields a full algorithm whose communication complexity matches the one stated in Theorem 4.11.*

Proof. Assume that such a network-oblivious algorithm $\mathcal{A}$ exists, and let H_1 and H_2 be the communication complexities of $\mathcal{A}$ when executed on $\mathsf{M}(p_1, B_1)$ and $\mathsf{M}(p_2, B_2)$, with $p_1 > p_2$. Since the two executions are full by hypothesis, and every data communicated in the $\mathsf{M}(p_2, B_2)$ execution must be also communicated in the $\mathsf{M}(p_1, B_1)$ execution, we must have that $B_1 p_1 H_1 \in \Omega(B_2 p_2 H_2)$, whence $B_1 p_1 H_1/(B_2 p_2 H_2) \in \Omega(1)$ (the asymptotic is with reference to n). Now, let us choose $p_1 = n^2/2$, $B_1 = 1$, $p_2 = \Theta(n^{2-\epsilon})$, for an arbitrary constant $0 < \epsilon < 1$, and $B_2 \in \Theta(n^2/p_2)$. Then, by Theorem 4.11 we have that $(B_1 p_1 H_1)/(B_2 p_2 H_2) \in \Theta(1/\log n) = o(1)$, a contradiction. □

Next, we prove that the n-MT lower bound can always be matched by parameter-aware full algorithms, hence the impossibility stated above stems from requiring network-obliviousness.

Theorem 4.15. *For every $1 < p \leq n^2$ and $1 \leq B \leq n^2/p$, there exists a full $\mathsf{M}(p, B)$ algorithm for the n-MT problem whose communication complexity matches the one stated in Theorem 4.11.*

Proof. The algorithm is obtained by suitably parallelizing the strategy of [AV88] employed for solving the n-MT problem in the EM model. In each processor, the $\Theta(n^2/p)$ records are partitioned into different target groups; each group in the decomposition is called a *target subgroup.* Before the start of the algorithm, the size of each target subgroup is $\lceil (n/p)^2 \rceil$.

The algorithm uses a merging procedure. The records in the same target subgroup remain together throughout the course of the algorithm. In each superstep, target subgroups are merged and become bigger. The algorithm terminates when each target subgroup is complete, that is, when each target subgroup has size $\Theta(n^2/p)$. In each superstep, which has block-degree $\Theta(n^2/(Bp))$, the size of each target subgroup increases by the multiplicative factor $n^2/(Bp)$. Hence, the number i of supersteps required by the algorithm is given by:

$$\left\lceil \frac{n}{p} \right\rceil^2 \left(\frac{n^2}{Bp} \right)^i \geq \frac{n^2}{p}.$$

Since the block-degree of each round is $\Theta\left(n^2/(Bp)\right)$ the theorem follows.

As an example, consider the case when $B = \Theta\left(n^2/p\right)$. The algorithm divides the input matrix into n^2/k^2 submatrices of size $k \times k$, with $k = \min\{n, n^2/p\}$. Since the matrix is distributed according with a row-major layout, each processor contains $\lceil n/p \rceil$ rows of a submatrix. Then, in parallel, processors assigned to a submatrix transpose it by means of a binary merge; after that, each processor completely contains one target group and, in the final superstep, the target groups are moved in the right final destinations. It is easy to see that the algorithm requires

$$\Theta\left(\log \frac{k}{\lceil n/p \rceil} \right) = \Theta\left(\log \min\{n^2/p, p\} \right)$$

supersteps, whose block-degree is $O(1)$. □

4.2.4 FFT

The *n-FFT problem* consists of computing an n-input FFT dag. The following theorem establishes a lower bound on the communication complexity of any $\mathsf{M}(p, B)$ algorithm for this problem.

Theorem 4.16. *Let $\mathcal{A}$ be any algorithm solving the n-FFT problem on an $\mathsf{M}(p, B)$ with $1 < p \leq n$ and $B \geq 1$. If the word-degree of each superstep is $\Theta(n/p)$, then the communication complexity of $\mathcal{A}$ is:*

$$\Omega\left(\frac{n}{Bp} \frac{\log n}{\log(1 + n/p)} \right).$$

Proof. Observe that the *or* of n bits can be computed by means of the FFT dag. Hence, the lower bound given in [Goo99, Theorem 4.2] on the number of supersteps for computing the *or* of n bits on an $\mathsf{M}(p, 1)$ translates into a lower bound for the n-FFT problem. Since the word-degree of each superstep is $\Theta(n/p)$, the theorem follows. □

The network-oblivious n-FFT algorithm, called *N-FFT*, on $\mathsf{M}(n)$ exploits the well-known recursive decomposition of the dag into two sets of $\sqrt{n}$-input FFT subdags, with each set containing $\sqrt{n}$ such dags [ACS87]. Inputs are initially distributed one per PE in such a way that the inputs of the j-th subdag in the first set are assigned to the j-th segment of $\sqrt{n}$ consecutively numbered PEs (i.e., an $\mathsf{M}(\sqrt{n})$ machine). The outputs of the first set of subdags are permuted to become the inputs of the second set, where the permutation pattern is equivalent to the matrix transposition of an $\sqrt{n} \times \sqrt{n}$ matrix (i.e., $\sqrt{n}$-MT); thus, the permutation can be performed with the network-oblivious algorithm N-MT. We have:

Theorem 4.17. *The communication and computation complexities of the network-oblivious algorithm N-FFT, when executed on an $\mathsf{M}(p, B)$ machine, with $1 < p \leq n$ and $1 \leq B \leq \sqrt{n/p}$, are*

$$H_{\text{N-FFT}}(n, p, B) \in \Theta\left(\frac{n}{Bp}\frac{\log n}{\log(1 + n/p)}\right) \tag{4.11}$$

$$T_{\text{N-FFT}}(n, p, B) \in \Theta\left(\frac{n}{p}\log n\right) \tag{4.12}$$

which are optimal for the specified ranges of p and B.

Proof. When the algorithm is run on an $\mathsf{M}(p, B)$ machine with $p \leq \sqrt{n}$, each subdag is computed locally by a single processor, and in this case, we must account only for the transposition step, which entails each processor sending and receiving $O(n/(Bp))$ blocks for Theorem 4.12.

Each recursive call of the algorithm with input size m is solved by m PEs, and let r be the number of $\mathsf{M}(p, B)$ processors that simulate these PEs. Since in each recursive level, the m PEs perform a matrix transposition, each processor has block-degree $O(m/(Br))$ and the communication complexity of the recursive call, $H(m, r)$ obeys the subsequent recurrence (for simplicity B is omitted from $H(m, r)$):

$$H(m, r) \leq \begin{cases} 2H\left(\sqrt{m}, \frac{r}{\sqrt{m}}\right) + O\left(\frac{m}{Br}\right) & \text{if } r > 1 \\ 0 & \text{if } r \leq 1 \end{cases}$$

which yields

$$H(m, r) \in O\left(\frac{m}{Br}\frac{\log m}{\log(1 + m/r)}\right).$$

Equation 4.11 is obtained by setting $m = n$ and $r = p$, and its optimality follows from Theorem 4.16 since the word-degree of an $\mathsf{M}(p, B)$ processor is $\Theta(n/p)$. The computation complexity, $T(m, r)$, of the recursive call is instead given by the following recurrence:

$$T(m,r) \le \begin{cases} 2T\left(\sqrt{m}, \frac{r}{\sqrt{n}}\right) + O\left(\frac{m}{r}\right) & \text{if } r > 1 \\ 2\sqrt{m}T\left(\sqrt{m}, 1\right) + O(m) & \text{if } r \le 1 \text{ and } m > 1 \\ O(1) & \text{if } r \le 1 \text{ and } m \le 1 \end{cases}$$

and we have

$$T(m,r) \in O\left(\frac{m}{r}\log m\right)$$

Equation 4.12 is obtained by setting $m = n$ and $r = p$, and is optimal because an n-input FFT dag contains $\Theta(n \log n)$ nodes. □

Observe that the small-block assumption is needed to guarantee an $O(n/(Bp))$ communication complexity for the matrix transposition problem within the algorithm. It is an interesting open problem to determine to what extent such an assumption is really needed in the n-FFT case. A similar question regarding the tall-cache assumption is open in the realm of cache-obliviousness.

The following theorem is a consequence of Theorems 4.5 and 4.17:

Corollary 4.18. *The network-oblivious algorithm N-FFT performs optimally on a D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$ where $1 < P \le n$ and $1 \le B_i \le \sqrt{n/p}$ for each $0 \le i < \log P$.*

Proof. Note that N-FFT's communications are due to matrix transposition. By Corollary 4.13, it follows that N-FFT is (α, n)-wise[4] and its i-granularity is $\Theta\left(n/2^{i/2}\right)$. The corollary follows from Theorem 4.5. □

4.2.5 Sorting

The *n-Sort problem* consists of sorting n keys. We require that the inputs are evenly distributed among the PEs, and that, at the end, the keys held by the i-th PE are all smaller than or equal to those held by the j-th PE, for every $j > i$. The following theorem establishes a lower bound on the communication complexity of any $\mathsf{M}(p, B)$ algorithm for this problem.

Theorem 4.19. *Let $\mathcal{A}$ be an algorithm solving the n-Sort problem on an $\mathsf{M}(p, B)$, with $1 < p \le n$ and $B \ge 1$. If the word-degree of each superstep is $\Theta(n/p)$, then the*

[4]The algorithm given by the concatenation of supersteps of (α, P)-wise algorithms is still (α, P)-wise.

communication complexity of $\mathcal{A}$ is:

$$\Omega\left(\frac{n}{Bp}\frac{\log n}{\log(1+n/p)}\right)$$

Proof. The theorem follows by dividing by B the lower bound for $\mathsf{M}(p,1)$ given in [Goo99] □

We now describe a network-oblivious algorithm, called *N-Sort*, for n-Sort based on a recursive version of the *Columnsort* algorithm as described in [Lei92]. We regard both the n keys and the PEs of $\mathsf{M}(n)$ as arranged on a $s \times r$ matrix with $s = n^{1/3}$ and $r = n^{2/3}$. The algorithm consists of seven *phases* numbered from 1 to 7. During Phases 1, 3, 5 and 7 the keys in each row are sorted recursively (except in Phase 5 where adjacent rows are sorted in reverse order) by r consecutive PEs (i.e., an $\mathsf{M}(n^{2/3})$ machine). During Phase 2 (resp., 4) a transposition (resp., reverse transposition) of the $s \times r$ matrix is performed maintaining the $s \times r$ shape. In Phase 6 two steps of odd-even transposition sort are applied to each column.

Theorem 4.20. *The communication and computation complexities of the network-oblivious algorithm N-Sort, when executed on an $\mathsf{M}(p,B)$ machine with $1 < p \leq n$ and $1 \leq B \leq \sqrt{n/p}$ are*

$$H_{\text{N-Sort}}(n,p,B) \in O\left(\frac{n}{Bp}\left(\frac{\log n}{\log(1+n/p)}\right)^{\log_{3/2} 4}\right),$$
$$T_{\text{N-Sort}}(n,p,B) \in O\left(\frac{n}{Bp}\left(\log n\right)^{\log_{3/2} 4}\right).$$

The communication complexity is optimal when $p \leq n^{1-\epsilon}$, for any constant ϵ with $0 < \epsilon < 1$.

Proof. The transposition performed in Phases 2 and 4 can be implemented by separately transposing in parallel the r/s submatrices of size $s \times s$ and then suitably permuting the rows of the $s \times r$ matrix. By employing the network-oblivious algorithm described in Subsection 4.2.2 for each submatrix and by using the stated upper bound on B, the transposition of the $s \times r$ matrix has communication complexity $\Theta\left(n/(Bp)\right)$. The stated communication and computation complexities of the entire algorithm is obtained by solving the following recurrences, where $H(m,\tau)$ and $T(m,\tau)$ are the communication and computation complexities of a recursive call with input size m and solved by τ processors of $\mathsf{M}(p, B)$.

$$H(m,\tau) \leq \begin{cases} 4H\left(m^{2/3}, \frac{\tau}{m^{1/3}}\right) + O\left(\frac{m}{B\tau}\right) & \text{if } \tau > 1 \\ 0 & \text{if } v \leq 1 \end{cases}$$
$$\in O\left(\frac{m}{B\tau}\left(\frac{\log m}{\log(1+m/\tau)}\right)^{\log_{3/2} 4}\right)$$

$$T(m,\tau) \leq \begin{cases} 4T\left(m^{2/3}, \frac{\tau}{m^{1/3}}\right) + O\left(\frac{m}{\tau}\right) & \text{if } \tau > 1 \\ 4m^{1/3}T\left(m^{2/3}, 1\right) + O\left(m\right) & \text{if } m > 1 \text{ and } \tau > 1 \\ O\left(1\right) & \text{if } m \leq 1 \text{ and } \tau \leq 1 \end{cases}$$
$$\in O\left(\frac{m}{r}\left(\log m\right)^{\log_{3/2} 4}\right).$$

Observe that when executed on an $\mathsf{M}(p, B)$ the network-oblivious algorithm N-Sort has the property that the word-degree of each superstep is $\Theta\left(n/p\right)$, hence the lower bound given in Theorem 4.19 applies to it. □

We conjecture that a similar result can be obtained by adapting other known sorting algorithms such as, for example, the one in [Goo99].

The following theorem is a consequence of Theorems 4.5 and 4.20:

Corollary 4.21. *The above network-oblivious algorithm for the n-Sort problem performs optimally on a $\mathsf{D\text{-}BSP}(P, \boldsymbol{g}, \boldsymbol{B})$ where $P \leq n^{1-\epsilon}$ and $1 \leq B_i \leq \sqrt{n/p}$ for each $0 \leq i < \log P$.*

Proof. The proof is similar to the one for Corollary 4.18. □

Chapter 5

Network-Oblivious Algorithms for the Gaussian Elimination Paradigm

An algorithm must be seen to be believed.
(Donald Knuth)

In Chapter 4, we described the network-oblivious framework and proposed some network-oblivious algorithms for fundamental problems, namely matrix multiplication and transposition, FFT and sorting. In this chapter we present network-oblivious algorithms which implement the *Gaussian Elimination Paradigm* (*GEP*) [CR06]. Prominent problems, like Floyd-Warshall's all-pairs shortest paths, Gaussian Elimination without pivoting, LU decomposition, and matrix multiplication, can be solved through this paradigm.

Specifically, we propose three algorithms, named *N-GEP*, *ϵN-GEP* and *PN-GEP*, which exploit the recursive structure of parallel and cache-oblivious implementations of GEP introduced in [CR07, CR08]. The three algorithms exhibit optimal communication and computation complexities in the evaluation model for suitable values of the processor number and communication block size. Furthermore, N-GEP yields optimal performance also in the D-BSP model for certain ranges of the parameters.

The rest of the chapter is organized as follows. In Section 5.1, we define GEP and the two parallel implementations given in [CR07, CR08]. Then, in Section 5.2, we describe and analyze our network-oblivious algorithms.

```
INPUT: $n \times n$ matrix $x$, function $f : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \times \mathcal{S} \to \mathcal{S}$, set $\Sigma_f$ of triplets
    $\langle i, j, k \rangle$, with $0 \leq i, j, k < n$.
OUTPUT: transformation of $x$ defined by $f$ and $\Sigma_f$.
1: for $k \leftarrow 0$ to $n-1$ do
2:   for $i \leftarrow 0$ to $n-1$ do
3:     for $j \leftarrow 0$ to $n-1$ do
4:       if $\langle i, j, k \rangle \in \Sigma_f$ then
5:         $x[i,j] \leftarrow f(x[i,j], x[i,k], x[k,j], x[k,k])$;
```

Figure 5.1: Gaussian Elimination Paradigm (GEP).

5.1 Preliminaries

5.1.1 Definition of GEP

Let x be an $n \times n$ matrix with entries chosen from an arbitrary domain $\mathcal{S}$, and let $f : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ be an arbitrary function, which we call *update function*. (For simplicity, we assume n to be a power of two). By *Gaussian Elimination Paradigm* (*GEP*) we refer to the computation defined by the pseudocode in Figure 5.1 where the algorithm modifies x by applying a given set of *updates*, denoted by $\langle i, j, k \rangle$, with $i, j, k \in [0, n)$, of the form

$$x[i,j] \leftarrow f(x[i,j], x[i,k], x[k,j], x[k,k]).$$

We let Σ_f denote the set of such updates that the algorithm needs to perform. We refer to Σ_f and n as *update set* and *input size*, respectively. We suppose that the inclusion check in Line 4 and function f are computed in constant time and without indirect addressing (for avoiding unexpected cache misses). Observe that an entry of x can be used many times and assume different values as a result of updates in Σ_f; by changing the order of the updates, the result of the GEP computation can be different.

As noted in [CR06] and illustrated in Figure 5.2, many problems can be solved using a GEP computation, including Floyd-Warshall's all-pairs shortest paths, Gaussian Elimination without pivoting, LU decomposition and matrix multiplication.

The following definition will be used for proving the correctness of the network-oblivious algorithms for GEP. We say a GEP computation is *commutative* if its update function f exhibits the following property: for each y, u_1, v_1, w_1, u_2, v_2, and w_2 in $\mathcal{S}$,

$$f(f(y, u_1, v_1, w_1), u_2, v_2, w_2) = f(f(y, u_2, v_2, w_2), u_1, v_1, w_1).$$

```
INPUT: $n \times n$ matrix $x$.
OUTPUT: Gaussian elimination without pivoting of $x$.
1: for $k \leftarrow 0$ to $n-1$ do
2:    for $i \leftarrow 0$ to $n-1$ do
3:       for $j \leftarrow 0$ to $n-1$ do
4:          if $(k < n-2) \wedge (k < i < n-1) \wedge (k < j)$ then
5:             $x[i,j] \leftarrow x[i,j] - \frac{x[i,k]}{x[k,k]} x[k,j]$;
```

```
INPUT: $n \times n$ matrix $x$.
OUTPUT: the all-pairs shortest paths of an $n$-node graph defined by the adja-
    cency matrix $x$.
1: for $k \leftarrow 0$ to $n-1$ do
2:    for $i \leftarrow 0$ to $n-1$ do
3:       for $j \leftarrow 0$ to $n-1$ do
4:          $x[i,j] \leftarrow x[i,j] \min(x[i,k] + x[k,j])$;
```

```
INPUT: three $n \times n$ matrices $a$, $b$ and $c$.
OUTPUT: $c \leftarrow a \cdot b$.
1: for $k \leftarrow 0$ to $n-1$ do
2:    for $i \leftarrow 0$ to $n-1$ do
3:       for $j \leftarrow 0$ to $n-1$ do
4:          $c[i,j] \leftarrow c[i,j] + a[i,k] \cdot b[k,j]$;
Remark: the computation requires three matrices, but it can easily transformed
into a GEP computation by considering one $n \times n$ matrix that encloses $a$, $b$ and $c$.
```

Figure 5.2: Examples of GEP computations. From the top: Gaussian elimination without pivoting, Floyd-Warshall's all-pairs shortest paths, matrix multiplication with semiring operations.

Not all GEP computations are commutative, however all of the instances of GEP for the aforementioned problems can be easily seen to be commutative.

5.1.2 Previous cache-oblivious and parallel implementations of GEP

Chowdhury and Ramachandran proposed in [CR06] a cache-oblivious recursive implementation of the GEP paradigm, called *I-GEP*. It executes updates in Σ_f in an order different than the one induced by the algorithm in Figure 5.1. Updates are performed in I-GEP in such a way to minimize cache misses and increase implicit parallelism. In general, changing the order of the updates may yield a different output, thus affecting the correctness of the computation. In fact, it is proved in [Cho07] that I-GEP produces the correct output under certain conditions on f and Σ_f, which are met by all notable instances mentioned before. Furthermore, in [CR07] the authors present an extension of I-GEP, referred to as C-GEP, which implements correctly any instance of GEP with no degradation in performance. We refer to the original papers for more details. The cache complexity of I-GEP is $O\left(n^3/B\sqrt{M}\right)$ when executed on an $\mathsf{IC}(M,B)$, which is optimal in the worst-case because it matches the lower bound on the cache complexity of matrix multiplication with semiring operations [HK81].

Below we describe two parallel implementations of I-GEP, which will be at the base of our network-oblivious algorithms. These two versions, which we name *PI-GEP1* and *PI-GEP2*, have been presented in [CR07] and [CR08], respectively, for a CREW (concurrent read, exclusive write) shared-memory model composed of p processors and one level of cache, which can be either shared or distributed[1]. In this model parallel tasks are assigned to processors by a scheduler. PI-GEP1 and PI-GEP2 correctly solve any GEP computation that is correctly solved by I-GEP, however they can be extended to fully generality by adopting the ideas in C-GEP.

PI-GEP1, whose pseudocode is reproduced in Appendix B for completeness, adopts a recursive strategy where in each call it performs all updates

$$x[i,j] \leftarrow f(x[i,j], x[i,k], x[k,j], x[k,k])$$

such that $\langle i,j,k\rangle \in \Sigma_f \cap (I \times J \times K)$, where $I = [i_0, i_1]$, $J = [j_0, j_1]$ and $K = [k_0, k_1]$ are suitable subranges of the full range $[0,n)$. For the initial call we have $I = J = K = [0,n)$. Put in another way, we can imagine that the four operands of f come, in order, from the following four submatrices of x, some of which may coincide:

- $X \equiv x[I,J]$,
- $U \equiv x[I,K]$,
- $V \equiv x[K,J]$,
- $W \equiv x[K,K]$.

There are four different types of recursive calls which are implemented by four recursive functions $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. Function $\mathcal{A}$ is invoked when $I = J = K$, hence $X = U = V = W$. Function $\mathcal{B}$ is invoked when $I = K$ and $I \cap J = \emptyset$, hence $X = V$ and $U = W$. Function $\mathcal{C}$ is invoked when $J = K$ and $I \cap J = \emptyset$, hence $X = U$ and $V = W$. Finally, function $\mathcal{D}$ is invoked when both $I \cap K = \emptyset$ and $J \cap K = \emptyset$, hence all four submatrices X, U, V and W are disjoint. The initial call is $\mathcal{A}(x,n)$. In each function, the four submatrices X, U, V and W are partitioned into four quadrants referred to, with obvious notation, as $X_{i,j}$, $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$, with $0 \le i,j \le 1$, and the eight subproblems they induce are solved recursively (some in parallel). It is

[1]In [CR07] and [CR08], both PI-GEP1 and PI-GEP2 are referred to as I-GEP.

Algorithm (constraints on p and B)	Communication complexity on $\mathsf{M}(p,B)$	Computation complexity on $\mathsf{M}(p,B)$	Optimality on $\mathsf{M}(p,B)$	Optimality on $\mathsf{D\text{-}BSP}(P,\mathbf{g},\mathbf{B})$
N-GEP ($p \leq n^2/\log^2 n$, $B \geq 1$)	$O\left(\frac{n^2}{B\sqrt{p}} + n\log^2 n\right)$	$\Theta\left(\frac{n^3}{p}\right)$	when $p \leq n^2/\log^4 n$ and $B \leq n/(\sqrt{p}\log^2 n)$	when $P \leq n/\log n$ and $B_i \in O\left(\frac{n2^{i/2}}{P\log n}\right)$
ϵN-GEP with ϵ arbitrary constant in $(0,1)$ ($p \leq n^2/\log^{4/3}$, $B \leq (n/\sqrt{p})^{1+\epsilon}$)	$O\left(\frac{n^2}{B\sqrt{p}}\log^2\frac{\log n}{\log(n^2/p)}\right)$	$O\left(\frac{n^3}{p}\log^2\frac{\log n}{\log(n^2/p)}\right)$	when $p \leq n^{2-\delta}$, and $B \leq (n/\sqrt{p})^{1+\epsilon}$, with δ an arbitrary constant in $(0,2)$	open problem
PN-GEP ($p \leq n^2$, $B \geq 1$)	$O\left(\frac{n^2}{B\sqrt{p}} + n\right)$	$\Theta\left(\frac{n^3}{p}\right)$	when $p \leq n^2$ and $B \leq n/\sqrt{p}$	no

Figure 5.3: Properties of network-oblivious algorithms described in this chapter.

shown in [CR07] that the algorithm requires $O\left(n^3/p + n\log^2 n\right)$ computational time and the number of misses performed by all p processors matches the $O\left(n^3/B\sqrt{M}\right)$ upper bound of the sequential implementation.

PI-GEP2 is an improved version of [CR08] which achieves optimal computational time $\Theta\left(n^3/p + n\right)$, without increasing the number of misses. Its pseudocode is given in Appendix B. The implementation still uses the four functions $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ specified before. However, in any invocation of one of these functions, each of the submatrices X, U, V and W, of size, say, $m \times m$, is partitioned into r^2 smaller submatrices of size $m/r \times m/r$, which are denoted, with obvious notation, by $X_{i,j}$, $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$, with $0 \leq i,j < r$. These smaller submatrices induce r^3 subproblems which are solved in r rounds. Parameter r is a suitable value chosen as a function of m. When $r = 2$, PI-GEP2 coincides with PI-GEP1.

5.2 Network-Oblivious Algorithms

In this section we propose three network-oblivious algorithms, N-GEP, ϵN-GEP and PN-GEP, for performing GEP computations. They exhibit optimal communication and computation complexities on an $\mathsf{M}(p,B)$ for wide ranges of the parameters and are based on the shared-memory implementations by [CR07, CR08], briefly described in the previous section. Results are summarized in the table in Figure 5.3.

N-GEP is a recursive algorithm, based on PI-GEP1, which exhibits optimal communication and computation complexities on an $\mathsf{M}(p, B)$ when $p \leq n^2/\log^4 n$ and $B \leq n/(\sqrt{p}\log^2 n)$. It also exhibits optimal performance in the D-BSP model for certain ranges of the parameters. ϵN-GEP, which is built on PI-GEP2, is an algorithm parametric in the constant $\epsilon \in (0, 1)$. It has optimal communication and computation complexities when $p \leq n^{2-\delta}$ and $B \leq (n/\sqrt{p})^{1+\epsilon}$, where δ is an arbitrary constant in $(0, 2)$. Thus, ϵN-GEP exploits large communication blocks more efficiently than N-GEP. Furthermore, ϵN-GEP is interesting since, when ϵ is increased by a constant factor, it achieves optimality for larger communication block sizes, while its communication and computation complexities increase only by a constant factor. When ϵ tends to one the algorithm becomes N-GEP. We do not analyze its performance on the D-BSP model. Finally, PN-GEP is a fast algorithm derived by ϵN-GEP by setting $\epsilon = 0$. It exhibits optimal communication and computation complexities on an $\mathsf{M}(p, B)$ when $p \leq n^2$ and $B \leq n/\sqrt{p}$. Although PN-GEP exhibits optimality in the evaluation model, it is not optimal on a D-BSP.

The section is organized as follows: in Section 5.2.1 we describe N-GEP, in Section 5.2.2 ϵN-GEP, and in Section 5.2.3 PN-GEP.

5.2.1 N-GEP

Algorithm Specification

N-GEP (*Network-oblivious GEP*) is built on PI-GEP1 [CR07], from which it inherits the recursive structure, and is designed for an $\mathsf{M}(n^2/\log^2 n)$. (The number of PEs in the specification model reflects the critical pathlength of PI-GEP1.) Its pseudocode is given in Figures 5.4, 5.5, 5.6 and 5.7. Here, the construct **sync** indicates the global synchronization at the end of a superstep (superstep labels are not reported for simplicity) and the assignment $L_2 \leftarrow L_1$, with L_1 and L_2 matrices of equal dimension, involves the copy of each entry of L_1 into the corresponding entry of L_2 and is achieved by means of a suitable communication among the PEs.

N-GEP consists of four functions, $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}^*$: the first three functions are suitable adaptations of their counterparts in PI-GEP1, while $\mathcal{D}^*$ is based on PI-GEP1's $\mathcal{D}$ but solves subproblems in a different order and is equivalent to $\mathcal{D}$ when the GEP computation is commutative. The initial call is $\mathcal{A}(x, n, \mathcal{P}, n^2/\log^2 n)$, where x is the $n \times n$ matrix on which the GEP computation has to be performed and $\mathcal{P} = \{PE_0, \ldots, PE_{n^2/\log^2 n-1}\}$ denotes the set of all the $\mathsf{M}(n^2/\log^2 n)$ PEs.

The differences between N-GEP and PI-GEP1 are a consequence of the different models for which they have been designed. Indeed, PI-GEP1 is built on a shared-memory model featuring concurrent reads and where computation is mapped on processors by a scheduler. On the other hand, N-GEP is defined in a distributed-memory model (i.e., the specification model) where PEs communicate in a point-to-point fashion and computation has to be explicitly partitioned among the PEs.

Each function receives as inputs at most four $m \times m$ matrices (i.e., X, U, V and W) and the set $\mathcal{P}$ containing the q consecutive numbered PEs assigned to the function. We assume each of the four input matrices to be distributed according with a row-major layout among $\min\{q, m^2\}$ PEs, evenly chosen from the q PEs of $\mathcal{P}$. We note that the algorithm does not guarantee the number q of assigned PEs to be small than the number m^2 of entries in a matrix.

When $m = 1$ or $q = 1$, each function solves the problem sequentially through I-GEP [CR06]. Observe that the second base case (i.e, $q = 1$), which is not present in PI-GEP1, is required since computation has to be explicitly mapped on PEs by the algorithm (this base case is not required in N-GEP's $\mathcal{A}$ because $q \geq m$ all the times).

When $m > 1$ and $q > 1$, each input matrix is split into four $m/2 \times m/2$ quadrants as in PI-GEP1, and then eight subproblems are solved recursively through calls to $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}^*$: the order in which subproblems are solved and the parallelism among them depend on the function and is shown in the pseudocode. The order in which subproblems are solved in $\mathcal{D}^*$ differs from the one used in PI-GEP1's $\mathcal{D}$: hence, the two functions in general are not equivalent, but $\mathcal{D}^*$ guarantees a constant memory blow-up, which would not be obtained by PI-GEP1's $\mathcal{D}$.

Each subproblem is solved by q/k consecutive numbered PEs of $\mathcal{P}$, where k, with $k \in \{1, 2, 4\}$, denotes the number of subproblems which are solved concurrently. In functions $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, inputs of the eight subproblems are stored in new matrices of size $m/2 \times m/2$ allocated among the q PEs of $\mathcal{P}$; hence $O\left(\lceil m^2/q \rceil\right)$ new words per PE are allocated in each invocation. In contrast, in $\mathcal{D}^*$ no new space is required per PE.

The following theorem shows that N-GEP can correctly solve a wide class of GEP computations.

Theorem 5.1. *The network-oblivious algorithm N-GEP performs correctly any commutative GEP computation which is correctly solved by PI-GEP1, and each PE exhibits a constant memory blow-up.*

Proof. When a GEP computation is commutative, updates in PI-GEP1's $\mathcal{D}$ can be performed in any order since U, V and W are fixed in $\mathcal{D}$. Then, it can be proved by induction that N-GEP's $\mathcal{D}^*$ is equivalent to PI-GEP1's $\mathcal{D}$. As a consequence, $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are also equivalent to their respective implementations in PI-GEP1. The first part of the theorem follows.

```
$\mathcal{A}(X, m, \mathcal{P}, q)$
INPUT: matrix $X \equiv x[I, I]$ with $I = [i_0, i_1] \subseteq [0, n)$ and $m = i_1 - i_0 + 1$; segment $\mathcal{P}$ of $q$
    consecutive numbered PEs.
OUTPUT: execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times I \times I)$.
 1: if $T = \emptyset$ then return;
 2: if $m = 1$ then
 3:     $X[0,0] \leftarrow f(X[0,0], X[0,0], X[0,0], X[0,0])$; // $X$ is a $1 \times 1$ matrix.
 4: else
 5:     Let $X_{i,j}$, with $0 \leq i, j \leq 1$, be the four quadrants of $X$;
 6:     Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be the partition of $\mathcal{P}$ where each set contains $q/2$ consecutive numbered
        PEs;
 7:     Allocate space for eight $m/2 \times m/2$ matrices denoted as $\tilde{X}$, $\tilde{X}_0$, $\tilde{X}_1$, $U$, $U'$, $V$, $V'$
        and $W$, which are allocated as follows: $U'$ and $\tilde{X}_0$ (resp., $V'$ and $\tilde{X}_1$) are distributed
        according with a row-major layout among the first $\min\{q/2, m^2/4\}$ PEs of $\mathcal{P}_0$ (resp.,
        $\mathcal{P}_1$); $\tilde{X}$, $U$, $V$ and $W$ are distributed according with a row-major layout among the
        first $\min\{q, m^2/4\}$ PEs of $\mathcal{P}$;
 8:     $\tilde{X} \leftarrow X_{0,0}$; sync;
 9:     $\mathcal{A}(\tilde{X}, m/2, \mathcal{P}, q)$;
10:     $X_{0,0} \leftarrow \tilde{X}$; sync;
11:     $\tilde{X}_0 \leftarrow X_{0,1}$, $\tilde{X}_1 \leftarrow X_{1,0}$, $U' \leftarrow X_{0,0}$, $V' \leftarrow X_{0,0}$; sync;
12:     In parallel invoke $\mathcal{B}(\tilde{X}_0, U', m/2, \mathcal{P}_0, q/2)$ and $\mathcal{C}(\tilde{X}_0, V', m/2, \mathcal{P}_1, q/2)$; sync;
13:     $X_{0,1} \leftarrow \tilde{X}_0$, $X_{1,0} \leftarrow \tilde{X}_1$; sync;
14:     $\tilde{X} \leftarrow X_{1,1}$, $U \leftarrow X_{1,0}$, $V \leftarrow X_{0,1}$, $W \leftarrow X_{0,0}$; sync;
15:     $\mathcal{D}^*(\tilde{X}, U, V, W, m/2, \mathcal{P}, q)$;
16:     $X_{1,1} \leftarrow \tilde{X}$; sync;
17:     $\tilde{X} \leftarrow X_{1,1}$; sync;
18:     $\mathcal{A}(\tilde{X}, m/2, \mathcal{P}, q)$;
19:     $X_{1,1} \leftarrow \tilde{X}$; sync;
20:     $\tilde{X}_0 \leftarrow X_{1,0}$, $\tilde{X}_1 \leftarrow X_{0,1}$, $U' \leftarrow X_{1,1}$, $V' \leftarrow X_{1,1}$; sync;
21:     In parallel invoke $\mathcal{B}(\tilde{X}_0, U', m/2, \mathcal{P}_0, q/2)$ and $\mathcal{C}(\tilde{X}_1, V', m/2, \mathcal{P}_1, q/2)$; sync;
22:     $X_{1,0} \leftarrow \tilde{X}_0$, $X_{0,1} \leftarrow \tilde{X}_1$; sync;
23:     $\tilde{X} \leftarrow X_{0,0}$, $U \leftarrow X_{0,1}$, $V \leftarrow X_{1,0}$, $W \leftarrow X_{1,1}$; sync;
24:     $\mathcal{D}^*(\tilde{X}, U, V, W, m/2, \mathcal{P}, q)$;
25:     $X_{0,0} \leftarrow \tilde{X}$; sync;
26:     Delete the eight temporary matrices;
```

Figure 5.4: Function $\mathcal{A}$ of N-GEP.

```
$\mathcal{B}(X, U, m, \mathcal{P}, q)$
**INPUT:** matrices $X \equiv x[I, J]$ and $U \equiv x[I, I]$, with $I = [i_0, i_1] \subseteq [0, n)$, $J = [j_0, j_1] \subseteq [0, n)$, $I \cap J = \emptyset$ and $m = i_1 - i_0 + 1 = j_1 - j_0 + 1$; segment $\mathcal{P}$ of $q$ consecutive numbered PEs.
**OUTPUT:** execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times I)$.
 1: **if** $T = \emptyset$ **then return**;
 2: **if** $m = 1$ or $q = 1$ **then**
 3:     Solve the problem sequentially with I-GEP;
 4: **else**
 5:     Let $X_{i,j}$ and $U_{i,j}$, with $0 \leq i, j \leq 1$, be the four quadrants of $X$ and $U$;
 6:     Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be the partition of $\mathcal{P}$ where each set contains $q/2$ consecutive numbered PEs;
 7:     Allocate space for eight $m/2 \times m/2$ matrices, denoted as $\tilde{X}_i$, $U_i$, $V_i$, and $W_i$ for $0 \leq i \leq 1$, in such a way that $\tilde{X}_i$, $U_i$, $V_i$ and $W_i$ are distributed according with a row-major layout among the first $\min\{q/2, m^2/4\}$ PEs of $\mathcal{P}_i$;
 8:     $\tilde{X}_i \leftarrow X_{0,i}$, $U_i \leftarrow U_{0,0}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
 9:     In parallel invoke $\mathcal{B}(\tilde{X}_i, U_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
10:     $X_{0,i} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
11:     $\tilde{X}_i \leftarrow X_{1,i}$, $U_i \leftarrow U_{1,0}$, $V_i \leftarrow X_{0,i}$, $W_i \leftarrow U_{0,0}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
12:     In parallel invoke $\mathcal{D}^*(\tilde{X}_i, U_i, V_i, W_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
13:     $X_{1,i} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
14:     $\tilde{X}_i \leftarrow X_{1,i}$, $U_i \leftarrow U_{1,1}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
15:     In parallel invoke $\mathcal{B}(\tilde{X}_i, U_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
16:     $X_{1,i} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
17:     $\tilde{X}_i \leftarrow X_{0,i}$, $U_i \leftarrow U_{0,1}$; $V_i \leftarrow X_{1,i}$, $W_i \leftarrow U_{1,1}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
18:     In parallel invoke $\mathcal{D}^*(\tilde{X}_i, U_i, V_i, W_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
19:     $X_{0,i} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
20:     Delete the eight temporary matrices;
```

Figure 5.5: Function $\mathcal{B}$ of N-GEP.

```
$\mathcal{C}(X, V, m, \mathcal{P}, q)$
**INPUT:** matrices $X \equiv x[I, J]$ and $V \equiv x[J, J]$, with $I = [i_0, i_1] \subseteq [0, n)$, $J = [j_0, j_1] \subseteq$
    $[0, n)$, $I \cap J = \emptyset$ and $m = i_1 - i_0 + 1 = j_1 - j_0 + 1$; segment $\mathcal{P}$ of $q$ consecutive numbered
    PEs.
**OUTPUT:** execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times J)$.
 1: **if** $T = \emptyset$ **then return**;
 2: **if** $m = 1$ or $q = 1$ **then**
 3:     Solve the problem sequentially with I-GEP;
 4: **else**
 5:     Let $X_{i,j}$ and $V_{i,j}$, with $0 \leq i, j \leq 1$, be the four quadrants of $X$ and $V$;
 6:     Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be the partition of $\mathcal{P}$ where each set contains $q/2$ consecutive numbered
        PEs;
 7:     Allocate space for eight $m/2 \times m/2$ matrices, denoted as $\tilde{X}_i$, $U_i$, $V_i$, and $W_i$ for
        $0 \leq i \leq 1$, in such a way that $\tilde{X}_i$, $U_i$, $V_i$ and $W_i$ are distributed according with a
        row-major layout among the first $\min\{q/2, m^2/4\}$ PEs of $\mathcal{P}_i$;
 8:     $\tilde{X}_i \leftarrow X_{i,0}$, $V_i \leftarrow V_{0,0}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
 9:     In parallel invoke $\mathcal{C}(\tilde{X}_i, V_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
10:     $X_{i,0} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
11:     $\tilde{X}_i \leftarrow X_{i,1}$, $U_i \leftarrow X_{i,0}$, $V_i \leftarrow V_{0,1}$, $W_i \leftarrow V_{0,0}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
12:     In parallel invoke $\mathcal{D}^*(\tilde{X}_i, U_i, V_i, W_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
13:     $X_{i,1} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
14:     $\tilde{X}_i \leftarrow X_{i,1}$, $V_i \leftarrow V_{1,1}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
15:     In parallel invoke $\mathcal{C}(\tilde{X}_i, U_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
16:     $X_{i,1} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
17:     $\tilde{X}_i \leftarrow X_{i,0}$, $U_i \leftarrow X_{i,1}$; $V_i \leftarrow V_{1,0}$, $W_i \leftarrow V_{1,1}$ for each $i$ with $0 \leq i \leq 1$; **sync**;
18:     In parallel invoke $\mathcal{D}^*(\tilde{X}_i, U_i, V_i, W_i, m/2, \mathcal{P}_i, q/2)$ for each $i$ with $0 \leq i \leq 1$; **sync**;
19:     $X_{i,0} \leftarrow \tilde{X}_i$ for each $i$ with $0 \leq i \leq 1$; **sync**;
20:     Delete the eight temporary matrices;
```

Figure 5.6: Function $\mathcal{C}$ of N-GEP.

```
$\mathcal{D}^*(X, U, V, W, m, \mathcal{P}, q)$
INPUT: matrices $X \equiv x[I,J]$, $U \equiv x[I,K]$, $V \equiv x[K,J]$ and $W \equiv x[K,K]$, with $I = [i_0, i_1] \subseteq [0,n)$, $J = [j_0, j_1] \subseteq [0,n)$, $K = [k_0, k_1] \subseteq [0,n)$, $I \cap K = \emptyset$, $J \cap K = \emptyset$ and $m = i_1 - i_0 + 1 = j_1 - j_0 + 1 = k_1 - k_0 + 1$; segment $\mathcal{P}$ of $q$ consecutive numbered PEs.
OUTPUT: execution of all updates $\langle i,j,k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times K)$.
1: if $T = \emptyset$ then return;
2: if $m = 1$ or $q = 1$ then
3:     Solve the problem sequentially with I-GEP;
4: else
5:     Let $\mathcal{P}_{i,j}$, with $0 \leq i,j \leq 1$, be a partition of $\mathcal{P}$ where each set contains $q/4$ consecutive numbered PEs;
6:     Let $X_{i,j}$, $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$, with $0 \leq i,j \leq 1$, be the four quadrants of $X$, $U$, $V$, $W$;
7:     $W_{0,1} \leftarrow W_{1,1}$, $W_{1,0} \leftarrow W_{0,0}$; sync;
8:     Execute the following data movements:
         • distribute the entries of each of $X_{0,0}$, $U_{0,0}$, $V_{0,0}$ and $W_{0,0}$ among the PEs in $\mathcal{P}_{0,0}$ in row-major;
         • distribute the entries of each of $X_{0,1}$, $U_{0,1}$, $V_{1,1}$ and $W_{0,1}$ among the PEs in $\mathcal{P}_{0,1}$ in row-major;
         • distribute the entries of each of $X_{1,0}$, $U_{1,1}$, $V_{1,0}$ and $W_{1,1}$ among the PEs in $\mathcal{P}_{1,0}$ in row-major;
         • distribute the entries of each of $X_{1,1}$, $U_{1,0}$, $V_{0,1}$ and $W_{1,0}$ among the PEs in $\mathcal{P}_{1,1}$ in row-major;
9:     sync;
10:    In parallel invoke:
       $\mathcal{D}^*(X_{0,0}, U_{0,0}, V_{0,0}, W_{0,0}, m/2, \mathcal{P}_{0,0}, q/4)$, $\mathcal{D}^*(X_{0,1}, U_{0,1}, V_{1,1}, W_{0,1}, m/2, \mathcal{P}_{0,1}, q/4)$,
       $\mathcal{D}^*(X_{1,0}, U_{1,1}, V_{1,0}, W_{1,1}, m/2, \mathcal{P}_{1,0}, q/4)$, $\mathcal{D}^*(X_{1,1}, U_{1,0}, V_{0,1}, W_{1,0}, m/2, \mathcal{P}_{1,1}, q/4)$;
11:    sync;
12:    Execute the following data movements:
         • distribute the entries of each of $X_{0,0}$, $U_{0,1}$, $V_{1,0}$ and $W_{1,1}$ among the PEs in $\mathcal{P}_{0,0}$ in row-major;
         • distribute the entries of each of $X_{0,1}$, $U_{0,0}$, $V_{0,1}$ and $W_{1,0}$ among the PEs in $\mathcal{P}_{0,1}$ in row-major;
         • distribute the entries of each of $X_{1,0}$, $U_{1,0}$, $V_{0,0}$ and $W_{0,0}$ among the PEs in $\mathcal{P}_{1,0}$ in row-major;
         • distribute the entries of each of $X_{1,1}$, $U_{1,1}$, $V_{1,1}$ and $W_{0,1}$ among the PEs in $\mathcal{P}_{1,1}$ in row-major;
13:    sync;
14:    In parallel invoke:
       $\mathcal{D}^*(X_{0,0}, U_{0,1}, V_{1,0}, W_{1,1}, m/2, \mathcal{P}_{0,0}, q/4)$, $\mathcal{D}^*(X_{0,1}, U_{0,0}, V_{0,1}, W_{1,0}, m/2, \mathcal{P}_{0,1}, q/4)$,
       $\mathcal{D}^*(X_{1,0}, U_{1,0}, V_{0,0}, W_{0,0}, m/2, \mathcal{P}_{1,0}, q/4)$, $\mathcal{D}^*(X_{1,1}, U_{1,1}, V_{1,1}, W_{0,1}, m/2, \mathcal{P}_{1,1}, q/4)$;
15:    sync;
16:    Re-establish the initial layout; sync;
```

Figure 5.7: Function $\mathcal{D}^*$ of N-GEP.

Function $\mathcal{D}^*$ does not require additional space. On the other hand, the additional space required by $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ is $S(m,q)$, which is bounded by the following recurrence:

$$S(m,q) \leq \begin{cases} S\left(\frac{m}{2},\frac{q}{2}\right) + O\left(\left\lceil \frac{m^2}{q} \right\rceil\right) & \text{if } m > 1 \text{ and } q > 1 \\ O(1) & \text{otherwise} \end{cases}$$

Hence, we yield:

$$S(m,q) \in O\left(\frac{m^2}{q} + \log\min\{m,q\}\right).$$

Since the algorithm starts with $m = n$ and $q = n^2/\log^2 n$, the additional space per PE is $O\left(\log^2 n\right)$, which is asymptotically optimal because the $n^2/\log^2 n$ PEs must hold the $n \times n$ input matrix x. □

We observe that if subproblems in function $\mathcal{D}^*$ are solved in the same order of PI-GEP1's $\mathcal{D}$ (see Figure B.4), N-GEP can be extended to perform correctly any GEP computation which is correctly solved by I-GEP. However, each PE would exhibit a $O(\log n)$ memory blow-up for replicating quadrants of U and V which are required concurrently by two subproblems. In contrast, the proposed order in $\mathcal{D}^*$ avoids replications and a constant memory blow-up is required.

N-GEP can be extended to correctly implement any commutative GEP computation, without performance degradation, by adopting the ideas in C-GEP. We do not analyze this issue further since it does not provide more interesting results.

Complexity of N-GEP on $\mathsf{M}(p,B)$

Here, we provide upper bounds on the communication and computation complexities of N-GEP when it is executed on an $\mathsf{M}(p,B)$ machine.

Theorem 5.2. *A commutative GEP computation on an $n \times n$ matrix can be performed by N-GEP with communication and computation complexities*

$$H_{\text{N-GEP}}(n,p,B) \in O\left(\frac{n^2}{B\sqrt{p}} + n\log^2 n\right), \tag{5.1}$$

$$T_{\text{N-GEP}}(n,p,B) \in \Theta\left(\frac{n^3}{p}\right). \tag{5.2}$$

on an $\mathsf{M}(p,B)$, for $1 < p \leq n^2/\log^2 n$ and $B \geq 1$.

Proof. It is easily seen that functions $\mathcal{B}$ and $\mathcal{C}$ exhibit the same complexities, hence we will analyze only $\mathcal{B}$. Consider the execution of $\mathcal{T}$, for $\mathcal{T} \in \{\mathcal{A},\mathcal{B},\mathcal{D}^*\}$, with input size m and q assigned PEs. We denote with r, where $r \leq q$, the number of consecutive $\mathsf{M}(p, B)$ processors that simulate the q PEs, and with $H_{\mathcal{T}}(m, r)$ the communication complexity of $\mathcal{T}$. Observe that m^2 can be either smaller or bigger than q. To simplify the notation, we omit B from $H_{\mathcal{T}}(m, r)$. When $\mathcal{T}$ is in a base case or is completely solved by PEs simulated by the same processor, there is no communication, hence $H_{\mathcal{T}}(m, r) = 0$. Otherwise, each processor performs a constant number of supersteps with block-degree $O\left(\lceil m^2/Br \rceil\right)$ for redistributing data.

When $\mathcal{T} = \mathcal{D}^*$, we have:

$$H_{\mathcal{D}^*}(m, r) \leq \begin{cases} 2H_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{r}{4}\right) + O\left(\left\lceil \frac{m^2}{Br} \right\rceil\right) & \text{if } m > 1 \text{ and } r > 1 \\ 0 & \text{if } m \leq 1 \text{ or } r \leq 1 \end{cases}$$

which yields

$$H_{\mathcal{D}^*}(m, r) \in O\left(\left\lceil \frac{m^2}{Br} \right\rceil \min\left\{\sqrt{r}, m\right\}\right). \tag{5.3}$$

The communication complexity of $\mathcal{B}$ is provided by the following recursive relation:

$$H_{\mathcal{B}}(m, r) \leq \begin{cases} 2H_{\mathcal{B}}\left(\frac{m}{2}, \frac{r}{2}\right) + 2H_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{r}{2}\right) + O\left(\left\lceil \frac{m^2}{Br} \right\rceil\right) & \text{if } m > 1 \text{ and } r > 1 \\ 0 & \text{if } m \leq 1 \text{ or } r \leq 1 \end{cases}$$

$$\leq 2^i H_{\mathcal{B}}\left(\frac{m}{2^i}, \frac{r}{2^i}\right) + O\left(\sum_{j=1}^{i} 2^j \left(H_{\mathcal{D}^*}\left(\frac{m}{2^j}, \frac{r}{2^j}\right) + \left\lceil \frac{m^2}{2^j Br} \right\rceil\right)\right).$$

The last two terms in the summation can be upper bounded by Equation 5.3. If $m \geq r$, the recurrence stops when the subproblem is solved by an unique $\mathsf{M}(p, B)$ processor. Hence by setting $i = \log r$, we get

$$H_{\mathcal{B}}(m, r) \in O\left(\sum_{j=1}^{\log r} 2^j \left\lceil \frac{m^2}{2^j Br} \right\rceil \sqrt{\frac{r}{2^j}}\right) \in O\left(\frac{m^2}{B\sqrt{r}} + r\right). \tag{5.4}$$

If $m < r$, the recurrence stops when the subproblem size is one. By setting $i = \log m$, we have

$$H_{\mathcal{B}}(m, r) \in O\left(\sum_{j=1}^{\log\lceil m^2/r \rceil} 2^j \left\lceil \frac{m^2}{2^j Br} \right\rceil \sqrt{\frac{r}{2^j}} + \sum_{j=\log\lceil m^2/r \rceil + 1}^{\log m} 2^j \left\lceil \frac{m^2}{2^j Br} \right\rceil \frac{m}{2^j}\right),$$

which yields

$$H_{\mathcal{B}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}} + m\log m\right). \tag{5.5}$$

Note that the upper bound given by Equation 5.5 does not match the one in Equation 5.4 when $r = \Theta(m)$, and that $H_{\mathcal{B}}(m,r)$ when $m \geq r$ can also be upper bounded by Equation 5.5.

The communication complexity of $\mathcal{A}$ is given by the following recurrence:

$$H_{\mathcal{A}}(m,r) \leq$$
$$\leq \begin{cases} 2H_{\mathcal{A}}\left(\frac{m}{2}, r\right) + 2H_{\mathcal{B}}\left(\frac{m}{2}, \frac{r}{2}\right) + 2H_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{r}{2}\right) + O\left(\left\lceil\frac{m^2}{Br}\right\rceil\right) & \text{if } m > 1 \\ 0 & \text{if } m \leq 1 \end{cases}$$

Observe that the recursion in function $\mathcal{A}$ stops when the subproblem size is one, that is, when $i = \log m$. Since $H_{\mathcal{B}}(m,r)$ and $H_{\mathcal{D}^*}(m,r)$ are both upper bounded by Equation 5.5, we have:

$$H_{\mathcal{A}}(m,r) \in O\left(\sum_{j=1}^{\log m}\left(\frac{m^2}{2^j B\sqrt{r}} + m\log\frac{m}{2^j}\right)\right),$$

from which we get

$$H_{\mathcal{A}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}} + m\log^2 m\right).$$

Since $H_{\mathrm{N-GEP}}(n,p,B) = H_{\mathcal{A}}(n,p)$, Equation 5.1 follows.

Let us denote with $T_{\mathcal{T}}(m,q,r)$ the computation complexity of function $\mathcal{T}$. Reminding that the number q of $\mathsf{M}(n^2/\log^2 n)$ PEs can be bigger that m^2, we note that each PE performs $O\left(\lceil m^2/q\rceil\right)$ operations on local data in each superstep, and therefore each $\mathsf{M}(p,B)$ processor performs $O\left(\lceil m^2/q\rceil(q/r)\right)$ operations per superstep.

The computation complexity of $\mathcal{D}^*$ is upper bounded as follows:

$$T_{\mathcal{D}^*}(m,q,r) \leq \begin{cases} 2T_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{q}{4}, \frac{r}{4}\right) + O\left(\left\lceil\frac{m^2}{q}\right\rceil\frac{q}{r}\right) & \text{if } m > 1 \text{ and } r > 1 \\ 8T_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{q}{4}, 1\right) + O\left(\left\lceil\frac{m^2}{q}\right\rceil q\right) & \text{if } m > 1,\ q > 1 \text{ and } r \leq 1 \\ m^3 & \text{if } m \leq 1 \text{ or } q \leq 1 \end{cases}$$

By unfolding the recurrence, we have:

$$T_{\mathcal{D}^*}(m,q,r) \leq$$
$$\leq \begin{cases} 2^i T_{\mathcal{D}^*}\left(\frac{m}{2^i}, \frac{q}{4^i}, \frac{r}{4^i}\right) + O\left(\frac{m^2}{r}2^i + \frac{q}{r}2^i\right) & \text{if } m>1 \text{ and } r>1 \\ 8^i T_{\mathcal{D}^*}\left(\frac{m}{2^i}, \frac{q}{4^i}, 1\right) + O\left(2^i m^2 + 2^i q\right) & \text{if } m>1,\ q>1 \text{ and } r\leq 1 \\ m^3 & \text{if } m\leq 1 \text{ or } q\leq 1 \end{cases}$$

which yields

$$T_{\mathcal{D}^*}(m,q,r) \in O\left(\frac{m^3}{r} + \frac{mq}{r}\right).$$

The computation complexity of $\mathcal{B}$ is given by the following relation:

$$T_{\mathcal{B}}(m,q,r) \leq$$
$$\leq \begin{cases} 2T_{\mathcal{B}}\left(\frac{m}{2}, \frac{q}{2}, \frac{r}{2}\right) + 2T_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{q}{2}, \frac{r}{2}\right) + O\left(\left\lceil \frac{m^2}{q}\right\rceil \frac{q}{r}\right) & \text{if } m>1 \text{ and } r>1 \\ 4T_{\mathcal{B}}\left(\frac{m}{2}, \frac{q}{2}, 1\right) + 2T_{\mathcal{D}^*}\left(\frac{m}{2}, \frac{q}{2}, 1\right) + O\left(\left\lceil \frac{m^2}{q}\right\rceil q\right) & \text{if } m>1,\ q>1 \text{ and } r\leq 1 \\ m^3 & \text{if } m\leq 1 \text{ or } q\leq 1 \end{cases}$$
$$\leq \begin{cases} 2^i T_{\mathcal{B}}\left(\frac{m}{2^i}, \frac{q}{2^i}, \frac{r}{2^i}\right) + O\left(\sum_{j=1}^{i}\left(\frac{m^3}{2^j r} + \frac{mq}{r}\right)\right) & \text{if } m>1 \text{ and } r>1 \\ 4^i T_{\mathcal{B}}\left(\frac{m}{2^i}, \frac{q}{2^i}, 1\right) + O\left(\sum_{j=1}^{i}\left(\frac{m^3}{2^j} + mq\right)\right) & \text{if } m>1,\ q>1 \text{ and } r\leq 1 \\ m^3 & \text{if } m\leq 1 \text{ or } q\leq 1 \end{cases}$$

Hence,

$$T_{\mathcal{B}}(m,q,r) \in O\left(\frac{m^3}{r} + \frac{mq}{r}\log m\right).$$

The computation complexity of $\mathcal{A}$ is given by the following relation:

$$T_{\mathcal{A}}(m,q,r) \leq$$
$$\leq \begin{cases} 2T_{\mathcal{A}}\left(\frac{m}{2}, q, r\right) + 2T_{\mathcal{B}}\left(\frac{m}{2}, \frac{q}{2}, \frac{r}{2}\right) + 2T_{\mathcal{D}^*}\left(\frac{m}{2}, q, r\right) + O\left(\left\lceil \frac{m^2}{q}\right\rceil \frac{q}{r}\right) & \text{if } m>1 \\ O(1) & \text{if } m\leq 1 \end{cases}$$
$$\in O\left(2^i T_{\mathcal{A}}\left(\frac{m}{2^i r}, q, r\right) + \sum_{i=1}^{\log m}\left(\frac{m^3}{4^j} + \frac{mq}{r}\log m\right)\right)$$

which gives

$$T_{\mathcal{A}}(m,q,r) \in O\left(\frac{m^3}{r} + \frac{mq}{r}\log^2 m\right).$$

Since $T_{\text{N-GEP}}(n,p,B) = T_{\mathcal{A}}(n, n^2/\log^2 n, p)$, Equation 5.2 follows. □

Corollary 5.3. *A commutative GEP computation on an $n \times n$ matrix can be performed by N-GEP with optimal communication and computation complexities when $1 < p \leq n^2/\log^4 n$ and $B \leq n/(\sqrt{p}\log^2 n)$.*

Proof. The n-MM problem (i.e., matrix multiplication with only semiring operations) can be computed by a commutative GEP computation. Hence, lower bounds on communication and computation complexities for matrix multiplication translate into worst-case lower bounds for an algorithm which performs any commutative GEP computation. An algorithm for solving the n-MM problem on $\mathsf{M}(p,B)$ requires $\Omega\left(n^3/p\right)$ operations and $\Omega\left(n^2/B\sqrt{p}\right)$ communications per processor if each one uses $\Theta\left(n^2/p\right)$ words (Theorem 4.6). It follows that N-GEP is optimal when $p \leq n^2/\log^4 n$ and $B \leq n/(\sqrt{p}\log^2 n)$, since each PE of $\mathsf{M}(n^2/\log^2 n)$ uses $\Theta\left(\log^2 n\right)$ space and each $\mathsf{M}(p,B)$ processor simulates $n^2/(p\log^2 n)$ PEs. □

D-BSP

We conclude by showing that, under certain circumstances, N-GEP performs optimally also on a D-BSP. Theorem 4.5 proves that a network-oblivious algorithm which performs optimally on an $\mathsf{M}(p,B)$ for some values of p and B and satisfies some assumptions on the granularity and wiseness of communications exhibits asymptotically optimal communication time also in the D-BSP model for a wide range of parameters. However, this theorem cannot be applied to N-GEP, because it does not satisfy the wiseness assumption. This is due to the fact that, when executing N-GEP on $\mathsf{M}(p,1)$, many supersteps are such that some PEs of $\mathsf{M}(n^2/\log^2 n)$ do not send messages. Nevertheless, the next theorem and corollary show that N-GEP is still optimal in the D-BSP model:

Theorem 5.4. *A commutative GEP computation on an $n \times n$ matrix can be performed by N-GEP with communication and computation times*

$$D_{\text{N-GEP}}(n,P,\boldsymbol{g},\boldsymbol{B}) \in O\left(\sum_{i=0}^{\log P-1}\left(\frac{n^2 2^{\frac{i}{2}}}{B_i P} + n\log n\right) g_i\right) \tag{5.6}$$

$$T_{\text{N-GEP}}(n,P,\boldsymbol{g},\boldsymbol{B}) \in \Theta\left(\frac{n^3}{P}\right) \tag{5.7}$$

on a $\mathsf{D\text{-}BSP}(P,\boldsymbol{g},\boldsymbol{B})$, for $1 < P \leq n^2/\log^2 n$.

Proof. As before, we consider only one among functions $\mathcal{B}$ and $\mathcal{C}$, since they have the same asymptotic complexities. Remember that B_j and g_j denote the block size and the inverse of the bandwidth, respectively, in a j-cluster with $r = P/2^j$ processors. Consider the execution of function $\mathcal{T}$, with $\mathcal{T} \in \{\mathcal{A}, \mathcal{B}, \mathcal{D}^*\}$, in a $\mathsf{D\text{-}BSP}(P, \mathbf{g}, \mathbf{B})$ with input size m and q assigned PEs; let j be the label of the cluster (hence $r = P/2^j$ processors) that simulates the q PEs. We denote with $D_{\mathcal{T}}(m, j)$ the communication time of $\mathcal{T}$. For notational convenience, we leave $\mathbf{B}$ and $\mathbf{g}$ out from $D_{\mathcal{T}}(m, j)$. The $D_{\mathcal{T}}(m, j)$'s are derived by recursive relations similar to ones employed in Theorem 5.2 for the communication complexity; however, different B_i's and g_i's are used in each recursive level.

The communication time of $\mathcal{D}^*$ is given by the following recurrence:

$$D_{\mathcal{D}^*}(m,j) \leq \begin{cases} 2D_{\mathcal{D}^*}\left(\frac{m}{2}, j+2\right) + O\left(\left\lceil \frac{m^2 2^j}{B_j P} \right\rceil g_j\right) & \text{if } m > 1 \text{ and } j < \log P \\ 0 & \text{if } m \leq 1 \text{ or } j \geq \log P \end{cases}$$
$$\in O\left(2^i D_{\mathcal{D}^*}\left(\frac{m}{2^i}, j+2i\right) + \sum_{k=0}^{i-1} 2^k \left(\frac{m^2 2^j}{B_{j+2k} P} + 1\right) g_{j+2k}\right).$$

By observing that the recurrence terminates when $m/2^i \leq 1$ if $m^2 < P/2^j$ and when $j + 2i \geq \log P$ otherwise, we derive the following bound:

$$D_{\mathcal{D}^*}(m,j) \in O\left(\sum_{k=0}^{\log \min\left\{\sqrt{\frac{P}{2^j}}, m\right\} - 1} 2^k \left(\frac{m^2 2^j}{B_{j+2k} P} + 1\right) g_{j+2k}\right),$$

which, by changing the index of the summation, yields:

$$D_{\mathcal{D}^*}(m,j) \in O\left(\sum_{i=j}^{\log \min\left\{P, m^2 2^j\right\} - 2} 2^{\frac{i-j}{2}} \left(\frac{m^2 2^j}{B_i P} + 1\right) g_i\right). \tag{5.8}$$

The communication time of function $\mathcal{B}$ is given by the following recurrence:

$$D_{\mathcal{B}}(m,j) \leq$$
$$\leq \begin{cases} 2D_{\mathcal{B}}\left(\frac{m}{2}, j+1\right) + 2D_{\mathcal{D}^*}\left(\frac{m}{2}, j+1\right) + O\left(\left\lceil \frac{m^2 2^j}{B_j P} \right\rceil g_j\right) & \text{if } m > 1 \text{ and } j < \log P \\ 0 & \text{if } m \leq 1 \text{ or } j \geq \log P \end{cases}$$
$$\in O\left(2^i D_{\mathcal{B}}\left(\frac{m}{2^i}, j+i\right) + \sum_{k=1}^{i} 2^k \left(D_{\mathcal{D}^*}\left(\frac{m}{2^k}, j+k\right) + \left(\frac{m^2 2^j}{2^k B_{j+k-1}} + 1\right) g_{j+k-1}\right)\right).$$

As in the derivation of $D_{\mathcal{D}^*}(m,j)$, we observe that the recurrence ends when $m/2^i \leq 1$ if $m < P/2^j$ and when $j+i \geq \log P$ otherwise, and we get

$$D_{\mathcal{B}}(m,j) \in O\left(\sum_{k=1}^{\log \min\left\{\frac{P}{2^j}, m\right\}} 2^k D_{\mathcal{D}^*}\left(\frac{m}{2^k}, j+k\right) + \left(\frac{m^2 2^j}{B_{j+k-1}} + 2^k\right) g_{j+k-1} \right).$$

Since $D_{\mathcal{D}^*}(m,j)$ is bounded by Equation 5.8, which assumes two different values according with the relative values of m, j and P, we have:

$$D_{\mathcal{B}}(m,j) \in O\left(D_1(m,j) + D_2(m,j) + \sum_{k=0}^{\log \min\left\{\frac{P}{2^j}, m\right\}-1} \left(\frac{m^2 2^j}{B_{j+k} P} + 2^k\right) g_{j+k} \right),$$

where $D_1(m,j)$ includes the contributions of $2^k D_{\mathcal{D}^*}(m/2^k, j+k)$ for those values of k for which $m^2/4^k \geq P/2^{j+k}$, while $D_2(m,j)$ includes the others. Thus, recalling that $P/2^j$ can be bigger than m^2, we have[2]:

$$D_1(m,j) \in O\left(\sum_{k=1}^{\log\left\lceil \frac{m^2 2^j}{P}\right\rceil} \sum_{i=j+k}^{\log P - 1} 2^{\frac{i-j}{2}} \left(\frac{m^2 2^j}{2^{\frac{k}{2}} B_i P} + 2^{\frac{k}{2}}\right) g_i \right),$$

which, after tedious but simple calculations, yields

$$D_1(m,j) \in O\left(\sum_{i=j+1}^{\log P - 1} 2^{\frac{i-j}{2}} \left(\frac{m^2 2^j}{B_i P} + 2^{\frac{i-j}{2}}\right) g_i \right).$$

$D_2(m,j)$ is bounded as follows:

$$D_2(m,j) \in O\left(\sum_{k=1+\log\left\lceil \frac{m^2 2^j}{P}\right\rceil}^{\log m} \sum_{i=j+k}^{j+\log m^2 - k - 1} 2^{\frac{i-j}{2}} \left(\frac{m^2 2^j}{2^{\frac{k}{2}} B_i P} + 2^{\frac{k}{2}}\right) g_i \right)$$

from which we derive:

$$D_2(m,j) \in O\left(\sum_{i=1+j+\log\left\lceil \frac{m^2 2^j}{P}\right\rceil}^{\log \min\left\{P, m^2 2^j\right\}-2} \left(2^{\frac{i-j}{2}} \frac{m^2 2^j}{B_i P} + m\right) g_i \right).$$

[2]For notational convenience we impose $\sum_{j=a}^{b}(\ldots) = 0$ when $a > b$.

Hence, we get:

$$D_{\mathcal{B}}(m,j) \in O\left(\sum_{i=j}^{\log\min\left\{P,m^2 2^j\right\}-1}\left(2^{\frac{i-j}{2}}\frac{m^2 2^j}{B_i P}+m\right)g_i\right). \tag{5.9}$$

Finally, we can derive the communication time of $\mathcal{A}$ from the following recursive relation:

$$\begin{aligned}
&D_{\mathcal{A}}(m,j) \leq \\
&\leq \begin{cases} 2D_{\mathcal{A}}\left(\frac{m}{2},j\right)+2D_{\mathcal{B}}\left(\frac{m}{2},j+1\right)+2D_{\mathcal{D}^*}\left(\frac{m}{2},j\right)+O\left(\left\lceil\frac{m^2 2^j}{B_j P}\right\rceil g_j\right) & \text{if } m>1 \\ 0 & \text{if } m \leq 1\end{cases} \\
&\in O\left(2^i D_{\mathcal{A}}\left(\frac{m}{2^i},j\right)+\sum_{k=1}^{i}2^k\left(D_{\mathcal{B}}\left(\frac{m}{2^k},j+1\right)+D_{\mathcal{D}^*}\left(\frac{m}{2^k},j\right)+\left\lceil\frac{m^2 2^j}{4^k B_j P}\right\rceil g_j\right)\right)
\end{aligned}$$

By observing that the right-hand side of Equation 5.9 gives also an upper bound on $D_{\mathcal{D}^*}(m,j)$, we get:

$$D_{\mathcal{A}}(m,j) \in O\left(\left(\frac{m^2 2^j}{B_j P}+m\right)g_j+\sum_{k=1}^{\log m}\sum_{i=j}^{\log\min\left\{P,\frac{m^2 2^j}{4^k}\right\}-1}\left(2^{\frac{i-j}{2}}\frac{m^2 2^j}{2^k B_i P}+m\right)g_i\right)$$

which, after some calculations, yields:

$$D_{\mathcal{A}}(m,j) \in O\left(\sum_{i=j}^{\min\left\{P,m^2 2^j\right\}-1}\left(2^{\frac{i-j}{2}}\frac{m^2 2^j}{B_i P}+m\log m\right)g_i\right).$$

Since $D_{\text{N-GEP}}(n,P,\mathbf{g},\mathbf{B}) = D_{\mathcal{A}}(n,0)$ and $P \leq n^2$, Equation 5.6 follows.

The computation time of N-GEP corresponds to the computation complexity of N-GEP on $\mathsf{M}(P,1)$, since it is independent of the communication block size. □

Corollary 5.5. *The communication time of N-GEP on a D-BSP(*$P,\boldsymbol{g},\boldsymbol{B}$*) is optimal when* $P \leq n/\log n$ *and* $B_i \in O\left(\frac{n2^{i/2}}{P\log n}\right)$ *for each* $0 \leq i < \log P$.

Proof. Remember that Equation 4.7 bounds from below the communication time required for solving the n-MM problem on a D-BSP where $B_i \leq n^2/P$ and each processor uses $\Theta\left(n^2/P\right)$ space. By setting $P \leq n/\log n$ and $B_i \in O\left(\frac{n2^{i/2}}{P\log n}\right)$ for $0 \leq i < \log P$, Equation 5.6 matches the lower bound for matrix multiplication. The result follows by recalling that the n-MM problem can be solved by a commutative GEP computation, and that N-GEP uses $\Theta(n^2/P)$ space per processor. $\square$

The ranges of D-BSP optimality for N-GEP can be widened. Indeed, if $g_i = g(P/2^i)^\alpha$ and $B_i = b(P/2^i)^\beta$, with α and β constants in $(0,1)$ and $b, g > 0$ arbitrary constants (this is the case of many point-to-point interconnections [BFPP01]), communication optimality is obtained for each $P \leq \left(\frac{n}{b \log n}\right)^{2/(1+2\alpha)}$. Finally, observe that the algorithm exhibits optimal computation time for every $P \leq n^2/\log^2 n$, independently of the values of $\mathbf{B}$ and $\mathbf{g}$.

5.2.2 ϵN-GEP

ϵN-GEP requires a slight different definition of network-oblivious algorithm. We still define it as a sequence of labelled supersteps, but we allow some PEs to skip some supersteps with the following restriction: if PE_j does not perform an i-superstep s, then also all processing elements PE_k, where k shares the i most significant bits with j, do not perform s. In other words, we allow some *i-clusters* to be idle while the others are executing an i-superstep.

Algorithm specification

ϵN-GEP (*ϵ-Network-oblivious GEP*) extends N-GEP as PI-GEP2 [CR08] extends PI-GEP1[CR07]: in each function, the at most four input matrices are divided into $m^{2-2\epsilon}$ submatrices of size $m^\epsilon \times m^\epsilon$, referred as $X_{i,j}$, $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$ for[3] $0 \leq i, j < m^{1-\epsilon}$, and then $m^{3-3\epsilon}$ subproblems are solved through recursive calls. The order in which subproblems are solved is different from the one adopted in PI-GEP2, however ϵN-GEP's $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are completely equivalent to their counterpart in PI-GEP2, while $\mathcal{D}^*$ corresponds to PI-GEP2's $\mathcal{D}$ when the GEP computation is commutative, as in N-GEP. The adopted order is a consequence of the specification model, since it does not feature concurrent reads as in PI-GEP2, and allows the algorithm to yield a constant memory blow-up.

We assume that each input matrix at the beginning and at the end of a recursive call is distributed among the q PEs according with a bit interleaved layout[4]. We

[3] $X_{0,j}$ (resp. $X_{m^{1-\epsilon}-1,j}$) represents the submatrices at the top (resp., bottom) of X, while $X_{i,0}$ (resp., $X_{i,m^{1-\epsilon}-1}$) represents the submatrices at the left (resp., right) of X. Similarly for $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$.

[4] An $n \times n$ matrix E is distributed according with the bit-interleaved layout among r PEs, $\text{PE}_0, \text{PE}_1, \ldots \text{PE}_{r-1}$, of $\mathsf{M}(h(n))$ as follows: E is divided into r submatrices $E_{i,j}$, with $0 \leq i, j < \sqrt{r}$, of size $n/\sqrt{r} \times n/\sqrt{r}$. Each $E_{i,j}$ is assigned to each PE_k where $k = \mathcal{B}^{-1}(\mathcal{B}(i) \bowtie \mathcal{B}(j))$, adopting the notation introduced in Section 4.2.2.

denote by $\mathcal{P}_{i,j}$, with $0 \leq i,j < m^{1-\epsilon}$, the $q/m^{2-2\epsilon}$ (consecutive numbered) PEs that hold $X_{i,j}$, $U_{i,j}$, $V_{i,j}$ and $W_{i,j}$ when the function begins, and with $X^k_{i,j}$, for $0 \leq k < m^{1-\epsilon}$, the value of $X_{i,j}$ when it has been updated by k recursive calls to $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ or $\mathcal{D}^*$ ($X^0_{i,j}$ reflects the initial value of $X_{i,j}$).

Each function, when not in a base case, consists of a number of phases and during each phase PEs in $\mathcal{P}_{i,j}$ computes recursively $X^k_{i,j}$ for some k or are idle. The four functions are defined below (we do not use pseudocode due to the technical difficult of the algorithm).

- **Base case of each function:** $m \leq \log^{2/3}\log n$**.** The problem is solved sequentially by a PE by means of I-GEP [CR06]. After the description of the four functions, it will be simple to see that $q \leq 1$ when $m \leq \log^{2/3}\log n$.

- **Function $\mathcal{A}(X, m, q)$:** It consists of $5m^{1-\epsilon} - 4$ phases during which each $\mathcal{P}_{i,j}$, if not idle, computes $X^k_{i,j}$ for some k, with $0 \leq k < m^{1-\epsilon}$, and sends/receives messages to/from its neighbors (i.e., $\mathcal{P}_{i\pm1,j\pm1}$) through a superstep involving the q PEs. During the t-th phase, with $1 \leq t \leq 5m^{1-\epsilon} - 4$, each $\mathcal{P}_{i,j}$, with $0 \leq i,j < m^{1-\epsilon}$, performs one of the following cases according with the relative values of i, j and t:

  - *Case 1: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 3k+1$ and $i = j = k$.* $\mathcal{P}_{k,k}$ computes recursively $\mathcal{A}(X^{k-1}_{k,k}, m^\epsilon, q/m^{2-2\epsilon})$, and then forwards $X^k_{k,k}$ to $\mathcal{P}_{k\pm1,k}$ and $\mathcal{P}_{k,k\pm1}$.

  - *Case 2: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 3k + 1 + |k - j|$, $i = k$ and $j \neq k$.* Suppose $j > k$: $\mathcal{P}_{k,j}$ receives $X^k_{k,k}$ from $\mathcal{P}_{k,j-1}$ and solves recursively $\mathcal{B}(X^{k-1}_{k,j}, X^k_{k,k}, m^\epsilon, q/m^{2-2\epsilon})$; then, $X^k_{k,j}$ is forwarded to $\mathcal{P}_{k\pm1,j}$ and $X^k_{k,k}$ to $\mathcal{P}_{k,j+1}$ and $\mathcal{P}_{k\pm1,j}$. Case $j < k$ is similar.

  - *Case 3: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 3k + 1 + |k - i|$, $j = k$ and $i \neq k$.* Suppose $i > k$: $\mathcal{P}_{i,k}$ receives $X^k_{k,k}$ from $\mathcal{P}_{i-1,k}$ and solves recursively $\mathcal{C}(X^{k-1}_{i,k}, X^k_{k,k}, m^\epsilon, q/m^{2-2\epsilon})$; then, $X^k_{i,k}$ is forwarded to $\mathcal{P}_{i,k\pm1}$ and $X^k_{k,k}$ to $\mathcal{P}_{i+1,k}$. Case $i < k$ is similar.

  - *Case 4: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 3k + 1 + |k - i| + |k - j|$ and $i, j \neq k$.* Suppose $i, j > k$; $\mathcal{P}_{i,j}$ receives $X^k_{k,k}$ and $X^k_{k,j}$ from $\mathcal{P}_{i-1,j}$ and $X^k_{i,k}$ from $\mathcal{P}_{i,j-1}$; then it computes recursively $D(X^{k-1}_{i,j}, X^k_{i,k}, X^k_{k,j}, X^k_{k,k})$ and forwards $X^k_{k,j}$ and $X^k_{k,k}$ to $\mathcal{P}_{i,j+1}$, and $X^k_{i,k}$ to $\mathcal{P}_{i+1,j}$. Cases $i, j < k$, $i < k < j$ and $j < k < i$ are similar.

  - *Case 5: $\mathcal{P}_{i,j}$ does not belong to one of the above cases.* $\mathcal{P}_{i,j}$ is idle until the t-th phase ends.

- **Function $\mathcal{B}(X, U, m, q)$:** It consists of $3m^{1-\epsilon} + \log m^{1-\epsilon} - 2$ phases. In each phase, $\mathcal{P}_{i,0}, \ldots \mathcal{P}_{i,m^{1-\epsilon}-1}$ compute in parallel and recursively $X^k_{i,0}, \ldots, X^k_{i,m^{1-\epsilon}-1}$ for some k, with $0 \leq k < m^{1-\epsilon}$, using $U_{i,k}$, and send/receive messages through a superstep involving all q PEs. Note that $U_{i,k}$ is required in $m^{1-\epsilon}$ subproblems concurrently, hence it must be replicated $m^{1-\epsilon}$ times. During the t-th phase, with $t > \log m^{1-\epsilon}$, each $\mathcal{P}_{i,j}$, with $0 \leq i, j < m^{1-\epsilon}$, performs one of the following cases according with the relative values of i, j and t:

  - *Case 1: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 2k + 1 + \log m^{1-\epsilon}$ and $i = k$.* Each $\mathcal{P}_{k,j}$, with $0 \leq j < m^{1-\epsilon}$, computes recursively $\mathcal{B}(X^{k-1}_{k,j}, U_{k,k}, m^{\epsilon}, q/m^{2-2\epsilon})$ and forward $X^k_{k,j}$ and $U_{k,k}$ to $\mathcal{P}_{k\pm1,j}$.
  - *Case 2: there exists an integer k, with $0 \leq k < m^{1-\epsilon}$, such that $t = 2k + 1 + \log m^{1-\epsilon} + |k - i|$ is an integer and $i \neq k$.* Suppose $i > k$: each $\mathcal{P}_{i,j}$, with $0 \leq j < m^{1-\epsilon}$, receives $X^k_{k,j}$ and $U_{k,k}$ from $\mathcal{P}_{i-1,j}$ and computes recursively $\mathcal{D}^*(X^{k-1}_{i,j}, U_{i,k}, X^k_{k,j}, U_{k,k})$; finally, $X^k_{k,j}$ and $U_{k,k}$ are forwarded to $\mathcal{P}_{i+1,j}$. The case $i < k$ is similar.
  - *Case 3: $\mathcal{P}_{i,j}$ does not belong to one of the above cases.* $\mathcal{P}_{i,j}$ is idle until the t-th phase ends.

  Each $\mathcal{P}_{i,j}$ performs also the replications described below. $U_{i,k}$, with $0 \leq i, k < m^{1-\epsilon}$, must be in $\mathcal{P}_{i,j}$ for every $0 \leq j < m^{1-\epsilon}$ at the beginning of the $2k + 1 + \log m^{1-\epsilon} + |k - i|$-th phase. The replication of $U_{i,k}$ is performed by the $m^{1-\epsilon}$ $\mathcal{P}_{i,j}$ starting from the $2k + 1 + |k - i|$-th phase using a binary tree. Thus, $\mathcal{P}_{i,0}, \ldots \mathcal{P}_{i,m^{1-\epsilon}-1}$ start a new replication every two phases and perform at most $(\log m^{1-\epsilon})/2$ concurrent replications. If the copies are not evenly distributed, some $\mathcal{P}_{i,j}$ could exhibit a non constant memory blow-up. However, it is not difficult to see that there are no more than $O(m^{1-\epsilon})$ copies in $\mathcal{P}_{i,0}, \ldots \mathcal{P}_{i,m^{1-\epsilon}-1}$ at any time and, if copies are evenly distributed among the $\mathcal{P}_{i,j}$, each PE incurs a constant memory blow-up.

  In order to keep the memory blow-up constant, we impose that $\mathcal{P}_{k,j}$, with $0 \leq j < m^{1-\epsilon}$, contains only $X^{k-1}_{k,j}$ and $U_{k,k}$ when the call $\mathcal{B}(X^{k-1}_{k,j}, U_{k,k}, m^{\epsilon}, q/m^{1-\epsilon})$ is invoked during the $2k + 1 + \log m^{1-\epsilon}$-th phase; if $\mathcal{P}_{k,j}$ holds other variables (e.g., for replications), then they are moved to $\mathcal{P}_{(k+1) \bmod m^{1-\epsilon}, j}$ before the call to $\mathcal{B}$, and then bring back after it.

- **Function $\mathcal{C}(X, V, m, q)$:** It is similar to function $\mathcal{B}$, but the roles of U and V and of rows and columns are inverted.

- **Function $\mathcal{D}^*(X, U, V, W, m, q)$:** It consists of $m^{1-\epsilon}$ phases. In the t-th one, $\mathcal{P}_{i,j}$ computes recursively $D(X_{i,j}^{k-1}, U_{i,k}, V_{k,j}, W_{k,k})$ where $k = (i + j + t - 1) \mod m^{1-\epsilon}$. Each submatrix of X, U and V is used once in each phase and they are moved to the respective $\mathcal{P}_{i,j}$ through a permutation. However, $W_{k,k}$ is used $m^{1-\epsilon}$ times in each phase: since only entries on the left-to-right diagonal of W are used, each $W_{k,k}$ can be replicated $m^{1-\epsilon}$ times at the beginning of function $\mathcal{D}^*$ by setting $W_{k,j} = W_{k,k}$ for each $0 \leq j < m^{1-\epsilon}$.

Theorem 5.6. *The network-oblivious algorithm ϵN-GEP performs correctly any commutative GEP computation that is correctly solved by PI-GEP2, and each PE exhibits a constant memory blow-up.*

Proof. Let us prove the first part of the theorem by showing that ϵN-GEP, when the GEP computation is commutative, is equivalent to the PI-GEP2 implementation shown in Figure B.5 with $r = m^{1-\epsilon}$. The base cases ($m \leq \log^{2/3} \log n$) are obviously correct, then we assume $m > \log^{2/3} \log n$.

- **Function $\mathcal{D}^*$:** When the GEP computation is commutative, the order by which subproblems are scheduled in $\mathcal{D}^*$ does not matter since U, V and the left-to-right diagonal of W do not change within function $\mathcal{D}^*$. Hence ϵN-GEP's $\mathcal{D}^*$ gives the same result of PI-GEP2's $\mathcal{D}$.

- **Function $\mathcal{B}$:** Let us define

$$\tau(i, k) = \log m^{1-\epsilon} + 2k + 1 + |k - i|.$$

  Since a replication requires $\log m^{1-\epsilon}$ phases, $U_{i,k}$ is in $\mathcal{P}_{i,j}$ for every j with $0 \leq j < m^{1-\epsilon}$ at the beginning of the $\tau(i, k)$-th phase. By an inductive argument on k, it can be proved that $X_{i,j}^k$ is solved by $\mathcal{P}_{i,j}$ during the $\tau(i, k)$-th phase.

  Denote by $\tilde{X}_{i,j}^k$ the value of $X_{i,j}$ in PI-GEP2 when it has been updated by k recursive calls to $\mathcal{B}$ or $\mathcal{D}$ ($\tilde{X}_{i,j}^0$ reflects the initial value of $X_{i,j}$). Remember that ϵN-GEP's $\mathcal{D}^*$ is equivalent to PI-GEP2's $\mathcal{D}$; then, by inductively assuming that ϵN-GEP's $\mathcal{B}$ coincides with PI-GEP2's $\mathcal{B}$ for smaller problems (which is true in the base case), it can be proved that $X_{i,j}^k = \tilde{X}_{i,j}^k$ for each $0 \leq i, j, k < m^{1-\epsilon}$ by induction on k.

- **Function $\mathcal{C}$:** ϵN-GEP's function $\mathcal{C}$ coincides with PI-GEP2's function $\mathcal{C}$, and the proof is a straightforward adaptation of the previous one.

- **Function $\mathcal{A}$:** By an inductive argument on k, it can be proved that $X_{i,j}^k$ is solved by $\mathcal{P}_{i,j}$ during the $\tau(i,j,k)$-th phase, where

$$\tau(i,j,k) = 3(k-1) + |k-i| + |k-j|.$$

  Denote with $\tilde{X}_{i,j}^k$ the value of $X_{i,j}$ in PI-GEP2 when it has been updated by k recursive calls to $\mathcal{A}$, $\mathcal{C}$, $\mathcal{B}$ or $\mathcal{D}$ ($\tilde{X}_{i,j}^0$ reflects the initial value of $X_{i,j}$). Remember that ϵN-GEP's $\mathcal{D}^*$, $\mathcal{B}$ and $\mathcal{C}$ are equivalent to their counterpart in PI-GEP2; then, by inductively assuming that ϵN-GEP's $\mathcal{A}$ coincides with PI-GEP2's $\mathcal{A}$ for smaller problems (which is true in the base case), it can be proved that $X_{i,j}^k = \tilde{X}_{i,j}^k$ for each $0 \leq i,j,k < m^{1-\epsilon}$ by induction on k.

It follows that ϵN-GEP is equivalent to PI-GEP2 for commutative GEP computations. By construction, each PE uses $O\left(\log^{4/3}\log n\right)$ space which is optimal since there are $n^2/\log^{4/3}\log n$ PEs and the input matrix has size $n \times n$. The theorem follows. $\square$

We observe that if subproblems in function $\mathcal{D}^*$ are solved in the same order of PI-GEP2's $\mathcal{D}$, ϵN-GEP can be extended to perform correctly any GEP computation which is correctly solved by I-GEP. However, each PE would exhibit a $O(\log\log n)$ memory blow-up for replicating submatrices of U and V which are required concurrently by many subproblems in $\mathcal{D}$. In contrast, the proposed order in $\mathcal{D}^*$ avoids replications and incurs a constant memory blow-up.

As in N-GEP, ϵN-GEP can be extended to correctly implement any commutative GEP computation, without performance degradation, by adopting the ideas in C-GEP.

Complexity of ϵN-GEP on $\mathsf{M}(p,B)$

Here, we provide upper bounds on the communication and computation complexities of ϵN-GEP when it is executed on an $\mathsf{M}(p,B)$ machine. To this purpose we need the following technical lemma, which is used in the proof of the subsequent theorem for solving some recurrence relations.

Lemma 5.7. *Let $0 < \epsilon < 1$ be a constant and $i \geq 1$ an integer. Then, for $m > 1$, we have:*

$$\sum_{j=1}^{i} \frac{1}{m^{\epsilon^j}} \in \Theta\left(\frac{1}{m^{\epsilon^i}}\right) \qquad \textit{and} \qquad \sum_{j=1}^{i} m^{\epsilon^j} \in \Theta\left(m^{\epsilon}\right).$$

Proof. Let $\ell = \log((1-\epsilon))/\log\epsilon$. If $i < \ell$, the summation has a constant number of terms, hence the theorem follows.

Suppose $i > \ell$. When $1 \le j \le i - \ell$, we get:

$$\frac{m^{\epsilon^j}}{m^{\epsilon^{j+1}}} = m^{\epsilon^j(1-\epsilon)} \Rightarrow m^{\epsilon^j} = m^{\epsilon^j(1-\epsilon)} m^{\epsilon^{j+1}} \ge m^{\epsilon^i} m^{\epsilon^{j+1}}, \tag{5.10}$$

from which follows that $m^{\epsilon^j} \ge (m^{\epsilon^i})^{i-\ell-j} m^{\epsilon^{i-\ell}}$ if $j \le i - \ell$. Therefore we have:

$$\begin{aligned}\sum_{j=1}^{i} \frac{1}{m^{\epsilon^j}} &\le \frac{1}{m^{\epsilon^{i-\ell}}} \sum_{j=1}^{i-\ell} \frac{1}{(m^{\epsilon^i})^{i-\ell-j}} + \sum_{j=i-\ell+1}^{i} \frac{1}{m^{\epsilon^j}} \\ &\le \frac{1}{m^{\epsilon^i}} \sum_{j=0}^{i-\ell-1} \frac{1}{(m^{\epsilon^i})^j} + \ell \frac{1}{m^{\epsilon^i}} \in \Theta\left(\frac{1}{m^{\epsilon^i}}\right),\end{aligned}$$

from which follows the first part of the lemma.

For $j \le i - \ell$, we derive the following inequality from Equation 5.10:

$$m^{\epsilon^{j+1}} \le \frac{m^{\epsilon^j}}{m^{\epsilon^i}} \Rightarrow m^{\epsilon^j} \le \frac{m^{\epsilon}}{(m^{\epsilon^i})^{j-1}}.$$

Thus:

$$\sum_{j=1}^{i} m^{\epsilon^j} \le m^{\epsilon} \sum_{j=1}^{i-\ell} \frac{1}{(m^{\epsilon^i})^{j-1}} + \sum_{j=i-\ell+1}^{i} m^{\epsilon^j} \in \Theta\left(m^{\epsilon}\right),$$

which proves the second part of the lemma. □

Theorem 5.8. *Let ϵ be a constant in $(0, 1)$. A commutative GEP computation on an $n \times n$ matrix can be performed by ϵN-GEP with communication and computation complexities:*

$$H_{\epsilon\text{N}-\text{GEP}}(n, p, B) \in O\left(\frac{n^2}{B\sqrt{p}} \log^2 \frac{\log n}{\log(n^2/p)}\right), \tag{5.11}$$

$$T_{\epsilon\text{N}-\text{GEP}}(n, p, B) \in O\left(\frac{n^3}{p} \log^2 \frac{\log n}{\log(n^2/p)}\right), \tag{5.12}$$

on an $\mathsf{M}(p, B)$, with $1 < p \le n^2/\log^{4/3} \log n$ and $1 \le B \le \left(n/\sqrt{p}\right)^{1+\epsilon}$.

Proof. As before, we consider functions $\mathcal{B}$ and $\mathcal{C}$ equivalent. Consider the execution of $\mathcal{T}$, for $\mathcal{T} \in \{\mathcal{A}, \mathcal{B}, \mathcal{D}^*\}$, with input size m and q assigned PEs. We denote with r, where $r \le q$, the number of consecutive $\mathsf{M}(p, B)$ processors that simulate the q PEs, and with $H_{\mathcal{T}}(m, r)$ the communication complexity of $\mathcal{T}$. For notational convenience, we omit B from $H_{\mathcal{T}}(m, r)$. Since m^2/r equals n^2/p in each recursive call and $B \leq (n/\sqrt{p})^{1+\epsilon}$, we have $B \leq (m/\sqrt{r})^{1+\epsilon}$.

Let us consider function $\mathcal{D}^*$. If $r \leq m^{2-2\epsilon}$, we have that

$$H_{\mathcal{D}^*}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}}\right).$$

Indeed each processor simulates PEs in $\mathcal{P}_{i,j}$ with $i \in [i_0..i_1]$ and $j \in [j_0..j_1]$, where i_0, i_1, j_0, j_1 are suitable values and $i_1 - i_0 = j_1 - j_0 = m^{1-\epsilon}/\sqrt{r}$. Moreover, it is not difficult to see that all PEs in $\mathcal{P}_{i,j}$ send/receive $O(m^{2\epsilon})$ messages to/from PEs in $\mathcal{P}_{(i\pm1) \mod m^{1-\epsilon},(j\pm1) \mod m^{1-\epsilon}}$, during the $O(m^{1-\epsilon})$ supersteps. If $r > m^{2-2\epsilon}$, $H_{\mathcal{D}^*}(m,r)$ is given by the following recurrence:

$$\begin{aligned} H_{\mathcal{D}^*}(m,r) &\leq \begin{cases} m^{1-\epsilon} H_{\mathcal{D}^*}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon}\frac{m^2}{Br}\right) & \text{if } r > m^{2-2\epsilon} \\ O\left(\frac{m^2}{B\sqrt{r}} + m^{1-\epsilon}\right) & \text{if } r \leq m^{2-2\epsilon} \end{cases} \\ &\leq m^{1-\epsilon^i} H_{\mathcal{D}^*}\left(m^{\epsilon^i}, \frac{r}{m^{2-2\epsilon^i}}\right) + O\left(\frac{m^2}{Br}\sum_{j=1}^{i} m^{1-\epsilon^j}\right), \end{aligned}$$

which yields

$$H_{\mathcal{D}^*}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}}\right).$$

Let us consider function $\mathcal{B}$. If $r \leq m^{2-2\epsilon}$, the communication complexity of $\mathcal{B}$ is:

$$H_{\mathcal{B}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}}\right). \tag{5.13}$$

Indeed, replications of U's submatrices require

$$O\left(\left\lceil\frac{m^{2\epsilon}}{B}\right\rceil \frac{m^{1-\epsilon}}{\sqrt{r}} \log r\right)$$

blocks, whilst other

$$O\left(\left\lceil\frac{m^{1+\epsilon}}{B\sqrt{r}}\right\rceil m^{1-\epsilon}\right)$$

blocks are required for communications between adjacent rows. Equation 5.13 follows since $B \leq (m/\sqrt{r})^{1+\epsilon}$. On the other hand, if $r > m^{2-2\epsilon}$, $H_{\mathcal{B}}(m,r)$ is given by the following recurrence:

$$H_{\mathcal{B}}(m,r) \leq$$
$$\leq \begin{cases} m^{1-\epsilon} H_{\mathcal{B}}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon}\left(H_{\mathcal{D}^*}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + \frac{m^2}{Br}\right)\right) & \text{if } r > m^{2-2\epsilon} \\ O\left(\frac{m^2}{B\sqrt{r}} + m^{1-\epsilon}\right) & \text{if } r \leq m^{2-2\epsilon} \end{cases}$$
$$\leq m^{1-\epsilon^i} H_{\mathcal{B}}\left(m^{\epsilon^i}, \frac{r}{m^{2-2\epsilon^i}}\right) + O\left(\frac{m^2}{B\sqrt{r}} i\right).$$

Hence,

$$H_{\mathcal{B}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}} \log_{\epsilon} \frac{\log m^2/r}{\log m}\right).$$

Similarly, if $r \leq m^{2-2\epsilon}$, the communication complexity of $\mathcal{A}$ is

$$H_{\mathcal{A}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}}\right),$$

while, if $r \leq m^{2-2\epsilon}$, $H_{\mathcal{A}}(m,r)$ is given by the following recurrence:

$$H_{\mathcal{A}}(m,r) \leq$$
$$\leq \begin{cases} m^{1-\epsilon} H_{\mathcal{A}}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon} H_{\mathcal{B}}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + m^{1-\epsilon} H_{\mathcal{D}^*}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + m^{1-\epsilon}\frac{m^2}{Br}\right) \\ \qquad \text{if } r > m^{2-2\epsilon} \\ O\left(\frac{m^2}{B\sqrt{r}} + m^{1-\epsilon}\right) \qquad \text{if } r \leq m^{2-2\epsilon} \end{cases}$$

From which follows that

$$H_{\mathcal{A}}(m,r) \leq m^{1-\epsilon} H_{\mathcal{A}}\left(m^{\epsilon}, \frac{r}{m^{2-2\epsilon}}\right) + O\left(i \frac{m^2}{B\sqrt{r}} \log_{\epsilon} \frac{\log m^2/r}{\log m}\right),$$

and then

$$H_{\mathcal{A}}(m,r) \in O\left(\frac{m^2}{B\sqrt{r}} \log_{\epsilon}^2 \frac{\log m^2/r}{\log m}\right). \tag{5.14}$$

Since $H_{\epsilon\text{N-GEP}}(n,p,B) = O\left(H_{\mathcal{A}}(n,p)\right)$ and ϵ is a constant in $(0,1)$, Equation 5.11 follows.

Note that in ϵN-GEP there are some idle PEs in each superstep. In the network-oblivious algorithms described previously there are no idle PEs and the overhead incurred simulating each such network-oblivious algorithm on $\mathsf{M}(p,B)$ is asymptotically negligible compared to the number of operations of the algorithm itself. However, when there are idle processors, this could not be the case: thus, we analyze the computation complexity of ϵN-GEP in a different way. The computation complexity is given by the sum of the following two contributions:

- $T^a(n,p)$: computation complexity of ϵN-GEP on $\mathsf{M}(p,B)$ without considering the simulation.
- $T^s(n,p)$: the overhead incurred when simulating ϵN-GEP on $\mathsf{M}(p,B)$. A processor in each superstep distributes messages among its PEs and checks the status of every PE, even if idle.

Let us denote by $T^a_{\mathcal{T}}(m,r)$ the computation complexity of function $\mathcal{T}$ without considering the simulation cost. We remind that the base case is reached when $m \leq \log^{2/3}\log n$.

For function $\mathcal{D}^*$, we have:

$$T^a_{\mathcal{D}^*}(m,r) \leq \begin{cases} m^{1-\epsilon}T^a_{\mathcal{D}^*}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon}\frac{m^2}{r}\right) & \text{if } r > m^{2-2\epsilon} \\ \frac{m^{3-3\epsilon}}{r}T^a_{\mathcal{D}^*}\left(m^\epsilon,1\right) + O\left(m^{1-\epsilon}\frac{m^2}{r}\right) & \text{if } r \leq m^{2-2\epsilon} \\ O\left(m^3\right) & \text{if } m \leq \log^{2/3}\log n \end{cases}$$

which yields

$$T^a_{\mathcal{D}^*}(m,r) \in O\left(\frac{m^3}{r}\right).$$

For function $\mathcal{B}$, we have:

$$T^a_{\mathcal{B}}(m,r) \leq$$
$$\leq \begin{cases} m^{1-\epsilon}T^a_{\mathcal{B}}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon}\left(T^a_{\mathcal{D}^*}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + \frac{m^2}{r}\right)\right) & \text{if } r > m^{2-2\epsilon} \\ \frac{m^{2-2\epsilon}}{r}T^a_{\mathcal{B}}\left(m^\epsilon,1\right) + O\left(\frac{m^{3-3\epsilon}}{r}T^a_{\mathcal{D}^*}\left(m^\epsilon,1\right) + m^{1-\epsilon}\frac{m^2}{r}\right) & \text{if } r \leq m^{2-2\epsilon} \\ O\left(m^3\right) & \text{if } m \leq \log^{2/3}\log n \end{cases}$$

from which we get

$$T^a_{\mathcal{B}}(m,r) \in O\left(\frac{m^3}{r}\log_\epsilon \frac{\log m^2/r}{\log m}\right).$$

Finally, $T^a_{\mathcal{A}}(m,r)$ is bounded by this recurrence:

$$T^a_{\mathcal{A}}(m,r) \leq$$

$$\leq \begin{cases} m^{1-\epsilon} T^a_{\mathcal{A}}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + O\left(m^{1-\epsilon}\left(T^a_{\mathcal{B}}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + T^a_{\mathcal{D}^*}\left(m^\epsilon, \frac{r}{m^{2-2\epsilon}}\right) + \frac{m^2}{r}\right)\right) & \\ & \text{if } r > m^{2-2\epsilon} \\ m^{1-\epsilon} T^a_{\mathcal{A}}\left(m^\epsilon, 1\right) + O\left(\frac{m^{2-2\epsilon}}{r} T^a_{\mathcal{B}}\left(m^\epsilon, 1\right) + \frac{m^{3-3\epsilon}}{r} T^a_{\mathcal{D}^*}\left(m^\epsilon, 1\right) + m^{1-\epsilon}\frac{m^2}{r}\right) & \\ & \text{if } r \leq m^{2-2\epsilon} \\ O\left(m^3\right) & \text{if } m \leq \log^{2/3}\log n \end{cases}$$

which provides the following equation:

$$T^a_{\mathcal{A}}(m,r) \in O\left(\frac{m^3}{r}\log^2_\epsilon \frac{\log m^2/r}{\log m}\right).$$

Since ϵ is a constant in $(0,1)$, we obtain:

$$T^a(n,p) = T^a_{\mathcal{A}}(n,p) \in O\left(\frac{n^3}{p}\log^2 \frac{\log n}{\log(n^2/p)}\right).$$

The cost of message distribution is upper bounded by $T^a(n,p)$, since each PE performs an operation on every sent/received message and a naïve distribution requires $O(1)$ operations per message; therefore, we ignore it in $T^s(n,p)$. Since each processor performs $O(1)$ operations per PE in each superstep, we have that $T^s(n,p) = O((h(n)/p)\Lambda(n))$, where $\Lambda(n)$ denotes the number of supersteps of ϵN-GEP in $\mathsf{M}(h(n))$. We define by $\Lambda_{\mathcal{T}}(n)$, with $\mathcal{T} \in \{\mathcal{A},\mathcal{B},\mathcal{D}\}$, the number of supersteps in $\mathcal{T}$. Let $\ell(n) = \sqrt{n^2/h(n)} = \log^{2/3}\log n$. Since each phase of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}^*$ consists of a constant number of supersteps, it is not difficult to see that:

$$\Lambda_{\mathcal{D}^*}(m) \leq \begin{cases} m^{1-\epsilon}\Lambda_{\mathcal{D}^*}(m^\epsilon) + O\left(m^{1-\epsilon}\right) & \text{if } m > \ell(n) \\ 0 & \text{if } m \leq \ell(n) \end{cases},$$

which yields

$$\Lambda_{\mathcal{D}^*}(m) \in O\left(\frac{m}{\ell(n)}\right).$$

Similarly, it can be proved that:

$$\Lambda_{\mathcal{B}}(m) \in O\left(\frac{m}{\ell(n)}\log_\epsilon \frac{\log \ell(n)}{\log m}\right), \quad \Lambda_{\mathcal{A}}(m) \in O\left(\frac{m}{\ell(n)}\log^2_\epsilon \frac{\log \ell(n)}{\log m}\right).$$

Since ϵ is a constant in $(0,1)$ and by setting $m = n$, it follows that

$$T^s(n,p) \in O\left(\frac{n^2}{l^2(n)p}\Lambda_{\mathcal{A}}(n)\right) \in O\left(\frac{n^3}{l(n)^3 p}\log^2\frac{\log n}{\log l(n)}\right) \in O\left(\frac{n^3}{p}\right).$$

Note that, if ϵN-GEP were specified in an $\mathsf{M}(n^2)$, the cost of simulation would have been asymptotically bigger that the number of operations due to ϵN-GEP itself. For this reason, ϵN-GEP has been described on $\mathsf{M}(h(n))$, with $h(n) = n^2/\log^{4/3}\log n$.

Equation 5.12 follows since $T_{\epsilon\mathrm{N-GEP}}(n,p,B) \in O\left(T^a(n,p) + T^s(n,p)\right)$. □

Corollary 5.9. *Let ϵ be a constant in $(0,1)$. A commutative GEP computation on an $n \times n$ matrix can be performed by ϵN-GEP on an $\mathsf{M}(p,B)$ with optimal communication and computation complexities when $p \leq n^{2-\delta}$, with δ an arbitrary constant in $(0,2)$, and $B \leq (n/\sqrt{p})^{1+\epsilon}$.*

Proof. The proof descends from Theorem 5.8, and can be proved as Corollary 5.3. □

N-GEP exhibits optimal communication and computation complexities for a wider range of values of p than ϵN-GEP; however the restriction on the block size B is less restrictive in ϵN-GEP. Since the assumption on the block B in N-GEP is likely to be satisfied in practical scenarios and ϵN-GEP requires strong assumptions on the integrality of the quantities involved, N-GEP is more appealing than ϵN-GEP. Nevertheless, ϵN-GEP is still interesting because it is a parametric algorithm which becomes N-GEP when $\epsilon = 1 - 1/\log m$, and becomes PN-GEP (described in the next section) when $\epsilon = 0$. Moreover, observe that, by increasing ϵ by a constant factor, ϵN-GEP achieves optimality for bigger communication block sizes, whilst its communication and computation complexities increase by a constant factor.

ϵN-GEP requires the matrix on which the GEP computation will be applied to be distributed among the PEs according with the bit-interleaved layout. This assumption can be relaxed and, by means of a network-oblivious algorithm similar to the one in Section 4.2.2 for matrix transposition, a row or column-major layout can be used. However, the new implementation exhibits optimal communication complexity on $\mathsf{M}(p,B)$ if $p \leq n^{2-\delta}$, for an arbitrary constant δ in $(0,2)$, and $B \leq \min\{n, (n/\sqrt{p})^{1+\epsilon}\}$.

It must be remarked that ϵN-GEP does not satisfy the fullness assumption since, when $p \leq n^{2-2\epsilon}$, the PEs that a processor is simulating send $O\left(n^2/p\right)$ messages, however only $O\left(n^{1+\epsilon}\sqrt{p}\right)$ of these are addressed outside the processor. We do not analyze the communication time of ϵN-GEP in a D-BSP, however note that it requires $O(n^{1-\epsilon})$ 0-supersteps, while N-GEP uses only $O(1)$ 0-supersteps.

ϵN-GEP is not of appealing in practical scenarios for requiring strong assumptions on the integrality of the quantities involved. Nevertheless, it is still interesting because it is a parametric algorithm which becomes N-GEP when $\epsilon = 1 - 1/\log m$, and becomes PN-GEP when $\epsilon = 0$. Moreover, by increasing ϵ by a constant factor, ϵN-GEP achieves optimality for bigger communication block sizes, whilst its communication and computation complexities increase by a constant factor.

5.2.3 PN-GEP

PN-GEP (*Pipelined GEP*) is a network-oblivious algorithm for performing any GEP computation, whose specification is given in $\mathsf{M}(n^2)$. It is a reminiscent of the systolic algorithm proposed by Lewis and Kung in [LK91] for the *algebraic path problems*, which are special cases of GEP computations and include Floyd-Warshall all-pairs shortest paths and matrix multiplication.

The algorithm descends from function $\mathcal{A}$ of ϵN-GEP by setting[5] $\epsilon = 0$. Specifically, PN-GEP partitions the $n \times n$ input matrix into n^2 submatrices of size 1×1, thus defining n^3 subproblems. Subproblems, which fall immediately into the base cases, are solved as in ϵN-GEP's function $\mathcal{A}$. It is immediate to see that PN-GEP implements correctly all GEP computations for which I-GEP is correct, although correctness in the general case can be achieved by modifying the algorithm using similar techniques to those employed in C-GEP [CR07].

The following theorem and corollary show the optimality of PN-GEP on an $\mathsf{M}(p, B)$, for all values of p and a wide range of values of B. Note also that the restriction to commutative GEP computations is not needed in this case.

Theorem 5.10. *A GEP computation on an $n \times n$ matrix can be performed by PN-GEP with communication and computation complexities*

$$H_{N-GEP}(n, p, B) \in O\left(\frac{n^2}{B\sqrt{p}} + n\right), \tag{5.15}$$

$$T_{N-GEP}(n, p, B) \in \Theta\left(\frac{n^3}{p}\right) \tag{5.16}$$

on an $\mathsf{M}(p, B)$, with $1 < p \leq n^2$ and $B \geq 1$.

Proof. Each $\mathsf{M}(p, B)$ processor holds an $n/\sqrt{p} \times n/\sqrt{p}$ submatrix of PEs, but only PEs on the border send messages outside the processor. Since the algorithm is com-

[5] ϵN-GEP is defined when $0 < \epsilon < 1$, but function $\mathcal{A}$ is well defined when $\epsilon = 0$.

posed of $\Theta(n)$ supersteps, Equations 5.15 and 5.16 follows. □

Corollary 5.11. *A commutative GEP computation on an $n \times n$ matrix can be performed by PN-GEP with optimal communication and computation complexities when $1 < p \leq n^2$ and $1 \leq B \leq n/\sqrt{p}$.*

Proof. The proof is straightforward. □

Although, PN-GEP yields optimal computation and communication complexities in the evaluation model for a wide range of the parameters, it is not optimal in a D-BSP. Indeed, PN-GEP uses only 0-supersteps and does not exploit the hierarchical structure of the D-BSP interconnection topology.

Chapter 6

Conclusions

Philosophically, the reason research in math matters is that by pursuing math ideas that are deep and interesting for their own sake, you will get real-world applications in the future.
(Steven Hofmann)

In this final chapter we summarize the main contributions of the thesis and discuss some future research directions.

6.1 Summary

In this thesis we contributed novel results on oblivious algorithms, pursuing two main directions: the investigation of the potentialities and intrinsic limitations of oblivious versus aware algorithms, and the introduction of the notion of oblivious algorithm in parallel setting.

In Chapter 3, we studied various aspects concerning the execution of rational permutations in a cache-RAM hierarchy focusing, in particular, on the oblivious settings. We first proved a lower bound on the work complexity of any algorithm that executes rational permutations with optimal cache complexity. By virtue of this bound we were able to show the work optimality of the cache-aware algorithm derivable from the one in [Cor93a], which also exhibits optimal cache complexity. Then, we developed a cache-oblivious algorithm for performing any rational permutation, which exhibits optimal cache and work complexities under the tall-cache assumption. When the rational permutation is a matrix transposition, our cache-oblivious algorithm represents an iterative version of the recursive cache-oblivious algorithm given in [FLPR99]. Finally, we investigate the separation in asymptotic complexity between cache-aware and cache-oblivious algorithms, showing that for certain families of rational permutations, including matrix transposition and bit-reversal, a cache-oblivious algorithm which achieves optimal cache complexity for all values of the Ideal Cache parameters cannot exist, while this is attainable through a cache-aware algorithm. This result specializes to the case of rational permutations the result proved in [BF03] for general permutations, and it is achieved by means of a simulation technique which formalizes the approach used in [BF03]. To the best of our knowledge, the only impossibility results of the kind presented in this chapter and in [BF03], were proved in [BP01]. These results provide interesting insights on the trade-off between efficiency and portability of cache-oblivious algorithms, and also shed light on the interaction of algorithms with the Translation Lookaside Buffer (TLB) which, if regarded as a cache, does generally not satisfy the tall-cache assumption [Kum03]. Indeed, several algorithms, even if aware, do not use TLB parameters, and with respect to this component they are oblivious.

In Chapter 4, we proposed a framework for the study of oblivious algorithms in the parallel setting. This framework explores the design of bulk-synchronous parallel algorithms that, without resorting to parameters for tuning the performance on the target platform, can execute efficiently on parallel machines with different degree of parallelism and bandwidth characteristics. A network-oblivious algorithm is designed on the specification model, which consists of a clique of processor/memory pairs, called processing elements, whose number is function exclusively of the input size. Then, the communication and computation complexities of the network-oblivious algorithm are analyzed in the evaluation model, which is similar to the specification model but provides two parameters, namely processor number and communication block-size, which capture parallelism and granularity of communication. Finally, the algorithm is run on the execution model, which is a block-variant of the D-BSP model. The framework is appealing because, as proved in the thesis, for a wide class of network-oblivious algorithms, optimality in the evaluation model implies optimality in the D-BSP model. We showed that a number of key problems, namely matrix multiplication and transposition, discrete Fourier transform and sorting, admit network-oblivious algorithms which are optimal for a wide range of machine parameters. We also present a further result on the separation between oblivious and aware approaches, showing the impossibility of designing a network-oblivious algorithm for matrix transposition which is optimal for all values of the evaluation model parameters, while this is attainable through an aware parallel approach.

Finally, in Chapter 5 we presented three algorithms, called N-GEP, ϵN-GEP and PN-GEP, for solving a wide class of computations encompassed by the Gaussian Elimination Paradigm [CR06], which includes all-pairs shortest paths, Gaussian elimination without pivoting and matrix multiplication. They are based on the parallel and cache-oblivious implementations of the GEP paradigm given in [CR07, CR08], and perform optimally in the evaluation model for different ranges of the parameters. In particular, ϵN-GEP and PN-GEP exploit large communication block size and an high number of processors, respectively, better than N-GEP. However, N-GEP is more appealing since we proved that it is also optimal in the D-BSP model for some ranges of the parameters, while this is not the case for PN-GEP and we conjecture for ϵN-GEP as well.

6.2 Further research

An interesting avenue for further research is to continue the study of the theoretical separation between oblivious and aware approaches, for instance, by proving impossibility results similar to those presented in this thesis and in previous works, for other fundamental problems. Moreover, deeper investigations are required to understand why the tall-cache and small-block assumptions are so crucial in certain cases to obtain optimal cache and network-oblivious algorithms.

The network-oblivious framework offers several interesting directions for further work. Naturally, one goal is to design efficient network-oblivious algorithms for other important problems, beyond the ones proposed in this thesis. Another issue regards the necessity of the assumptions made in Theorem 4.5 to prove that optimality in the evaluation model translates into optimality in the D-BSP. In fact, in the thesis we developed a network-oblivious algorithm, N-GEP, which does not satisfies these assumptions, and yet it is optimal in a D-BSP for certain ranges of the parameters. It is evident that assumptions in the theorem are too conservative. Furthermore, it would be interesting to study the relations between optimality in the evaluation model and optimality in other execution models alternative to the D-BSP.

So far the network-oblivious framework and the algorithms developed as case studies have been investigated only from a theoretical perspective by means of asymptotic analysis. Cache-oblivious algorithms are fully compatible with actual platforms in the sense that the simulation of the specification model in the execution model is performed by the system through automatic replacement policies of cache lines. An analogous simulation tool is not available for running network-oblivious algorithms on parallel architectures. Hence, the realization of a library to support the execution of network-oblivious algorithms is of fundamental importance for assessing experimentally the actual performance attained by this type of algorithms.

Finally, another interesting research work is to develop new compiling techniques or to design specific architectural features which provide (a more) effective support to the execution of oblivious algorithms. One example, as noted in [Fri99], is the development of techniques to reduce the cost of procedure calls which are useful in cache and network-oblivious algorithms. Another one is the identification of fast techniques to support execution of large number of light threads, that would be useful for the simulation of processing elements of the specification model in processors of the execution model in the network-oblivious framework.

Appendix A

Properties of Function $f(x)$

We prove some lemmas that have been used in Chapter 3. Remind that the convex function $f(x)$ is defined as follows:

$$f(x) = \begin{cases} x \log x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}.$$

Lemma A.1. *If $b_i \geq 0$ for each i, with $0 \leq i < k$, and $\sum_{i=1}^{k-1} b_i \leq b_0$, then*

$$f\left(b_0 - \sum_{i=1}^{k-1} b_i\right) + \sum_{i=1}^{k-1} f(b_i) \leq f(b_0).$$

Proof. Without loss of generality we suppose $b_i > 0$ for each i, with $0 \leq i < k$, and $\sum_{i=1}^{k-1} f(b_i) > f(b_0)$. (The general case is a simple adaptation.) By the definition of f we get:

$$\begin{aligned} f\left(b_0 - \sum_{i=1}^{k-1} b_i\right) + \sum_{i=1}^{k-1} f(b_i) &= \left(b_0 - \sum_{i=1}^{k-1} b_i\right) \log \left(b_0 - \sum_{i=1}^{k-1} b_i\right) + \left(\sum_{i=1}^{k-1} b_i \log b_i\right) \\ &\leq \left(b_0 - \sum_{i=1}^{k-1} b_i\right) \log b_0 + \left(\sum_{i=1}^{k-1} b_i\right) \log b_0 = f(b_0). \end{aligned}$$

□

Lemma A.2. *Let $\boldsymbol{a} = (a_0, \ldots, a_{s-1})$ and $a_k > 0$ for each k, with $0 \leq k < s$. The function*

$$F(\boldsymbol{a}) = f\left(\sum_{k=0}^{s-1} a_k\right) - \sum_{k=0}^{s-1} f(a_k)$$

is concave and non decreasing for each a_k, with $0 \leq k < s$.

Proof. Let $H(\boldsymbol{a})$ be the Hessian matrix of $F(\boldsymbol{a})$ and denote with $h_{i,j}(\boldsymbol{a})$ the entry of H in the i-th row and j-th column, that is, the second order partial derivative of function $F(\boldsymbol{a})$ with respect to variables a_j and a_i (in the given order). It is not difficult to see that:

$$h_{i,j}(\boldsymbol{a}) = \begin{cases} \dfrac{1}{\sum_{k=0}^{s-1} a_k} & \text{if } i \neq j \\ \\ \dfrac{1}{\sum_{k=0}^{s-1} a_k} - \dfrac{1}{a_i} & \text{if } i = j \end{cases} .$$

Let $x = (x_0, \ldots, x_{n-1})$ and $x \in \mathbb{R}^s$. We get:

$$xH(\boldsymbol{a})x^T = -\frac{1}{\sum_{k=0}^{s-1} a_k} \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} \left(\sqrt{\frac{a_i}{a_j}} x_i - \sqrt{\frac{a_j}{a_i}} x_j \right)^2 \leq 0$$

Hence, $H(\boldsymbol{a})$ is a negative semidefinite matrix and $F(\boldsymbol{a})$ is concave.

Since the first order partial derivatives of $F(\boldsymbol{a})$ are non negative when $a_k > 0$ for each k, with $0 \leq k < s$, $F(\boldsymbol{a})$ is non decreasing for each a_k. □

Lemma A.3. *Let A be an $r \times s$ matrix, with $r, s \geq 1$, and denote with $a_{i,j} \geq 0$ the entry of A in the i-th row and j-th column. For $0 \leq j < s$ let M_j be a fixed value such that $\sum_{i=0}^{r-1} a_{i,j} \leq M_j$. Then an upper bound on the function*

$$\tilde{F}(A) = \sum_{i=0}^{r-1} \left[f\left(\sum_{j=0}^{s-1} a_{i,j} \right) - \sum_{j=0}^{s-1} f\left(a_{i,j}\right) \right] \tag{A.1}$$

is given by setting $a_{i,j} = M_j/r$ for all i and j, with $0 \leq i < r$ and $0 \leq j < s$.

Proof. Suppose $a_{i,j} > 0$ for each i and j, with $0 \leq i < r$ and $0 \leq j < s$. $\tilde{F}(A)$ can be rewritten as follows:

$$\tilde{F}(A) = r \sum_{i=0}^{r-1} \frac{F\left((a_{i,0}, \ldots, a_{i,s-1})\right)}{r}.$$

Since $F(\cdot)$ is concave, we can apply the multivariate Jensen's inequality [Neu90]:

$$\begin{aligned} \tilde{F}(A) &\leq rF\left(\sum_{i=0}^{r-1} \frac{(a_{i,0}, \ldots, a_{i,s-1})}{r} \right) \\ &\leq rF\left(\left(\sum_{i=0}^{r-1} \frac{a_{i,0}}{r}, \ldots, \sum_{i=0}^{r-1} \frac{a_{i,s-1}}{r} \right) \right) \end{aligned}$$

Since $\sum_{i=0}^{r-1} a_{i,j} \leq M_j$ and $F(\cdot)$ is non decreasing for each component, we get:

$$\begin{aligned}\tilde{F}(A) &\leq \sum_{i=0}^{r-1} F\left(\left(\frac{M_0}{r}, \ldots, \frac{M_{s-1}}{r}\right)\right) \\ &\leq \sum_{i=0}^{r-1}\left[f\left(\sum_{j=0}^{s-1}\frac{M_j}{r}\right) - \sum_{j=0}^{s-1} f\left(\frac{M_j}{r}\right)\right],\end{aligned}$$

from which the lemma follows. The previous equation is an upper bound on $\tilde{F}(A)$ even if there are some $a_{i,j} = 0$ since $F(\cdot)$ is continuous and non decreasing in each component. □

Corollary A.4. *Let A be an $r \times s$ matrix, with $r, s \geq 1$, and denote with $a_{i,j}$ the entry of A in the i-th row and j-th column. Let M_0 and M be two fixed values such that $\sum_{i=0}^{r-1} a_{i,0} \leq M_0$ and $\sum_{j=1}^{s-1}\sum_{i=0}^{r-1} a_{i,j} \leq M$. Then, an upper bound to function $F(A)$, defined in Equation A.1, is given by setting $a_{i,0} = M_0/r$ and $a_{i,j} = M/(r(s-1))$ for each i and j, with $0 \leq i < r$ and $1 \leq j < s$.*

Proof. Let M_j denote the partial sum $\sum_{i=0}^{r-1} a_{i,j}$ for each j with $0 \leq j < s$. By Lemma A.3, we have that

$$\begin{aligned}\tilde{F}(A) &\leq \sum_{i=0}^{r-1}\left[f\left(\sum_{j=0}^{s-1}\frac{M_j}{r}\right) - \sum_{j=0}^{s-1} f\left(\frac{M_j}{r}\right)\right]. \\ &\leq \sum_{i=0}^{r-1}\left[f\left(\frac{M_0}{r} + \sum_{j=1}^{s-1}\frac{M}{r(s-1)}\right) - f\left(\frac{M_0}{r}\right)\right] - \sum_{i=0}^{r-1}\sum_{j=1}^{s-1} f\left(\frac{M_j}{r}\right). \qquad \text{(A.2)}\end{aligned}$$

For the Jensen's inequality, we have:

$$-\sum_{i=0}^{r-1}\sum_{j=1}^{s-1} f\left(\frac{M_j}{r}\right) \leq -\sum_{i=0}^{r-1}\sum_{j=1}^{s-1} f\left(\frac{M}{r(s-1)}\right).$$

Thus,the following upper bound on $\tilde{F}(A)$ follows:

$$\tilde{F}(A) \leq \sum_{i=0}^{r-1}\left[f\left(\frac{M_0}{r} + \sum_{j=1}^{s-1}\frac{M}{r(s-1)}\right) - f\left(\frac{M_0}{r}\right) - \sum_{j=1}^{s-1} f\left(\frac{M}{r(s-1)}\right)\right].$$

□

Appendix B

Pseudocode of PI-GEP1 and PI-GEP2

In this appendix, we reproduce the pseudocode of the two implementations of I-GEP described in [CR07] and [CR08] for a CREW (concurrent read, exclusive write) shared-memory model composed of p processors and one level of cache that can be either shared or distributed. We refer to them as PI-GEP1 and PI-GEP2, respectively. The initial call in both algorithms is $\mathcal{A}(x, n)$, where x is the $n \times n$ input matrix of the GEP computation.

```
$\mathcal{A}(X, m)$
**INPUT:** matrix $X \equiv x[I, I]$ with $I = [i_0, i_1] \subseteq [0, n)$ with $m = i_1 - i_0 + 1$.
**OUTPUT:** execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times I \times I)$.
 1: if $T = \emptyset$ then return;
 2: if $m = 1$ then
 3:     $X[i_0, i_0] \leftarrow f(X[i_0, i_0], X[i_0, i_0], X[i_0, i_0], X[i_0, i_0])$;
 4: else
 5:     $\mathcal{A}(X_{0,0}, m/2)$;
 6:     In parallel invoke $\mathcal{B}(X_{0,1}, X_{0,0}, m/2)$, $\mathcal{C}(X_{1,0}, X_{0,0}, m/2)$
 7:     $\mathcal{D}(X_{1,1}, X_{1,0}, X_{0,1}, X_{0,0}, m/2)$;
 8:     $\mathcal{A}(X_{1,1}, m/2)$;
 9:     In parallel invoke $\mathcal{B}(X_{1,0}, X_{1,1}, m/2)$, $\mathcal{C}(X_{0,1}, X_{1,1}, m/2)$;
10:     $\mathcal{D}(X_{0,0}, X_{0,1}, X_{1,0}, X_{1,1}, m/2)$;
```

Figure B.1: Function $\mathcal{A}$ of PI-GEP1[CR07].

```
$\mathcal{B}(X, U, m)$
INPUT: matrices $X \equiv x[I, J]$ and $U \equiv x[I, I]$, with $I = [i_0, i_1] \subseteq [0, n)$, $J = [j_0, j_1] \subseteq$
    $[0, n)$, $I \cap J = \emptyset$ and $m = i_1 - i_0 + 1 = j_1 - j_0 + 1$.
OUTPUT: execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times I)$.
 1: if $T = \emptyset$ then return;
 2: if $m = 1$ then
 3:     $X[i_0, i_0] \leftarrow f(X[i_0, j_0], U[i_0, i_0], X[i_0, j_0], U[i_0, i_0])$;
 4: else
 5:     parallel: $\mathcal{B}(X_{0,0}, U_{0,0}, m/2), \mathcal{B}(X_{0,1}, U_{0,0}, m/2)$;
 6:     parallel: $\mathcal{D}(X_{1,0}, U_{1,0}, X_{0,0}, U_{0,0}, m/2), \mathcal{D}(X_{1,1}, U_{1,0}, X_{0,1}, U_{0,0}, m/2)$;
 7:     parallel: $\mathcal{B}(X_{1,0}, U_{1,1}, m/2), \mathcal{B}(X_{1,1}, U_{1,1}, m/2)$;
 8:     parallel: $\mathcal{D}(X_{0,0}, U_{0,1}, X_{1,0}, U_{1,1}, m/2), \mathcal{D}(X_{0,1}, U_{0,1}, X_{1,1}, U_{1,1}, m/2)$;
```

Figure B.2: Function $\mathcal{B}$ of PI-GEP1[CR07].

```
$\mathcal{C}(X, V, m)$
INPUT: matrices $X \equiv x[I, J]$ and $V \equiv x[J, J]$, with $I = [i_0, i_1] \subseteq [0, n)$, $J = [j_0, j_1] \subseteq$
    $[0, n)$, $I \cap J = \emptyset$ and $m = i_1 - i_0 + 1 = j_1 - j_0 + 1$.
OUTPUT: execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times J)$.
 1: if $T = \emptyset$ then return;
 2: if $m = 1$ then
 3:     $X[i_0, j_0] \leftarrow f(X[i_0, j_0], X[i_0, j_0], V[j_0, j_0], V[j_0, j_0])$;
 4: else
 5:     parallel: $\mathcal{C}(X_{0,0}, V_{0,0}, m/2), \mathcal{C}(X_{1,0}, V_{0,0}, m/2)$;
 6:     parallel: $\mathcal{D}(X_{0,1}, X_{0,0}, V_{0,1}, V_{0,0}, m/2), \mathcal{D}(X_{1,1}, X_{1,0}, V_{0,1}, V_{0,0}, m/2)$;
 7:     parallel: $\mathcal{C}(X_{0,1}, V_{1,1}, m/2), \mathcal{C}(X_{1,1}, V_{1,1}, m/2)$;
 8:     parallel: $\mathcal{D}(X_{0,0}, X_{0,1}, V_{1,0}, V_{1,1}, m/2), \mathcal{D}(X_{1,0}, X_{1,1}, V_{1,0}, V_{1,1}, m/2)$;
```

Figure B.3: Function $\mathcal{C}$ of PI-GEP1[CR07].

```
$\mathcal{D}(X, U, V, W, m)$
INPUT: matrices $X \equiv x[I, J]$, $U \equiv x[I, K]$, $V \equiv x[K, J]$ and $W \equiv x[K, K]$, with $I =$
    $[i_0, i_1] \subseteq [0, n)$, $J = [j_0, j_1] \subseteq [0, n)$, $K = [k_0, k_1] \subseteq [0, n)$, $I \cap K = \emptyset$, $J \cap K = \emptyset$ and
    $m = i_1 - i_0 + 1 = j_1 - j_0 + 1 = k_1 - k_0 + 1$.
OUTPUT: execution of all updates $\langle i, j, k \rangle \in T$, with $T = \Sigma_f \cap (I \times J \times K)$.
 1: if $T = \emptyset$ then return;
 2: if $m = 1$ then
 3:     $X[i_0, j_0] \leftarrow f(X[i_0, j_0], U[i_0, k_0], V[k_0, j_0], W[k_0, k_0])$;
 4: else
 5:     parallel: $\mathcal{D}(X_{0,0}, U_{0,0}, V_{0,0}, W_{0,0}, m/2), \mathcal{D}(X_{0,1}, U_{0,0}, V_{0,1}, W_{0,0}, m/2)$,
                  $\mathcal{D}(X_{1,0}, U_{1,0}, V_{0,0}, W_{0,0}, m/2), \mathcal{D}(X_{1,1}, U_{1,0}, V_{0,1}, W_{0,0}, m/2)$;
 6:     parallel: $\mathcal{D}(X_{0,0}, U_{0,1}, V_{1,0}, W_{1,1}, m/2), \mathcal{D}(X_{0,1}, U_{0,1}, V_{1,1}, W_{1,1}, m/2)$,
                  $\mathcal{D}(X_{1,0}, U_{1,1}, V_{1,0}, W_{1,1}, m/2), \mathcal{D}(X_{1,1}, U_{1,1}, V_{1,1}, W_{1,1}, m/2)$.
```

Figure B.4: Function $\mathcal{D}$ of PI-GEP1[CR07].

```
$\mathcal{A}(X, m)$
1: As in Lines 1-4 of $\mathcal{A}$ in Figure B.1;
2: for $k \leftarrow 0$ to $r - 1$ do
3:     $\mathcal{A}(X_{k,k}, m/r)$;
4:     parallel: $\mathcal{B}(X_{k,j}, X_{k,k}, m/r)$, $\mathcal{C}(X_{i,k}, X_{k,k}, m/r)$, for all $0 \leq i, j < r$ and $i, j \neq k$;
5:     parallel: $\mathcal{D}(X_{i,j}, X_{i,k}, X_{k,j}, X_{k,k}, m/r)$, for all $0 \leq i, j < r$ and $i, j \neq k$;

$\mathcal{B}(X, U, m)$
1: As in Lines 1-4 of $\mathcal{B}$ in Figure B.2;
2: for $k \leftarrow 0$ to $r - 1$ do
3:     parallel: $\mathcal{B}(X_{k,j}, U_{k,k}, m/r)$, for all $0 \leq j < r$ and $j \neq k$;
4:     parallel: $\mathcal{D}(X_{i,j}, U_{i,k}, X_{k,j}, U_{k,k}, m/r)$, for all $0 \leq i, j < r$ and $i, j \neq k$;

$\mathcal{C}(X, V, m)$
1: As in Lines 1-4 of $\mathcal{C}$ in Figure B.3;
2: for $k \leftarrow 0$ to $r - 1$ do
3:     parallel: $\mathcal{C}(X_{i,k}, V_{k,k}, m/r)$, for all $0 \leq i < r$ and $i \neq k$;
4:     parallel: $\mathcal{D}(X_{i,j}, X_{i,k}, V_{k,j}, V_{k,k}, m/r)$, for all $0 \leq i, j < r$ and $i, j \neq k$;

$\mathcal{D}(X, U, V, W, m)$
1: As in Lines 1-4 of $\mathcal{D}$ in Figure B.4;
2: for $k \leftarrow 0$ to $r - 1$ do
3:     parallel: $\mathcal{D}(X_{i,j}, U_{i,k}, V_{k,j}, W_{k,k}, m/r)$, for all $0 \leq i, j < r$;
```

Figure B.5: Functions $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ of PI-GEP2 [CR08].

Bibliography

[AACS87] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 305–314, 1987.

[ABC$^+$06] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006.

[ABF05] Lars Arge, Gerth Stølting Brodal, and Rolf Fagerberg. Cache-oblivious data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 34, page 27. CRC Press, 2005.

[ACFS94] Bowen Alpern, Larry Carter, Ephraim Feig, and Ted Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2/3):72–109, 1994.

[ACS87] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Hierarchical memory with block transfer. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 204–216, 1987.

[ACS90] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

[Aea02] Narasimha R. Adiga and et al. An overview of the BlueGene/L supercomputer. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 1–22, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[Aeo] AEOLUS project website *http://aeolus.ceid.upatras.gr*.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Arg04] Lars Arge. External geometric data structures. In Kyung-Yong Chwa and J. Ian Munro, editors, *Proceedings of the 10th Annual International Conference of Computing and Combinatorics*, volume 3106 of *Lecture Notes in Computer Science*. Springer, 2004.

[Arv81] Arvind. Data flow languages and architecture. In *Proceedings of the 8th Annual Symposium on Computer Architecture*, 1981.

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[Bac78] John K. Backus. Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs. *Communication of the ACM*, 21(8):613–641, 1978.

[BBP99] Sandeep N. Bhatt, Gianfranco Bilardi, and Geppino Pucci. Area-universal circuits with constant slowdown. In *Proceedings of the 18th International Conference on Advanced Research in VLSI*, pages 89–98, 1999.

[BDadH98] Armin Bäumker, Wolfgang Dittrich, and Friedhelm Meyer auf der Heide. Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model. In *Selected papers from the 3rd European Symposium on Algorithms*, pages 175–203, Amsterdam, The Netherlands, 1998. Elsevier Science Publishers B. V.

[BDP99] Armin Bäumker, Wolfgang Dittrich, and Andrea Pietracaprina. The complexity of parallel multisearch on coarse-grained machines. *Algorithmica*, 24(3-4):209–242, 1999.

[Bel66] Laszlo A. Belady. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[Ber08] Alberto Bertoldo. *An adaptive parallel solver for finite-elements applications*. PhD thesis, Department of Information Engineering, University of Padova, 2008.

[BF03] Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 307–315, June 2003.

[BFGK05] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Bradley C. Kuszmaul. Concurrent cache-oblivious B-trees. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 228–237, New York, NY, USA, 2005. ACM.

[BFPP01] Gianfranco Bilardi, Carlo Fantozzi, Andrea Pietracaprina, and Geppino Pucci. On the effectiveness of D-BSP as a bridging model of parallel computation. In *Proceedings of the International Conference on Computational Science-Part II*, volume 2074 of *Lecture Notes in Computer Science*, pages 579–588, London, UK, 2001. Springer-Verlag.

[BP97] Gianfranco Bilardi and Franco Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part I, upper bounds. *Theory of Computing Systems*, 30:523–546, 1997.

[BP99] Gianfranco Bilardi and Franco Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems*, 32:531–559, 1999.

[BP01] Gianfranco Bilardi and Enoch Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 128–139, 2001.

[BPP99] Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci. A quantitative measure of portability with application to bandwidth-latency models for parallel computing. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 543–551, September 1999.

[BPP$^+$05] Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, Fabio Schifano, and Raffaele Tripiccione. The potential of on-chip multiprocessing for QCD machines. In *Proceedings of the 12th International Conference on High-Performance Computing*, volume 3769 of *Lecture Notes in Computer Science*, pages 386–397, 2005.

[BPP07] Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci. Decomposable BSP: A bandwidth-latency model for parallel and hierarchical computation. In John Reif and Sanguthevar Rajasekaran, editors, *Handbook of Parallel Computing: Models, Algorithms and Applications*, pages 277–315. CRC Press, 2007.

[BPPS07] Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri. Network-oblivious algorithms. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, March 2007.

[CFSV95] Thomas Cheatham, Amr F. Fahmy, Dan C. Stefanescu, and Leslie G. Valiant. Bulk synchronous parallel computing - a paradigm for transportable software. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, page 268, Washington, DC, USA, 1995. IEEE Computer Society.

[CG98] Larry Carter and Kang Su Gatlin. Towards an optimal bit-reversal permutation program. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 544–555, 1998.

[Cho07] Rezaul A. Chowdhury. *Cache-efficient Algorithms and Data Structures: Theory and Experimental Evaluation.* PhD thesis, Department of Computer Sciences, The University of Texas at Austin, August 2007.

[CKP$^+$96] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Eunice E. Santos, Klaus E. Schauser, Ramesh Subramonian, and Thorsten von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.

[CLPT99] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the 11th ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, 1999.

[CLPT02] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast matrix multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 13(11):1105–1123, November 2002.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, USA, second edition, September 2001.

[Cor93a] Thomas H. Cormen. Fast permuting on disk arrays. *Journal of Parallel and Distributed Computing*, 17(1-2), 1993.

[Cor93b] Thomas H. Cormen. *Virtual Memory for Data-Parallel Computing.* PhD thesis, Massachussetts Institute of Technology, 1993.

[CR06] Rezaul A. Chowdhury and Vijaya Ramachandran. Cache-oblivious dynamic programming. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithm*, pages 591–600, 2006.

[CR07] Rezaul A. Chowdhury and Vijaya Ramachandran. The cache-oblivious Gaussian elimination paradigm: theoretical framework, parallelization and experimental evaluation. In *Proceedings of the 19th annual Symposium on Parallelism in Algorithms and Architectures*, pages 71–80, New York, NY, USA, 2007. ACM.

[CR08] Rezaul A. Chowdhury and Vijaya Ramachandran. Cache-efficient dynamic programming algorithms for multicores. In *Proceedings of the 20th Symposium on Parallelism in Algorithms and Architectures*, pages 207–216, New York, NY, USA, 2008. ACM.

[CSG98] David Culler, J. P. Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, August 1998.

[Dem56] Howard B. Demuth. *Electronic Data Sorting.* PhD thesis, Stanford University, October 1956.

[Dem85] Howard B. Demuth. Electronic data sorting. *IEEE Transactions on Computers*, 34(4):296–310, 1985.

[Dem02] Erik Demaine. Cache-oblivious algorithms and data structures. Lecture Notes from the EEF Summer School on Massive Data Sets, 2002.

[DHMD99] Frank Dehne, David Hutchinson, Anil Maheshwari, and Wolfgang Dittrich. Reducing I/O complexity by simulating coarse grained parallel algorithms. In *Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed*

Processing, pages 14–20, Washington, DC, USA, 1999. IEEE Computer Society.

[DlTK95] Pilar De la Torre and Clyde P. Kruskal. A structural theory of recursively decomposable parallel processor-networks. In *Proceedings of the IEEE Symposium on Parallel and Distributeed Processing*, page 570, Washington, DC, USA, 1995. IEEE Computer Society.

[DlTK96] Pilar De la Torre and Clyde P. Kruskal. Submachine locality in the bulk synchronous setting. In *Proceedings of the 2nd International Euro-Par Conference on Parallel Processing-Volume II*, volume 1124 of *Lecture Notes in Computer Science*, pages 352–358, August 1996.

[Fan03] Carlo Fantozzi. *A Computational Model for Parallel and Hierarchical Machines.* PhD thesis, Department of Information Engineering, University of Padova, 2003.

[FIP99] FIPS PUB 46-3. *Data Encryption Standard (DES).* National Institute for Standards and Technology, Gaithersburg, MD, USA, October 1999.

[FJ98] Matteo Frigo and Steven G. Johnson. FFTW: an adaptive software architecture for the FFT. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 3:1381–1384 vol.3, May 1998.

[FK03] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Flo72] Richard W. Floyd. Permuting information in idealized two-level storage. In R. Miller and J. W. Thatche, editors, *Complexity of Computer Computations*, pages 105–109. Plenum, 1972.

[FLPR99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 285–298, 1999.

[FPP03] Carlo Fantozzi, Andrea Pietracaprina, and Geppino Pucci. A general Pram simulation scheme for clustered machines. *International Journal on Foundations of Computer Science*, 14(6):1147–1164, 2003.

[FPP06] Carlo Fantozzi, Andrea Pietracaprina, and Geppino Pucci. Translating submachine locality into locality of reference. *Journal of Parallel and Distributed Computing*, 66(5):633–646, 2006.

[Fri99] Matteo Frigo. *Portable High-Performance Programs.* PhD thesis, Massachusetts Institute of Technology, June 1999.

[Fri08] Matteo Frigo. Personal communication, 2008.

[FS06] Matteo Frigo and Volker Strumpen. The cache complexity of multithreaded cache oblivious algorithms. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 271–280, New York, NY, USA, 2006. ACM.

[FW78] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 114–118, New York, NY, USA, 1978. ACM.

[GMR99] Phillip B. Gibbons, Y. Matias, and Vijaya Ramachandran. Can a shared-memory model serve as a bridging-model for parallel computation? *Theory of Computing Systems*, 32(3):327–359, 1999.

[Gol78] Leslie M. Goldschlager. A unified approach to models of synchronous parallel machines. In *Proceedings of the 10th annual ACM symposium on Theory of computing*, pages 89–94, New York, NY, USA, 1978. ACM.

[Goo99] Michael T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, October 1996.

[GVW96] Garth A. Gibson, Jeffrey Scott Vitter, and John Wilkes. Strategic directions in storage I/O issues in large-scale computing. *ACM Computer Survey*, 28(4):779–793, 1996.

[HK81] Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 326–333, New York, NY, USA, 1981. ACM.

[HP06] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[ITT04] Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[JáJ92] Joseph JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1992.

[KA02] Ken Kennedy and John R. Allen. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[Ker70] Leslie Robert Kerr. *The effect of algebraic structure on the computational complexity of matrix multiplication.* PhD thesis, Cornell University, 1970.

[Knu98] Donald E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley Professional, second edition, April 1998.

[KR90] Richard M. Karp and Vijaya Ramachandran. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 869–942. MIT Press, 1990.

[Kum03] Piyush Kumar. Cache oblivious algorithms. In Meyer et al. [MSS03], pages 193–212.

[Lei85] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.

[Lei92] Frank Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Kaufmann, 1992.

[LK91] Paul S. Lewis and Sun-Yuan Kung. An optimal systolic array for the algebraic path problem. *IEEE Transactions on Computers*, 40(1):100–105, 1991.

[MHS05] Marjan Merenik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-speciïňĄc languages. *ACM Computer Surveys*, 37(4):316–344, 2005.

[MS00] Kurt Mehlhorn and Peter Sanders. Scanning multiple sequences via cache memory. *Algorithmica*, 35:2003, 2000.

[MSS03] Ulrich Meyer, Peter Sanders, and Jop F. Sibeyn, editors. *Algorithms for Memory Hierarchies, Advanced Lectures (Dagstuhl Research Seminar, March 10-14, 2002)*, volume 2625 of *Lecture Notes in Computer Science*. Springer, 2003.

[Neu90] Edward Neuman. Inequalities involving multivariate convex functions ii. *Proceedings of the American Mathematical Society*, 109(4):965–974, 1990.

[ONH$^+$96] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–11. ACM Press, 1996.

[Pie95] Andrea Pietracaprina. Lower bound for BSPC matrix multiplication. Manuscript, 1995.

[PPS06] Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri. Cache-oblivious simulation of parallel programs. In *Proceedings of the 8th Workshop on Advances in Parallel and Distributed Computational Models*, April 2006.

[Rah02] Naila Rahman. Algorithms for hardware caches and TLB. In Meyer et al. [MSS03], pages 171–192.

[SCD02] Sandeep Sen, Siddhartha Chatterjee, and Neeraj Dumir. Towards a theory of cache-efficient algorithms. *Journal of the ACM*, 49(6):828–858, 2002.

[Sil05] Francesco Silvestri. Simulazione di algoritmi paralleli per il modello D-BSP su una gerarchia di cache ideali. Master's thesis, Department of Information Engineering, University of Padova, Italy, October 2005. Laurea Thesis (in italian).

[Sil06] Francesco Silvestri. On the limits of cache-oblivious matrix transposition. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *Proceedings of the 2nd Symposium of Trustworthy Global Computing*, volume 4661 of *Lecture Notes in Computer Science*, pages 233–243. Springer, 2006.

[Sil08] Francesco Silvestri. On the limits of cache-oblivious rational permutations. *Theoretical Computer Science*, 402(2-3):221–233, 2008.

[SK97] Jop F. Sibeyn and Michael Kaufmann. BSP-like external-memory computation. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity*, pages 229–240, London, UK, 1997. Springer-Verlag.

[SN96] Elizabeth A.M. Shriver and Mark Nodine. An introduction to parallel I/O models and algorithms. In *Input/output in parallel and distributed computer systems*, pages 31–68. Kluwer Academic Publishers, 1996.

[SV87] John E. Savage and Jeffrey Scott Vitter. Parallelism in space-time tradeoffs. In *Advances in Computing Research*, volume 4, pages 117–146. North-Holland, 1987.

[Tok] Tokutek website: *www.tokutek.com*.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[Vit01] Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[VS94] Jeffrey Scott Vitter and Elizabeth A.M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2/3):110–147, 1994.

[Wol96] Michael Wolfe. Parallelizing compilers. *ACM Computer Surveys*, 28(1):261–262, 1996.

[WPD01] R. Clinton Whaley, Antoine Petitet, and Jack Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1-2):3–35, 2001.

[YRP$^+$07] Kamen Yotov, Thomas Roeder, Keshav Pingali, John A. Gunnels, and Fred G. Gustavson. An experimental comparison of cache-oblivious and cache-conscious programs. In *Proceedings of the 19th ACM Symposium on Parallel Algorithms and Architectures*, 2007.