Network-Oblivious Algorithms*

Gianfranco Bilardi[1,2], Andrea Pietracaprina[1], Geppino Pucci[1], and Francesco Silvestri[1]

[1] Dept. of Information Engineering
University of Padova, Italy

[2] IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA

{bilardi,capri,geppo,silvest1}@dei.unipd.it

Abstract

The design of algorithms that can run unchanged yet efficiently on a variety of machines characterized by different degrees of parallelism and communication capabilities is a highly desirable goal. We propose a framework for network-obliviousness *based on a model of computation where the only parameter is the problem's input size. Algorithms are then evaluated on a model with two parameters, capturing parallelism and granularity of communication. We show that, for a wide class of network-oblivious algorithms, optimality in the latter model implies optimality in a block-variant of the Decomposable BSP model, which effectively describes a wide and significant class of parallel platforms. We illustrate our framework by providing optimal network-oblivious algorithms for a few key problems, and also establish some negative results.*

1 Introduction

Communication is a major factor determining the performance of algorithms on current computing systems. Since the relevance of this factor increases with the size of the system, communication will play an even greater role in future years. Reducing the communication requirements of algorithms is then of paramount importance, if they have to run efficiently on physical machines. Recognition of this fact has motivated a large body of results in algorithm design and analysis. While often useful and sometimes deep, these results do not yet provide a coherent and unified theory of the communication requirements of computations. One major obstacle toward such a theory lies in the fact that, prima facie, 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. The outlined situation has been widely recognized and a number of approaches have been proposed to solve it or mitigate it.

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 would be proportional to the product of the number of processing units by the worst-case latency of the target machine [23]. Further 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 and the PRAM models of computations have often been supported with similar arguments. Unfortunately, as argued in [8, 9], latency hiding is not a scalable technique, due to fundamental physical constraints. Hence, parallel slackness does not really solve the communication problem. (Nevertheless, functional and PRAM models are quite valuable and have significantly contributed to the understanding of other dimensions of computing.)

On the other end of the spectrum, we could place the *universality* approach, whose objective is the devel-

*This work was supported in part by MIUR of Italy under project *MAINSTREAM*, and by the EU under the EU/IST Project 15964 *AEOLUS*.

1-4244-0910-1/07/$20.00

opment of machines (nearly) as efficient as any other machine of (nearly) the same cost, at executing any computation (see, *e.g.*, [20, 8, 4]). 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 LPRAM [2], BSP [23] and its refinements [13, 6], LogP [12], QSM [16], and several others. These models aim at capturing features common to most (reasonable) machines, while ignoring features that differ. The hope is that performance of real machines be largely determined by the modeled features, so that optimal algorithms in the proposed model translate into near optimal ones on real machines. A drawback of these models is that they include parameters that affect execution time. Then, in general, efficient algorithms are parameter-aware, since different algorithmic strategies can be more efficient for different values of the parameters. One parameter present in virtually all models is the number of processors. 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, can considerably increase the effectiveness of the model with respect to realistic architectures, such as point-to-point networks, as extensively discussed in [7]. A price is paid in the increased complexity of algorithm design necessary to gain greater efficiency across a larger class of machines. The complications further compound if the hierarchical nature of the memory is also taken into account, so that communication between processors and memories becomes an optimization target as well.

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 [15].

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.

- *Algorithm specification model.* This model, denoted by $M(n)$, is a set of n CPU/memory nodes, called processing elements (PEs), computing in supersteps, and able to send messages to each other. Network-oblivious algorithms will be formulated in this model. The number of PEs, n, is chosen by the algorithm designer exclusively as a function of the problem input (reasonably, n reflects the amount of parallelism of the algorithm at hand).

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

  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 $M(p, B)$, by measuring how close $H_{\mathcal{A}}(n, p, B)$ comes to the minimum communication complexity $H^*(n, p, B)$ achievable by any $M(p, B)$ algorithm solving the same problem as $\mathcal{A}$. Algorithm $\mathcal{A}$ is optimal if $H_{\mathcal{A}}(n, p, B) = O(H^*(n, p, B))$, for a suitable set of (p, B) pairs.

- *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 a block-based variant of the decomposable Bulk Synchronous Parallel model, D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$, where $\boldsymbol{g}$ and $\boldsymbol{B}$ are vectors with logarithmically many parameters in P. Thanks to this multiplicity of parameters, the model can describe reasonably well the behavior of a large class of point-to-point networks. For example, in [6], it is shown how, under suitable assumptions, optimal D-BSP algorithms

do translate into optimal algorithms for multi-dimensional arrays.

Fortunately, as shown in this paper, for a wide and interesting class of network-oblivious algorithms, optimality with respect to the $M(p, B)$ model, for suitable ranges of (p, B) translates into optimality with respect to the D-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 symplify 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 [15]. In the latter, the algorithm specification model is the Random Access Machine; the algorithm evaluation model is the Ideal Cache Model IC(Z, L), a machine with only one level of cache of size Z and line length L; and the machine execution model is a machine with a hierarchy of many caches, each with its own size and line length. In the cache-oblivious context, the simplification in the analysis arises from the fact that, loosely speaking, optimality on IC(Z, L) for all values of Z and L translates into optimality on multilevel hierarchies.

The rest of this paper is organized as follows. In Section 2, we define the three models relevant to the framework and establish the key relations among them. In Section 3, we illustrate the framework by deriving network-oblivious algorithms for key problems, such as matrix multiplication, matrix transposition, FFT, and sorting, showing their optimality for wide ranges of parameters. In the case of matrix transposition, we also present a negative result proving that no network-oblivious algorithm can achieve optimality for a full range of parameters. In the conclusions, we outline directions for further work.

2 The Framework

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

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 $M(n)$ of n *Processing Elements* (PEs), PE_0, ..., PE_{n-1}, each consisting of a CPU and an unbounded local memory. $\mathcal{A}$ consists of a sequence of *labeled supersteps*, with labels in the integer range $[0, \log n)$. (In the paper, all logarithms are taken to the base 2.) 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, *i.e.*, k and j are both in the interval $[\lfloor j2^i/n \rfloor, \lfloor j2^i/n \rfloor + n/2^i)$ [1].

In order to analyze $\mathcal{A}$'s communication complexity on different machines, we introduce the machine model $M(p, B)$, where the parameters p and B are positive integers (for simplicity, powers of two). $M(p, B)$ is essentially an $M(p)$ with a communication cost function parametrized by B, whose processing elements are denoted as PE_j^p, with $0 \leq j < p$, to distinguish them from those of $M(n)$. Specifically, the *block-degree* of a superstep where processing element PE_j^p sends w_{jk} words to PE_k^p, with $0 \leq j, k < p$, is defined as

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

The *communication complexity* of an algorithm is the sum of the block-degrees of its supersteps. In $M(p, B)$, words exchanged between two PEs in a superstep can be envisioned as traveling within *blocks* of fixed size B (in words). The block-degree of a superstep can then be viewed as the maximum number of blocks sent/received by a single PE in that superstep. Hence, the model rewards batched over fine-grained communication. The quantity $h = h(p, 1)$ is also called the *word-degree* of the superstep. Clearly, $\lceil h/B \rceil \leq h(p, B) \leq h$.

A network-oblivious algorithm $\mathcal{A}$ formulated for $M(n)$ can be naturally executed on an $M(p, B)$ machine, for every $1 \leq p \leq n$ and for every B, by stipulating that processing element PE_j^p, $0 \leq j < p$, of $M(p, B)$ will carry out the operations of the n/p consecutively numbered processing elements of $M(n)$ starting with $\mathrm{PE}_{(n/p)j}$. Supersteps with a label $i < \log p$ on $M(n)$ become supersteps with the same label on $M(p, B)$; 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, and let $h^s(n, p, B)$ be the block-degree of the execution of superstep s on $M(p, B)$. 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 $M(p, B)$, for varying p and B.

[1] The results of this paper 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.

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 PEs that will carry out the computation, and the block size of the communication. Similarly, as in the cache-oblivious framework the algorithm designer "knows" that memory accesses with stack distance not greater than Z/L will be serviced by an ideal cache of at least Z/L cache lines [21], in the network-oblivious framework the designer does know that messages between PEs of $M(n)$ whose numbers coincide in the m most significant bits will translate into local memory accesses in a machine with at most 2^m PEs.

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

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 introduce a block variant of the Decomposable BSP (D-BSP) model [7], denoted as a D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$, where $\boldsymbol{g} = (g_0, g_1, \ldots g_{\log P-1})$ and $\boldsymbol{B} = (B_0, B_1, \ldots B_{\log P-1})$. D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$ is essentially an $M(P, \cdot)$ machine featuring various block sizes, where the *communication time* of a superstep is defined to be $h(P, B_i)g_i$, where i is the label of the superstep and $h(P, B_i)$ denotes the block-degree of the superstep with respect to block-size B_i. The communication time T of an algorithm is the sum of the communication times of its supersteps.

Within the D-BSP model, for $0 \leq i < \log P$, a set formed by all the $P/2^i$ PEs whose numbers share the most significant i bits is called an i-cluster. Informally, B_i can be thought of as the block size and g_i as an inverse measure of bandwidth, in time units per block, for i-clusters. For simplicity, we take all 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 suffice to hide latency and smaller time suffices to route message sets with the same word-degree h. Previous versions of D-BSP [13, 6] did not feature blocks but included a latency parameter vector $(\ell_0, \ell_1, \ldots \ell_{\log P-1})$, so that the cost of an i-superstep of word-degree h was $hg_i + \ell_i$. The introduction of blocks makes the model more descriptive of actual platforms and also compensates for the absence of the latency parameters.

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 patterns employed by the algorithm and on the machine parameters. We begin with the following technical lemma.

Lemma 1 *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 non-increasing 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:

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

$$= \sum_{j=0}^{m-1} f_j S_j - \sum_{j=1}^{m-1} f_j S_{j-1} \geq$$

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

□

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

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

Consequently, when executing $\mathcal{A}$ on $M(2^i, 1)$, in any superstep, if PE_j^p sends any words to PE_k^p, then it sends at least b_i words to it.

Definition 3 *Let $\alpha > 0$ be constant. An algorithm $\mathcal{A}$ for $M(n)$ is said to be (α, P)-wise if, for any i such that $1 \leq 2^i \leq P$, we have*

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

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 processors 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 developed in this paper. Quite interestingly, these algorithms achieve optimal performance on D-BSP, as better established in the following theorem.

Theorem 1 *Let $\mathcal{A}$ be an (α, P^*)-wise optimal network-oblivious algorithm for a problem Π, specified for the $M(n)$ model, with i-granularity b_i, for $0 \leq i \leq \log P^*$. Then, $\mathcal{A}$ exhibits asymptotically optimal communication time when executed on any D-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 $D_{\mathcal{A}}(j)$ be the sum of block-degrees of all j-supersteps when $\mathcal{A}$ is executed on D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$. From the hypthesis on the granularity of $\mathcal{A}$, we have that the minimum amount of 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 D-BSP$(P, \boldsymbol{g}, \boldsymbol{B})$ is $T = \sum_{j=0}^{\log P - 1} D_{\mathcal{A}}(j) g_j$. Suppose $\mathcal{A}'$ were an asymptotically faster D-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 $T' < \epsilon T$, 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 that the ratios g_i/B_i are non-increasing, we can apply Lemma 1, 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 $M(2^i, B_i)$ 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 $M(2^i, B_i)$. (Note that in the above inequalities we used the fact that the B_j's are powers of two and that are non-increasing.) □

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.

3 Algorithms for key problems

In this section we develop optimal network-oblivious algorithms for a number of relevant computational problems, namely matrix multiplication, matrix transposition, FFT and sorting. In some cases, optimality requires restrictions on the ranges of some machine and input parameters and, in one case, we will prove that these restrictions are necessary to obtain optimality in a network-oblivious fashion.

3.1 Matrix multiplication

The *n-MM problem* consists of multiplying two $\sqrt{n} \times \sqrt{n}$ matrices using only semiring operations. We first establish a lower bound on the communication complexity of any $M(p, B)$ algorithm for this problem.

Theorem 2 *Let $\mathcal{A}$ be any algorithm solving the n-MM problem on an $M(p, B)$, with $1 < p \leq n$ and $1 \leq B \leq$*

n/p. If initially the inputs are evenly distributed among the p PEs, then the communication complexity of the algorithm is

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

Proof The theorem is an immediate consequence of [18, Lemma 5.1]. □

Next, we describe an optimal network-oblivious algorithm for the n-MM problem. Let X, Y and Z denote, respectively, the two input matrices and the output matrix, and suppose that their entries are evenly distributed among the PEs. For $0 \leq i, j < \sqrt{n}$, we denote by $P(i, j)$ the processing element $\mathrm{PE}_{i\sqrt{n}+j}$ of $M(n)$, and require that such a PE holds $[X]_{ij}$, $[Y]_{ij}$, and $[Z]_{ij}$. The algorithm is based on the following simple recursive strategy. Partition each matrix A ($A = X, Y$ or Z) into four $\sqrt{n}/2 \times \sqrt{n}/2$ quadrants A_{hk}, with $h, k \in \{0, 1\}$. Define $M_{hk\ell} = X_{h\ell} \cdot Y_{\ell k}$, whence $Z_{hk} = M_{hk0} + M_{hk1}$. The algorithm works as follows:

1. Regard the n PEs as partitioned into eight segments $S_{hk\ell}$, with $h, k, \ell \in \{0, 1\}$, of $n/8$ PEs each. Replicate and distribute the inputs so that the entries of $X_{h\ell}$ and $Y_{\ell k}$ be evenly spread among the PEs in $S_{hk\ell}$.

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

3. For $h, k, \ell \in \{0, 1\}$ and $1 \leq i, j \leq n/2$ in parallel, send $[M_{hk\ell}]_{ij}$ to $P(i + h\sqrt{n}/2, j + k\sqrt{n}/2)$.

4. For $1 \leq i, j \leq n$ in parallel, compute $[Z]_{ij}$ in $P(i, j)$ by adding the two values received in the previous step.

Theorem 3 *The communication complexity of the above n-MM algorithm when executed on an $M(p, B)$ machine, with $1 < p \leq n$ and $1 \leq B \leq n/p$, is*

$$H_{\mathrm{MM}}(n, p, B) = O\left(\frac{n}{Bp^{2/3}}\right),$$

which is optimal for all values of p and B in the specified ranges.

Proof The communication complexity of the algorithm is obtained by solving the recurrence $H_{\mathrm{MM}}(n, p, B) = H_{\mathrm{MM}}(n/4, p/8, B) + \Theta\left(n/(Bp)\right)$, for all values p and B specified in the statement of the theorem. □

Note that the above recursive algorithm incurs a memory blow-up of $\Theta\left(p^{1/3}\right)$ when executed on an $M(p, B)$ machine. In [18] it is shown that such a blow-up is necessary to achieve minimum communication time. Using results from the same paper, it can be shown that any $M(p, B)$ algorithm solving the n-MM problem with constant memory blow-up requires $\Omega\left(n/(Bp^{1/2})\right)$ communication complexity. An optimal network-oblivious algorithm with constant memory blow-up can be obtained by still using the above recursive strategy but letting each segment of $n/4$ PEs of the $M(n)$ machine solve two $(n/4)$-MM subproblems sequentially. (More details will be provided in the full version of this extended abstract.)

3.2 Matrix transposition

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

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

Proof (sketch) We use an argument similar to the one employed in [3] to bound from below the I/O complexity of transposition in external memory. For $0 \leq i < p$ we define the *i-th target group* as the set of entries that will be in PE_i^p 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 PEs during the entire execution. Let us index the blocks communicated among the PEs 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 PE_j^p 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 $f(0) = 0$. It can be easily seen that $\mathrm{POT}(0) = n \log(\lceil \sqrt{n}/p \rceil^2)$ and that $\mathrm{POT}(H') = n \log(n/p)$. By reasoning as in [3], it can be shown that, for a suitable constant $c > 0$, the block of index t increases the potential by the quantity

$$\mathrm{POT}(t) - \mathrm{POT}(t-1) \leq cB \log(1 + n/(pB)) \stackrel{\text{def}}{=} \nabla\mathrm{POT}.$$

Therefore,

$$\begin{aligned} p \cdot H \cdot \nabla\mathrm{POT} &\geq \mathrm{POT}(H') - \mathrm{POT}(0) \\ &= n \log(n/p) - n \log\left(\lceil \sqrt{n}/p \rceil^2\right), \end{aligned}$$

and the theorem follows. □

We now describe a network-oblivious algorithm for the n-MT problem on $M(n)$. 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 $u = (u_{d-1} \ldots u_0)$ and $v = (v_{d-1} \ldots v_0)$ we let $u \bowtie v$ denote their bitwise interleaving, i.e., $u \bowtie v = u_{d-1} v_{d-1} \ldots u_0 v_0$. Let A be the $\sqrt{n} \times \sqrt{n}$ input matrix and let $P(i, j)$ denote the processing element $\mathrm{PE}_{i\sqrt{n}+j}$, which initially holds $[A]_{ij}$ and at the end will hold $[A^T]_{ij}$, with $0 \leq i, j < \sqrt{n}$. The algorithm consists of a 1-superstep followed by a 0-superstep.

1. For $0 \leq i, j < \sqrt{n}$, $P(i, j)$ sends $[A]_{ij}$ to the PE_q, where $q = \mathcal{B}^{-1}(\mathcal{B}(i) \bowtie \mathcal{B}(j))$;
2. For $0 \leq q < n$, if the PE_q has received entry $[A]_{ij}$ 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 [11]. The communication complexity of the above algorithm, whose correctness is immediate, is established in the following theorem.

Theorem 5 *The communication complexity of the above n-MT algorithm when executed on an $M(p, B)$ machine, with $1 < p \leq n$ and $1 \leq B \leq \sqrt{n/p}$, is*

$$H_{\mathrm{MT}}(n, p, B) = O\left(\frac{n}{Bp}\right),$$

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

Proof (sketch) For simplicity, when defining $M(p, B)$ we assumed B to be a power of two. With this assumption, it can be easily shown that in the first superstep each segment of B consecutive PEs of $M(n)$ sends their entries to PEs belonging to a segment of size at most $B^2 \leq n/p$. Similarly, it can be shown that in the second superstep each segment of B PEs of $M(n)$ receives all their data from PEs belonging to a segment of size $2B^2$. Therefore, when the algorithm is executed on $M(p, B)$ with $B \leq \sqrt{n/p}$, the block-degree of the communication involved in each superstep is $O(n/(pB))$. Optimality follows from Theorem 4. (The theorem holds also when B is not a power of two, and the proof needs only slight modifications whose details will be provided in the full version.) □

The restriction on the range of 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 [15]. 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 [22]. We say that an $M(p, B)$ algorithm is *full* if in each of its supersteps all processors send/receive the same number of blocks, with each block containing $\Theta(B)$ data words.

Theorem 6 *There cannot exist a network-oblivious algorithm $\mathcal{A}$ for the n-MT problem such that, for every $1 < p \leq n$ and $1 \leq B \leq n/p$, its execution on $M(p, B)$ yields a full algorithm whose communication complexity matches the one stated in Theorem 4.*

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 $M(p_1, B_1)$ and $M(p_2, B_2)$, with $p_1 > p_2$. Since the two executions are full by hypothesis, and every data communicated in the $M(p_2, B_2)$ execution must be also communicated in the $M(p_1, B_1)$ execution, we must have that $B_1 p_1 H_1 = \Omega(B_2 p_2 H_2)$, whence $B_1 p_1 H_1 / (B_2 p_2 H_2) = \Omega(1)$ (the asymptotics is w.r.t. n). Now, let us choose $p_1 = n/2$, $B_1 = 1$, $p_2 = \Theta(n^\epsilon)$, with ϵ constant, $0 < \epsilon < 1$, and $B_2 = \Theta(n/p_2)$. Then, by Theorem 4 we have that $(B_1 p_1 H_1)/(B_2 p_2 H_2) = \Theta(1/\log n) = o(1)$, 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 7 *For every $1 < p \leq n$ and $1 \leq B \leq n/p$, there exists a full $M(p, B)$ algorithm for the n-MT problem whose communication complexity matches the one stated in Theorem 4.*

Proof (sketch) The algorithm is obtained by suitably parallelizing the recursive strategy of [3] employed for solving the n-MT problem in external memory. The $M(p, B)$ algorithm makes explicit use of parameters p and B. Full details will be given in the full version. □

3.3 FFT

The *n-FFT problem* consists of computing an n-input FFT dag.

Theorem 8 *Let $\mathcal{A}$ be any algorithm solving the n-FFT problem on an $M(p, B)$ with $1 < p \leq n$ and $1 \leq B \leq n/p$. If the inputs are initially evenly distributed among the p PEs, then the communication complexity of the algorithm is*

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

Proof (sketch) The result follows by a suitable adaptation of the similar lower bound derived for the LPRAM model in [2]. □

The network-oblivious n-FFT algorithm on $M(n)$ 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 [1]. 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. 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 a matrix transposition. We have:

Theorem 9 *The communication complexity of the above n-FFT algorithm when executed on an $M(p, B)$ machine, with $1 < p \leq n$ and $1 \leq B \leq \sqrt{n/p}$, is*

$$H_{\text{FFT}}(n, p, B) = O\left(\frac{n}{pB}\frac{\log n}{\log(1+n/p)}\right),$$

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

Proof When the algorithm is run on an $M(p, B)$ machine with $p \leq \sqrt{n}$, each subdag is computed locally by a single PE, and in this case, we must account only for the transposition step, which entails each PE sending and receiving $\Theta(n/(pB))$ blocks. Otherwise, the total communication complexity of the algorithm obeys the recurrence $H_{\text{FFT}}(n, p, B) = 2H_{\text{FFT}}(\sqrt{n}, p/\sqrt{n}, B) + \Theta(n/(pB))$, whose solution yields the stated result. □

The small-block assumption needed to prove the optimality of the n-FFT network-oblivious algorithm derives from the use of matrix transposition. 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.

3.4 Sorting

The *n-Sort problem* consists of sorting n keys. We require that the inputs be evenly distributed among the PEs, and that, at the end, the keys held by the i-th PE are all less 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 $M(p, B)$ algorithm for this problem.

Theorem 10 *Let $\mathcal{A}$ be an algorithm solving the n-Sort problem on an $M(p, B)$ with $1 < p \leq n$ and $1 \leq B \leq n/p$. If the word-degree of each superstep is $O(n/p)$, then the communication complexity of $\mathcal{A}$ is:*

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

Proof The theorem follows by dividing by B the lower bound for $M(p, 1)$ given in [17]. □

We now describe a network-oblivious algorithm for n-Sort based on a recursive version of the *Columnsort* algorithm as described in [19]. We regard both the n keys and the PEs of $M(n)$ as arranged in an $s \times r$ 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 (in Phase 5 adjacent rows are sorted in reverse order). 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.

Observe that when executed on an $M(p, B)$ the above algorithm has the property that the word-degree of each superstep is $O(n/p)$, hence the lower bound proved before applies to it. We have:

Theorem 11 *The communication complexity of the above n-Sort algorithm when executed on an $M(p, B)$ machine with $1 < p \leq n$ and $1 \leq B \leq \sqrt{n/p}$ is*

$$H_{\text{Sort}}(n, p, B) = O\left(\frac{n}{pB}\left(\frac{\log n}{\log(1+n/p)}\right)^{\log_{3/2} 4}\right),$$

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

Proof (sketch) The transposition performed in Phases 2 and 4 can be implemented by separately transposing each $s \times s$ submatrix and then suitably permuting the blocks among the PEs. It is easy to show that by employing the network-oblivious algorithm described in Subsection 3.2 for each submatrix and by using the stated upper bound on B, the transposition of the $s \times r$ matrix has communication complexity $\Theta(n/pB)$. The stated communication complexity of the entire algorithm is obtained by solving the recurrence $H_{\text{Sort}}(n,p,B) = 4H_{\text{Sort}}\left(n^{2/3}, p/n^{1/3}, B\right) + \Theta(n/pB)$. □

We conjecture that a similar result can be obtained by adapting other known sorting algorithms such as, for example, the one in [17].

4 Conclusions

In this paper, we have introduced a framework to explore the design of algorithms that, without resorting to parameters used for tuning on the target platform, can execute efficiently on machines with different bandwidth characteristics. In this framework, an optimal algorithm written for n "virtual" processors is one that minimizes a suitable measure of interprocessor communication, irrespective of the number p of "physical" processors used for the execution, as long as a suitable mapping is assumed between virtual and physical processors. When the algorithm is actually executed on a machine of p interconnected nodes, optimality in the above sense translates into minimization of the communication between certain submachines and the rest of the system, which leads to good performance.

In the network-oblivious approach to algorithm design and implementation, any intelligence about the locality of communication is embedded in the linear ordering of the virtual PEs. Clearly, the mapping of virtual processors to physical processors has to be based on the bandwidth properties of the target machine. When the machine is modeled as a D-BSP system, the mapping is indeed specified by the chosen correspondence between the PEs of the machine and the PEs of D-BSP. For example, the results of [6], concerning the translation of algorithms that are optimal for D-BSP into algorithms that are optimal for multi-dimensional arrays, assume a natural mapping obtained by a recursive bisection of the array into equal-size subarrays. The efficiency of recursive bisection for other types of networks deserves further investigation. In all cases, an efficient processor mapping needs to be known to and to be enforced by the operating and run-time systems, for the efficiency of a network-oblivious algorithms to be properly exploited.

We have shown that a number of key problems admit optimal network-oblivious algorithms. It is too early to say how wide the class of these problems is without further exploration of both positive and negative cases. In the analogous context of cache-obliviousness, a number of optimal algorithms have been developed. In comparison, negative results are quite few (*e.g.*, [5, 10, 22]), and techniques of general applicability have not yet emerged. However, the pursuit of oblivious algorithms appears worthwhile even when the outcome is a proof that no such algorithm can be optimal on all target machines. Indeed, the analysis behind such a result is likely to reveal what kind of adaptivity to the target machine is essential to optimal performance. In turn, this can provide valuable insights, particularly to the development of adaptive libraries, based on parametrized implementations that can be tuned to best the properties of the target platform, in the spirit of the ATLAS library for linear algerbra [24] and of the FFTW library for Fourier transforms [14].

The present work can be naturally extended in several directions, some of which are briefly outlined next.

- Naturally, the design of efficient network-oblivious algorithms can be considered for many key problems, beyhond the few case studies of this paper.
- An interesting, but challenging goal is the development of lower-bound techniques that limit the level of optimality that network-oblivious algorithm can reach on certain classes of target platforms.
- It would be useful to weaken the assumptions under which an optimal network-oblivious algorithm achieves optimal time in the D-BSP model.
- It would be useful to identify other classes of machines, for which network-oblivious optimality translates into optimal time.
- Another natural question to investigate is how to determine an efficient virtual-to-physical processor mapping, for a network with arbitrary topology.

References

[1] A. Aggarwal, A. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. of the 28th IEEE Symp. on Foundations of Computer Science*, pages 204–216, 1987.

[2] A. Aggarwal, A. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990. See also *Proc. of ICALP '88*, 1–17.

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

[4] S. Bhatt, G. Bilardi, and G. Pucci. Area-universal circuits with constant slowdown. In *Proc. of the 18th Int. Conference on Advanced Research in VLSI*, pages 89–98, 1999.

[5] G. Bilardi and E. Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *Proc. of 28th Int. Colloquium on Automata, Languages and Programming*, LNCS 2076, pages 128–139, 2001.

[6] G. Bilardi, A. Pietracaprina, and G. Pucci. A quantitative measure of portability with application to bandwidth-latency models for parallel computing. In *Proc. of EUROPAR 99*, LNCS 1685, pages 543–551, Sept. 1999.

[7] G. Bilardi, A. Pietracaprina, and G. Pucci. Decomposable BSP: A bandwidth-latency model for parallel and hierarchical computation. In J. Reif and S. Rajasekaran, editors, *Handbook of Parallel Computing: Models, Algorithms and Applications*, pages 277–315. CRC Press, 2007. To appear.

[8] G. Bilardi and F. Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part I, upper bounds. *Theory of Computing Systems*, 30:523–546, 1997.

[9] G. Bilardi and F. Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems*, 32:531–559, 1999.

[10] G. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proc. of the 35th ACM Symp. on Theory of Computing*, pages 307–315, June 2003.

[11] S. Chatterjee, A. Lebeck, P. Patnala, and M. Thottethodi. Recursive array layouts and fast matrix multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 13(11):1105–1123, 2002.

[12] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, Nov. 1996.

[13] P. De la Torre and C. Kruskal. Submachine locality in the bulk synchronous setting. In *Proc. of EUROPAR 96*, LNCS 1124, pages 352–358, Aug. 1996.

[14] M.Frigo and S.G.Johnson. FFTW: An Adaptive Software Architecture for the FFT. *ICASSP*, pp. 1381-1384, Seattle, WA, 1998.

[15] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. of 40th IEEE Symp. on Foundations of Computer Science*, pages 285–298, 1999.

[16] P. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model serve as a bridging-model for parallel computation? *Theory of Computing Systems*, 32(3):327–359, 1999.

[17] M. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[18] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[19] F. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[20] C. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. on Computers*, C-34(10):892–901, Oct. 1985.

[21] R. Mattson, J.Gecsei, D. Slutz, and I. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[22] F. Silvestri. On the limits of cache-oblivious matrix transposition. In *Proc. of 2nd Symp. of Trustworthy Global Computing*, pages 147–157, Nov. 2006.

[23] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

[24] R.C.Whaley and J.J.Dongarra. Automatically Tuned Linear Algebra Software. http://www.netlib.org/atlas/index.html