Network-Oblivious Algorithms

GIANFRANCO BILARDI, ANDREA PIETRACAPRINA, GEPPINO PUCCI, MICHELE SCQUIZZATO, and FRANCESCO SILVESTRI, University of Padova

A framework is proposed for the design and analysis of network-oblivious algorithms, namely algorithms that can run unchanged, yet efficiently, on a variety of machines characterized by different degrees of parallelism and communication capabilities. The framework prescribes that a network-oblivious algorithm be specified on a parallel model of computation where the only parameter is the problem's input size, and then evaluated on a model with two parameters, capturing parallelism granularity and communication latency. It is shown that for a wide class of network-oblivious algorithms, optimality in the latter model implies optimality in the decomposable bulk synchronous parallel model, which is known to effectively describe a wide and significant class of parallel platforms. The proposed framework can be regarded as an attempt to port the notion of obliviousness, well established in the context of cache hierarchies, to the realm of parallel computation. Its effectiveness is illustrated by providing optimal network-oblivious algorithms for a number of key problems. Some limitations of the oblivious approach are also discussed.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Communication, models of computation, network locality, oblivious algorithms, parallel algorithms

ACM Reference Format:
Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, Michele Scquizzato, and Francesco Silvestri. 2016. Network-oblivious algorithms J. ACM 63, 1, Article 3 (March 2016), 36 pages.
DOI: http://dx.doi.org/10.1145/2812804

1. INTRODUCTION

Communication plays a major role in determining the performance of algorithms on current computing systems and has a considerable impact on energy consumption. Since the relevance of communication increases with the size of the system, it is expected to play an even greater role in the future. Motivated by this scenario, a large body of results have been devised concerning the design and analysis of communication-efficient algorithms. Although often useful and deep, these results do not yet provide a coherent and unified theory of the communication requirements of computations. One

This work was supported in part by MIUR of Italy under project AMANDA, and by the University of Padova under projects STPD08JA32, CPDA121378/12, and CPDA152255/15. F. Silvestri was also supported by the European Research Council under grant agreement 614331. A preliminary version of this article appeared in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*.
Authors' addresses: G. Bilardi, A. Pietracaprina, and G. Pucci, Department of Information Engineering, University of Padova, 35131 Padova, Italy; emails: {bilardi, capri, geppo}@dei.unipd.it; M. Scquizzato (current address), Department of Computer Science, University of Houston, Houston, TX 77204-3010; email: michele@cs.uh.edu; F. Silvestri (current address), IT University of Copenhagen, Rued Langgaards Vej 7, 2300 København S, Denmark; email: fras@itu.dk.

DOI: http://dx.doi.org/10.1145/2812804

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. Hence, the design, optimization, and analysis of algorithms 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 to mitigate it.

On one end of the spectrum, we have the *parallel slackness* approach, based on the assumption that as long as a sufficient amount of parallelism is exhibited, general and automatic latency-hiding techniques can be deployed to achieve an efficient execution. Broadly speaking, the required algorithmic parallelism should be at least proportional to the product of the number of processing units and the worst-case latency of the target machine [Valiant 1990]. Further assuming that this amount of parallelism is available in computations of practical interest, algorithm design can dispense altogether with communication concerns and focus on the maximization of parallelism. The functional/dataflow and the PRAM models of computations have often been supported with similar arguments. Unfortunately, as argued in Bilardi and Preparata [1995, 1997, 1999], latency hiding is not a scalable technique due to fundamental physical constraints (namely, upper bounds to the speed of messages and lower bounds to the size of devices). 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 development of machines (nearly) as efficient as any other machine of (nearly) the same cost, at executing any computation (e.g., see Leiserson [1985], Bilardi and Preparata [1995], Bhatt et al. [2008], and Bilardi and Pucci [2011]). 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. Although this appears as a perspective worthy of investigation, at the present stage neither the known theoretical results nor the trends of commercially available platforms indicate an imminent strong convergence.

In the middle of the spectrum, a variety of computational models proposed in the literature can be viewed as variants of an approach aiming at realizing an *efficiency/portability/design-complexity trade-off* [Bilardi and Pietracaprina 2011]. Well-known examples of these models are LPRAM [Aggarwal et al. 1990], DRAM [Leiserson and Maggs 1988], BSP [Valiant 1990] and its refinements (e.g., D-BSP [de la Torre and Kruskal 1996; Bilardi et al. 2007a], BSP* [Bäumker et al. 1998], E-BSP [Juurlink and Wijshoff 1998], and BSPRAM [Tiskin 1998]), LogP [Culler et al. 1996], QSM [Gibbons et al. 1999], MapReduce [Karloff et al. 2010; Pietracaprina et al. 2012], 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 is 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, as 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 Bilardi et al. [2007a]. 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 communication between processors and memories becomes an optimization target as well.

It is natural to wonder whether, at least for some problems, parallel algorithms can be designed that, although independent of any machine/model parameters, are nevertheless efficient for wide ranges of these parameters. In other words, we are interested in exploring the world of efficient *network-oblivious* algorithms with a spirit similar to the one that motivated the development of efficient *cache-oblivious* algorithms [Frigo et al. 2012]. In this article, we define the notion of network-oblivious algorithms and propose a framework for their design and analysis. Our framework is based on three models of computation, each with a different role, as briefly outlined next.

The three models are based on a common organization consisting of a set of CPU/memory nodes communicating through some interconnection. Inspired by the bulk synchronous parallel (BSP) model and its aforementioned variants, we assume that the computation proceeds as a sequence of supersteps, where in a superstep each node performs local computation and sends/receives messages to/from other nodes, which will be consumed in the subsequent superstep. Each message occupies a constant number of words.

The first model of our framework (specification model) is used to specify network-oblivious algorithms. In this model, the number of CPU/memory nodes, referred to as virtual processors, is a function $v(n)$ of the input size and captures the amount of parallelism exhibited by the algorithm. The second model (evaluation model) is the basis for analyzing the performance of network-oblivious algorithms on different machines. It is characterized by two parameters, independent of the input: the number p of CPU/memory nodes, simply referred to as processors in this context, and a fixed latency/synchronization cost σ per superstep. The communication complexity of an algorithm is defined in this model as a function of p and σ. Finally, the third model (execution machine model) enriches the evaluation model by replacing parameter σ with two independent parameter vectors of size logarithmic in the number of processors, which represent, respectively, the inverse of the bandwidth and the latency costs of suitable nested subsets of processors. In this model, the communication time of an algorithm is analyzed as a function of p and of the two parameter vectors. In fact, the execution machine model of our framework coincides with the decomposable bulk synchronous parallel (D-BSP) model [de la Torre and Kruskal 1996; Bilardi et al. 2007a], which is known to describe reasonably well the behavior of a large class of point-to-point networks by capturing their hierarchical structure [Bilardi et al. 1999].

A network-oblivious algorithm is designed in the specification model but can be run on the evaluation or execution machine models by letting each processor of these models carry out the work of a prespecified set of virtual processors. The main contribution of this article is an optimality theorem showing that for a wide and interesting class of network-oblivious algorithms, which satisfy some technical conditions and whose communication requirements depend only on the input size and not on the specific input instance, optimality in the evaluation model automatically translates into optimality in the D-BSP model for suitable ranges of the models' parameters. It is this circumstance that motivates the introduction of the intermediate evaluation model, which simplifies the analysis of network-oblivious algorithms, while effectively bridging the performance analysis to the more realistic D-BSP model.

To illustrate the potentiality of the framework, we devise network-oblivious algorithms for several fundamental problems, such as matrix multiplication, fast Fourier transform (FFT), comparison-based sorting, and a class of stencil computations. In all cases, except for stencil computations, we show, through the optimality theorem, that these algorithms are optimal when executed on the D-BSP for wide ranges of the parameters. Unfortunately, there exist problems for which optimality on the D-BSP cannot be attained in a network-oblivious fashion for wide ranges of parameters. We show that this is the case for the broadcast problem.

To help place our network-oblivious framework into perspective, it may be useful to compare it to the well-established sequential cache-oblivious framework [Frigo et al. 2012]. In the latter, the specification model is the random access machine; the evaluation model is the ideal cache model IC($\mathcal{M}, \mathcal{B}$), with only one level of cache of size $\mathcal{M}$ and line length $\mathcal{B}$; and the execution machine 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 IC($\mathcal{M}, \mathcal{B}$), for all values of $\mathcal{M}$ and $\mathcal{B}$, translates into optimality on multilevel hierarchies.

The notion of obliviousness in parallel settings has been addressed by several research works. In a preliminary version of the current work [Bilardi et al. 2007b] (see also [Herley 2011]), we proposed a framework similar to the one presented here, where messages are packed in blocks whose fixed size is a parameter of the evaluation and execution machine models. Although blocked communication may be preferable for models where the memory and communication hierarchies are seamlessly integrated, such as multicores, latency-based models like the one used here are equivalent for that scenario and also capture the case when communication is accomplished through a point-to-point network. In recent years, obliviousness in parallel platforms has been explored in the context of multicore architectures, where processing units communicate through a multilevel cache hierarchy at the top of a shared memory [Chowdhury et al. 2013; Cole and Ramachandran 2010, 2012a, 2012b; Blelloch et al. 2010, 2011]. Although these works have significantly contributed to the development of oblivious algorithmics, the proposed results exploit the blocked and shared-memory nature of the communication system and thus do not suit platforms with distributed memories and point-to-point networks, for which our model of obliviousness is more appropriate. Chowdhury et al. [2013] introduced a multilevel hierarchical model for multicores and the notion of a *multicore-oblivious* algorithm for this model. A multicore-oblivious algorithm is specified with no mention of any machine parameters, such as the number of cores, number of cache levels, cache sizes, and block lengths, but it may include some simple hints to the runtime scheduler, like space requirements. These hints are then used by a suitable scheduler, aware of the multicore parameters, to efficiently schedule the algorithm on multicores with a multilevel cache hierarchy and any given number of cores. Cole and Ramachandran [2010, 2012a, 2012b] presented *resource-oblivious* algorithms: these are multicore-oblivious algorithms with no hints, which can be efficiently executed on two-level memory multicores by schedulers that are not aware of the multicore parameters. In Blelloch et al. [2010, 2011], it is shown that multicore resource-oblivious algorithms can be analyzed independently of both the parallel machine and the scheduler. In the first work, the claim is shown for hierarchies of only private or only shared caches. In the second work, the result is extended to a multilevel hierarchical multicore by introducing a parallel version of the cache-oblivious framework of Frigo et al. [2012], named the parallel cache-oblivious model, and a scheduler for oblivious irregular computations. In contrast to these oblivious approaches, Valiant [2011] studies parallel algorithms for multicore architectures advocating a parameter-aware design of portable algorithms. The work presents optimal algorithms for multi-BSP, a bridging model for multicore architectures that exhibits a hierarchical structure akin to that of our execution machine model.

The rest of the article is organized as follows. In Section 2, we formally define the three models relevant to the framework, and in Section 3, we prove the optimality theorem mentioned earlier. In Section 4, we present the network-oblivious algorithms for matrix multiplication, FFT, comparison-based sorting, and stencil computations. We also discuss the impossibility result regarding the broadcast problem. Section 5 extends the optimality theorem by presenting a less powerful version, which, however, applies to a wider class of algorithms. Section 6 concludes the article with some final remarks. Appendix A provides a table that summarizes the main notations and symbols used in the article.

2. THE FRAMEWORK

We begin by introducing a parallel machine model $M(v)$, which underlies the specification, the evaluation, and the execution components of our framework. Specifically, $M(v)$ consists of a set of v processing elements, denoted by $\mathrm{P}_0, \mathrm{P}_1, \ldots, \mathrm{P}_{v-1}$, each equipped with a CPU and an unbounded local memory, which communicate through some interconnection. For simplicity, throughout this article, we assume that the number of processing elements is always a power of 2. The instruction set of each CPU is essentially that of a standard random access machine, augmented with the three primitives $\mathtt{sync}(i)$, $\mathtt{send}(m, q)$, and $\mathtt{receive}()$. Furthermore, each P_r has access to its own index r and to the number v of processing elements. When P_r invokes primitive $\mathtt{sync}(i)$, with i in the integer range $[0, \log v)$, a barrier synchronization is enforced among the $v/2^i$ processing elements whose indices share the i most significant bits with r.[1] When P_r invokes $\mathtt{send}(m, q)$, with $0 \leq q < v$, a constant-size message m is sent to P_q; the message will be available in P_q only after a $\mathtt{sync}(k)$, where k is not bigger than the number of most significant bits shared by r and q. On the other hand, the function $\mathtt{receive}()$ returns an element in the set of messages received up to the preceding barrier and removes it from the set.

In this article, we restrict our attention to algorithms where the sequence of labels of the sync operations is the same for all processing elements, and where the last operation executed by each processing element is a sync.[2] In this case, the execution of an algorithm can be viewed as a sequence of *supersteps*, where a superstep consists of all operations performed between two consecutive sync operations, including the second of these sync operations. Supersteps are labeled by the index of their terminating sync operation—namely, a superstep terminating with $\mathtt{sync}(i)$ will be referred to as an *i-superstep*, for $0 \leq i < \log v$. Furthermore, we make the reasonable assumption that in an i-superstep, each P_r can send messages only to processing elements whose index agrees with r in the i most significant bits—that is, message exchange occurs only between processors belonging to the same synchronization subset. We observe that the results of this work would hold even if, in the various models considered, synchronizations were not explicitly labeled. However, explicit labels can help reduce synchronization costs. For instance, they become crucial for the efficient execution of the algorithms on point-to-point networks, especially those of large diameter.

In a more intuitive formulation, processing elements in $M(v)$ can be conceptually envisioned as the leaves of a complete binary tree of height $\log v$. When a processing

[1]For notational convenience, throughout this article we use $\log x$ to mean $\max\{1, \log_2 x\}$.

[2]As we will see in the article, several algorithms naturally comply or can easily be adapted to comply with these restrictions. Nevertheless, a less restrictive family of algorithms for $M(v)$ can be defined by allowing processing elements to feature different traces of labels of their sync operations, still ensuring termination. The exploration of the potentialities of these algorithms is left for future research.

element P_r invokes the primitive sync(i), all processing elements belonging to the subtree rooted at the ancestor of P_r at level i are synchronized. Similarly, an i-superstep imposes that message exchange and synchronization are performed independently within the groups of leaves associated with the different subtrees rooted at level i. However, we remark that the tree is a conceptual construction and that $M(v)$ should not be confused with a tree network, as no assumption is made on the specific communication infrastructure between processing elements.

Consider an $M(v)$-algorithm $\mathcal{A}$ satisfying the preceding restrictions. For a given input instance I, we use $L^i_{\mathcal{A}}(I)$ to denote the set of i-supersteps executed by $\mathcal{A}$ on input I, and define $S^i_{\mathcal{A}}(I) = |L^i_{\mathcal{A}}(I)|$, for $0 \leq i < \log v$. Algorithm $\mathcal{A}$ can be naturally and automatically adapted to execute on a smaller machine $M(2^j)$, with $0 \leq j < \log v$, by stipulating that processing element P_r of $M(2^j)$ will carry out the operations of the $v/2^j$ consecutively numbered processing elements of $M(v)$ starting with $\mathrm{P}_{r(v/p)}$, for each $0 \leq r < 2^j$. We call this adaptation *folding*. Under folding, supersteps with a label $i < j$ on $M(v)$ become supersteps with the same label on $M(2^j)$, whereas supersteps with label $i \geq j$ on $M(v)$ become local computation on $M(2^j)$. Hence, when considering the communication occurring in the execution of $\mathcal{A}$ on $M(2^j)$, the set $L^i_{\mathcal{A}}(I)$ is relevant as long as $i < j$.

A *network-oblivious algorithm* $\mathcal{A}$ for a given computational problem Π is designed on $M(v(n))$, referred to as *specification model*, where the number $v(n)$ of processing elements, which is a function of the input size, is selected as part of the algorithm design. The processing elements are called *virtual processors* and are denoted by $\mathrm{VP}_0, \mathrm{VP}_1, \ldots, \mathrm{VP}_{v(n)-1}$ to distinguish them from the processing elements of the other two models. Since the folding mechanism illustrated earlier enables $\mathcal{A}$ to be executed on a smaller machine, the design effort can be kept focussed on just one convenient virtual machine size, oblivious to the actual number of processors on which the algorithm will be executed.

Although a network-oblivious algorithm is specified for a large virtual machine, it is useful to analyze its communication requirements on machines with reduced degrees of parallelism. For these purposes, we introduce the *evaluation model* $M(p, \sigma)$, where $p \geq 1$ is a power of 2 and $\sigma \geq 0$, which is essentially an $M(p)$ where the additional parameter σ is used to account for the latency plus synchronization cost of each superstep. The processing elements of $M(p, \sigma)$ are called *processors* and are denoted by $\mathrm{P}_0, \mathrm{P}_1, \ldots, \mathrm{P}_{p-1}$. Consider the execution of an algorithm $\mathcal{A}$ on $M(p, \sigma)$ for a given input I. For each superstep s, the metric of interest that we use to evaluate the communication requirements of the algorithm is the maximum number of messages $h^s_{\mathcal{A}}(I, p)$ sent/destined by/to any processor in that superstep. Thus, the set of messages exchanged in the superstep can be viewed as forming an $h^s_{\mathcal{A}}(I, p)$*-relation*, where $h^s_{\mathcal{A}}(I, p)$ is often referred to as the *degree* of the relation. In the evaluation model, the communication cost of a superstep of degree h is defined as $h + \sigma$, and it is independent of the superstep's label. For our purposes, it is convenient to consider the cumulative degree of all i-supersteps, for $0 \leq i < \log p$:

$$F^i_{\mathcal{A}}(I, p) = \sum_{s \in L^i_{\mathcal{A}}(I)} h^s_{\mathcal{A}}(I, p).$$

Then, the *communication complexity* of $\mathcal{A}$ on $M(p, \sigma)$ is defined as

$$H_{\mathcal{A}}(n, p, \sigma) = \max_{I:|I|=n} \left\{ \sum_{i=0}^{\log p - 1} \left(F^i_{\mathcal{A}}(I, p) + S^i_{\mathcal{A}}(I) \cdot \sigma \right) \right\}. \quad (1)$$

We observe that the evaluation model with this performance metric coincides with the BSP model [Valiant 1990] where the bandwidth parameter g is set to 1 and the latency/synchronization parameter ℓ is set to σ.

Next, we turn our attention to the last model used in the framework, called the *execution machine model*, which represents the machines where network-oblivious algorithms are actually executed. We focus on parallel machines whose underlying interconnection exhibits a hierarchical structure and use the D-BSP model [de la Torre and Kruskal 1996; Bilardi et al. 2007a] as our execution machine model. A D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$, with $\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log p-1})$ and $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log p-1})$, is an $M(p)$ where the cost of an i-superstep depends on parameters g_i and ℓ_i, for $0 \le i < \log p$. The processing elements, called *processors* and denoted by $\mathrm{P}_0, \mathrm{P}_1, \ldots, \mathrm{P}_{p-1}$ as in the evaluation model, are partitioned into nested clusters: for $0 \le i \le \log p$, a set formed by all the $p/2^i$ processors whose indices share the most significant i bits is called an *i-cluster*. As for the case of the specification model, if we envision a conceptual tree-like organization with the p D-BSP processors at the leaves, then i-clusters correspond to the leaves of subtrees rooted at level i. Observe that during an i-superstep, each processor communicates only with processors of its i-cluster. For the communication within an i-cluster, parameter ℓ_i represents the latency plus synchronization cost (in time units), whereas g_i represents an inverse measure of bandwidth (in units of time per message). By importing the notation adopted in the evaluation model, we define the *communication time* of an algorithm $\mathcal{A}$ on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ as

$$D_{\mathcal{A}}(n, p, \boldsymbol{g}, \boldsymbol{\ell}) = \max_{I:|I|=n} \left\{ \sum_{i=0}^{\log p-1} \left(F^i_{\mathcal{A}}(I, p) g_i + S^i_{\mathcal{A}}(I) \ell_i \right) \right\}. \tag{2}$$

The results in Bilardi et al. [1999] provide evidence that D-BSP is an effective machine model, as its hierarchical structure and its $2\log p$ bandwidth and latency parameters are sufficient to capture reasonably well the cost of both balanced and unbalanced communication for a large class of point-to-point networks [Bilardi et al. 1999].

Through the folding mechanism discussed earlier, any network-oblivious algorithm $\mathcal{A}$ specified on $M(v(n))$ can be transformed into an algorithm for $M(p)$ with $p < v(n)$, and hence into an algorithm for $M(p, \sigma)$ or D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$. In this case, the quantities $H_{\mathcal{A}}(n, p, \sigma)$ and $D_{\mathcal{A}}(n, p, \boldsymbol{g}, \boldsymbol{\ell})$ denote, respectively, the communication complexity and communication time of the folded algorithm. Moreover, since algorithms designed on the evaluation model $M(p, \sigma)$ or on the execution machine model D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ can be regarded as algorithms for $M(p)$, once the parameters σ or $\boldsymbol{g}$ and $\boldsymbol{\ell}$ are fixed, we can also analyze the communication complexities/times of their foldings on smaller machines (i.e., machines with 2^j processors, for any $0 \le j < \log p$). These relations among the models are crucial for the effective exploitation of our framework.

The following definitions establish useful notions of optimality for the two complexity measures introduced earlier relative to the evaluation and execution machine models. For each measure, optimality is defined with respect to a class of algorithms, whose actual role will be made clear later in the article. Let $\mathscr{C}$ denote a class of algorithms, solving a given problem Π.

Definition 2.1. Let $0 < \beta \le 1$. An $M(p, \sigma)$-algorithm $\mathcal{B} \in \mathscr{C}$ is *β-optimal on $M(p, \sigma)$ with respect to $\mathscr{C}$* if for each $M(p, \sigma)$-algorithm $\mathcal{B}' \in \mathscr{C}$ and for each n,

$$H_{\mathcal{B}}(n, p, \sigma) \le \frac{1}{\beta} H_{\mathcal{B}'}(n, p, \sigma).$$

Definition 2.2. Let $0 < \beta \le 1$. A D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$-algorithm $\mathcal{B} \in \mathscr{C}$ is *β-optimal on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ with respect to $\mathscr{C}$* if for each D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$-algorithm $\mathcal{B}' \in \mathscr{C}$ and for each n,

$$D_{\mathcal{B}}(n, p, \boldsymbol{g}, \boldsymbol{\ell}) \leq \frac{1}{\beta} D_{\mathcal{B}'}(n, p, \boldsymbol{g}, \boldsymbol{\ell}).$$

Note that the preceding definitions do not require β to be a constant: intuitively, larger values of β correspond to higher degrees of optimality.

3. OPTIMALITY THEOREM FOR STATIC ALGORITHMS

In this section, we show that for a certain class of network-oblivious algorithms, β-optimality in the evaluation model, for suitable ranges of parameters p and σ, translates into β'-optimality in the execution machine model, for some $\beta' = \Theta(\beta)$ and suitable ranges of parameters p, $\boldsymbol{g}$, and $\boldsymbol{\ell}$. This result, which we refer to as optimality theorem, holds under a number of restrictive assumptions; nevertheless, it is applicable in several interesting case studies, as illustrated in subsequent sections. The optimality theorem shows the usefulness of the intermediate evaluation model since it provides a form of "bootstrap," whereby from a given degree of optimality on a family of machines we infer a related degree of optimality on a much larger family. It is important to remark that the class of algorithms for which the optimality theorem holds includes algorithms that are *network aware*—that is, whose code can make explicit use of the architectural parameters of the model (p and σ for the evaluation model, and p, $\boldsymbol{g}$, and $\boldsymbol{\ell}$ for the execution machine model) for optimization purposes.

In a nutshell, the approach we follow hinges on the fact that both communication complexity and communication time (Equations (1) and (2)) are expressed in terms of quantities of the type $F^i_{\mathcal{A}}(I, p)$. If communication complexity is low, then these quantities must be low, and thus communication time must be low as well. Next, we discuss a number of obstacles to be faced when attempting to refine the outlined approach into a rigorous argument and how they can be handled.

A first obstacle arises whenever the performance functions are linear combinations of other auxiliary metrics. Unfortunately, worst-case optimality of these metrics does not imply optimality of their linear combinations (nor vice versa), as the worst case of different metrics could be realized by different input instances. In the cases of our interest, the "losses" incurred cannot be generally bounded by constant factors. To circumvent this obstacle, we restrict our attention to *static algorithms*, defined by the property that the following quantities are equal for all input instances of the same size n: (i) the number of supersteps, (ii) the sequence of labels of the various supersteps, and (iii) the set of source-destination pairs of the messages exchanged in any individual superstep. This restriction allows us to overload the notation, writing n instead of I in the argument of functions that become invariant for instances of the same size, namely $L^i_{\mathcal{A}}(n)$, $S^i_{\mathcal{A}}(n)$, $h^s_{\mathcal{A}}(n, p)$, and $F^i_{\mathcal{A}}(n, p)$. Likewise, the max operation becomes superfluous and can be omitted in Equations (1) and (2). Static algorithms naturally arise in directed acyclic graph (DAG) computations. In a *DAG algorithm*, for every instance size n, there exists (at most) one DAG where each node with indegree 0 represents an input value, whereas each node with indegree greater than 0 represents a value produced by a unit-time operation whose operands are the values of the node's predecessors (nodes with outdegree 0 are viewed as outputs). The computation requires the execution of all operations specified by the nodes, complying with the data dependencies imposed by the arcs.[3]

To prove the optimality theorem, we need a number of technical results and definitions. Recall that folding can be employed to transform an $M(p, \sigma)$-algorithm into an $M(2^j, \sigma)$-algorithm, for any $1 \leq j \leq \log p$: as already mentioned, an algorithm designed

[3]In the literature, DAG problems have also been referred to as pebble games [Savage 1998, Section 10.1].

on the $M(p, \sigma)$ can be regarded as algorithms for $M(p)$, once the parameter σ is fixed; then, we can analyze the communication complexity of its folding on a smaller $M(2^j, \sigma')$ machine, for any $0 \le j < \log p$ and $\sigma' \ge 0$. The following lemma establishes a useful relation between the communication metrics when folding is applied.

LEMMA 3.1. *Let $\mathcal{B}$ be a static $M(p, \sigma)$-algorithm. For every input size n, $1 \le j \le \log p$ and $\sigma' \ge 0$, considering the folding of $\mathcal{B}$ on $M(2^j, \sigma')$ we have*

$$\sum_{i=0}^{j-1} F^i_{\mathcal{B}}(n, 2^j) \le \frac{p}{2^j} \sum_{i=0}^{j-1} F^i_{\mathcal{B}}(n, p).$$

PROOF. The lemma follows by observing that in every i-superstep, with $i < j$, messages sent/destined by/to processor P_k of $M(2^j, \sigma')$, with $0 \le k < 2^j$, are a subset of those sent/destined by/to the $p/2^j$ $M(p, \sigma)$-processors whose computations are carried out by P_k. □

It is easy to come up with algorithms where the bound stated in the preceding lemma is not tight. In fact, whereas in an i-superstep each message must be exchanged between processors whose indices share *at least* i most significant bits, some messages that contribute to $F^i_{\mathcal{B}}(n, p)$ may be exchanged between processors whose indices share $j > i$ most significant bits, thus not contributing to $F^i_{\mathcal{B}}(n, 2^j)$. Motivated by this observation, next we define a class of network-oblivious algorithms where a parameter α quantifies how tight the upper bound of Lemma 3.1 is when considering their foldings on smaller machines. This parameter will be employed to control the extent to which an optimality guarantee in the evaluation model translates into an optimality guarantee in the execution model.

Definition 3.2. A static network-oblivious algorithm $\mathcal{A}$ specified on $M(v(n))$ is said to be (α, p)-*wise*, for some $0 < \alpha \le 1$ and $1 < p \le v(n)$, if considering the folding of $\mathcal{A}$ on $M(2^j, 0)$ we have

$$\sum_{i=0}^{j-1} F^i_{\mathcal{A}}(n, 2^j) \ge \alpha \frac{p}{2^j} \sum_{i=0}^{j-1} F^i_{\mathcal{A}}(n, p),$$

for every $1 \le j \le \log p$ and every input size n.

(We remark that in the preceding definition, parameter α is not necessarily a constant and can be made, for example, a function of p.) Intuitively, (α, p)-wiseness is meant to capture, in an average sense, the property that for each i-superstep involving an h-relation, there exists an i-cluster where an α-fraction of the processors send/receive h messages to/from processors belonging to a different $(i+1)$-subcluster. As an example, a network-oblivious algorithm for $M(v(n))$ where, for each i-superstep there is always at least one segment of $v(n)/2^{i+1}$ virtual processors consecutively numbered starting from $k \cdot (v(n)/2^{i+1})$, for some $k \ge 0$, each sending a number of messages equal to the superstep degree to processors outside the segment, is an (α, p)-wise algorithm for each $1 < p \le v(n)$ and $\alpha = 1$. However, (α, p)-wiseness holds even if the aforementioned communication scenario is realized only in an average sense. Furthermore, consider a pair of values α' and p' such that $1 < p' \le p$ and $1 < \alpha' \le \alpha$. It is easy to see that $(p/p')F^i_{\mathcal{A}}(n, p) \ge F^i_{\mathcal{A}}(n, p')$, for every $0 \le i < \log p'$, and this implies that a network-oblivious algorithm that is (α, p)-wise is also (α', p')-wise.

A final issue to consider is that the degrees of supersteps with different labels contribute with the same weight to the communication complexity while they contribute with different weights to the communication time. The following lemma will help in bridging this difference.

LEMMA 3.3. *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 real values, and let $\langle f_0, f_1, \ldots, f_{m-1}\rangle$ be a nonincreasing sequence of nonnegative real values. If $\sum_{i=0}^{k-1} X_i \leq \sum_{i=0}^{k-1} Y_i$, for every $1 \leq k \leq m$, then*

$$\sum_{i=0}^{m-1} X_i f_i \leq \sum_{i=0}^{m-1} Y_i f_i\,.$$

PROOF. By defining $S_0 = 0$ and $S_k = \sum_{j=0}^{k-1}(Y_j - X_j) \geq 0$, for $1 \leq k \leq m$, we have

$$\begin{aligned}\sum_{i=0}^{m-1} f_i(Y_i - X_i) &= \sum_{i=0}^{m-1} f_i(S_{i+1} - S_i) = \sum_{i=0}^{m-1} f_i S_{i+1} - \sum_{i=1}^{m-1} f_i S_i\\ &\geq \sum_{i=0}^{m-1} f_i S_{i+1} - \sum_{i=1}^{m-1} f_{i-1} S_i = f_{m-1} S_m \geq 0.\end{aligned}$$

We then get the desired inequality $\sum_{i=0}^{m-1} X_i f_i \leq \sum_{i=0}^{m-1} Y_i f_i$. □

We are now ready to state and prove the optimality theorem. Let $\mathscr{C}$ denote a class of static algorithms solving a problem Π, with the property that for any algorithm $\mathcal{A} \in \mathscr{C}$ for v processing elements, all of its foldings on 2^j processing elements, for each $1 \leq j < \log v$, also belong to $\mathscr{C}$.

THEOREM 3.4 (OPTIMALITY THEOREM). *Let $\mathcal{A} \in \mathscr{C}$ be network oblivious and $(\alpha, p^\star)$-wise, for some $\alpha \in (0, 1]$ and $p^\star$ a power of 2. Let also $(\sigma_0^m, \sigma_1^m, \ldots, \sigma_{\log p^\star-1}^m)$ and $(\sigma_0^M, \sigma_1^M, \ldots, \sigma_{\log p^\star-1}^M)$ be two vectors of nonnegative values, with $\sigma_j^m \leq \sigma_j^M$, for every $0 \leq j < \log p^\star$. If $\mathcal{A}$ is β-optimal on $M(2^j, \sigma)$ with respect to $\mathscr{C}$, for $\sigma_{j-1}^m \leq \sigma \leq \sigma_{j-1}^M$ and $1 \leq j \leq \log p^\star$, then for every p power of 2, $p \leq p^\star$, $\mathcal{A}$ is $\alpha\beta/(1+\alpha)$-optimal on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ with respect to $\mathscr{C}$ as long as*

—*$g_i \geq g_{i+1}$ and $\ell_i/g_i \geq \ell_{i+1}/g_{i+1}$, for $0 \leq i < \log p - 1$;*
—*$\max_{1\leq k\leq \log p}\{\sigma_{k-1}^m 2^k/p\} \leq \ell_i/g_i \leq \min_{1\leq k\leq \log p}\{\sigma_{k-1}^M 2^k/p\}$, for $0 \leq i < \log p$.*[4]

PROOF. Fix the value p and the vectors $\boldsymbol{g}$ and $\boldsymbol{\ell}$ so as to satisfy the hypotheses of the theorem, and consider a D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$-algorithm $\mathcal{C} \in \mathscr{C}$. By the β-optimality of $\mathcal{A}$ on the evaluation model $M(2^j, \psi p/2^j)$, for each $1 \leq j \leq \log p$ and ψ such that $\sigma_{j-1}^m \leq \psi p/2^j \leq \sigma_{j-1}^M$, we have

$$H_{\mathcal{A}}\left(n, 2^j, \frac{\psi p}{2^j}\right) \leq \frac{1}{\beta} H_{\mathcal{C}}\left(n, 2^j, \frac{\psi p}{2^j}\right)$$

since $\mathcal{C}$ can be folded into an algorithm for $M(2^j, \psi p/2^j)$, still belonging to $\mathscr{C}$. By the definition of communication complexity, it follows that

$$\sum_{i=0}^{j-1}\left(F_{\mathcal{A}}^i(n, 2^j) + S_{\mathcal{A}}^i(n)\frac{\psi p}{2^j}\right) \leq \frac{1}{\beta}\sum_{i=0}^{j-1}\left(F_{\mathcal{C}}^i(n, 2^j) + S_{\mathcal{C}}^i(n)\frac{\psi p}{2^j}\right),$$

[4]Note that to allow for a nonempty range of values for the ratio ℓ_i/g_i, the σ^m and σ^M vectors must be such that $\max_{1\leq k\leq \log p}\{\sigma_{k-1}^m 2^k/p\} \leq \min_{1\leq k\leq \log p}\{\sigma_{k-1}^M 2^k/p\}$. This will always be the case for the applications discussed in the next section.

and then, by applying Lemma 3.1 to the right side of the preceding inequality, we obtain

$$\sum_{i=0}^{j-1}\left(F^i_{\mathcal{A}}(n,2^j)+S^i_{\mathcal{A}}(n)\frac{\psi p}{2^j}\right)\le\frac{1}{\beta}\sum_{i=0}^{j-1}\left(\frac{p}{2^j}F^i_{\mathcal{C}}(n,p)+S^i_{\mathcal{C}}(n)\frac{\psi p}{2^j}\right). \tag{3}$$

Define $\psi^m_p=\max_{1\le k\le\log p}\{\sigma^m_{k-1}2^k/p\}$ and $\psi^M_p=\min_{1\le k\le\log p}\{\sigma^M_{k-1}2^k/p\}$. The condition imposed by the theorem on the ratio ℓ_i/g_i implies that $\psi^m_p\le\psi^M_p$, and hence, by definition of these two quantities, we have that $\sigma^m_{j-1}2^j/p\le\psi^m_p$, $\psi^M_p\le\sigma^M_{j-1}2^j/p$.

Let us first set $\psi=\psi^M_p$ in Inequality (3), and note that by the preceding observation, $\sigma^m_{j-1}\le\psi^M_p p/2^j\le\sigma^M_{j-1}$. By multiplying both terms of the inequality by $2^j/(\psi^M_p p)$, and by exploiting the nonnegativeness of the $F^i_{\mathcal{A}}(n,2^j)$ terms, we obtain

$$\sum_{i=0}^{j-1}S^i_{\mathcal{A}}(n)\le\frac{1}{\beta}\sum_{i=0}^{j-1}\left(\frac{F^i_{\mathcal{C}}(n,p)}{\psi^M_p}+S^i_{\mathcal{C}}(n)\right).$$

Next, we make $\log p$ applications of Lemma 3.3, one for each $j=1,2,\dots,\log p$, by setting $m=j$, $X_i=S^i_{\mathcal{A}}(n)$, $Y_i=(1/\beta)(F^i_{\mathcal{C}}(n,p)/\psi^M_p+S^i_{\mathcal{C}}(n))$, and $f_i=\ell_i/g_i$. This gives

$$\sum_{i=0}^{j-1}S^i_{\mathcal{A}}(n)\frac{\ell_i}{g_i}\le\frac{1}{\beta}\sum_{i=0}^{j-1}\left(F^i_{\mathcal{C}}(n,p)\frac{\ell_i}{\psi^M_p g_i}+S^i_{\mathcal{C}}(n)\frac{\ell_i}{g_i}\right),$$

for $1\le j\le\log p$. Since by hypothesis $\ell_i/g_i\le\psi^M_p$, for each $0\le i<\log p$, we have $\ell_i/\psi^M_p g_i\le 1$, and hence we can write

$$\sum_{i=0}^{j-1}S^i_{\mathcal{A}}(n)\frac{\ell_i}{g_i}\le\frac{1}{\beta}\sum_{i=0}^{j-1}\left(F^i_{\mathcal{C}}(n,p)+S^i_{\mathcal{C}}(n)\frac{\ell_i}{g_i}\right), \tag{4}$$

for $1\le j\le\log p$.

Now, let us set $\psi=\psi^m_p$ in Inequality (3), which again guarantees $\sigma^m_{j-1}\le\psi^m_p p/2^j\le\sigma^M_{j-1}$. By exploiting the wiseness of $\mathcal{A}$ in the left side and the nonnegativeness of $S^i_{\mathcal{A}}(n)$, we obtain

$$\sum_{i=0}^{j-1}\alpha\frac{p}{2^j}F^i_{\mathcal{A}}(n,p)\le\frac{1}{\beta}\sum_{i=0}^{j-1}\left(\frac{p}{2^j}F^i_{\mathcal{C}}(n,p)+S^i_{\mathcal{C}}(n)\frac{\psi^m_p p}{2^j}\right).$$

By multiplying both terms by $2^j/(p\alpha)$ and observing that by hypothesis $\psi^m_p\le\ell_i/g_i$, for each $0\le i<\log p$, we get

$$\sum_{i=0}^{j-1}F^i_{\mathcal{A}}(n,p)\le\frac{1}{\alpha\beta}\sum_{i=0}^{j-1}\left(F^i_{\mathcal{C}}(n,p)+S^i_{\mathcal{C}}(n)\frac{\ell_i}{g_i}\right). \tag{5}$$

Summing Inequality (4) with Inequality (5) yields

$$\sum_{i=0}^{j-1}\left(F^i_{\mathcal{A}}(n,p)+S^i_{\mathcal{A}}(n)\frac{\ell_i}{g_i}\right)\le\frac{1+\alpha}{\alpha\beta}\sum_{i=0}^{j-1}\left(F^i_{\mathcal{C}}(n,p)+S^i_{\mathcal{C}}(n)\frac{\ell_i}{g_i}\right),$$

for $1 \leq j \leq \log p$. Applying Lemma 3.3 with $m = \log p$, $X_i = F^i_{\mathcal{A}}(n, p) + S^i_{\mathcal{A}}(n)\ell_i/g_i$, $Y_i = (1+\alpha)/(\alpha\beta)(F^i_{\mathcal{C}}(n, p) + S^i_{\mathcal{C}}(n)\ell_i/g_i)$, and $f_i = g_i$ yields

$$\sum_{i=0}^{\log p-1} \left(F^i_{\mathcal{A}}(n, p)g_i + S^i_{\mathcal{A}}(n)\ell_i\right) \leq \frac{1+\alpha}{\alpha\beta} \sum_{i=0}^{\log p-1} \left(F^i_{\mathcal{C}}(n, p)g_i + S^i_{\mathcal{C}}(n)\ell_i\right). \tag{6}$$

Then, by definition of communication time, we have

$$D_{\mathcal{A}}(n, p, \boldsymbol{g}, \boldsymbol{\ell}) \leq \frac{1+\alpha}{\alpha\beta} D_{\mathcal{C}}(n, p, \boldsymbol{g}, \boldsymbol{\ell}),$$

and the theorem follows. □

Note that the theorem requires that both the g_i's and ℓ_i/g_i's form nonincreasing sequences. The assumption is rather natural, as it reflects the fact that larger submachines exhibit more expensive communication (hence, a larger g parameter) and larger network capacity (hence, a larger ℓ/g ratio).

A few remarks regarding the preceding optimality theorem are in order. First, the proof of the theorem heavily relies on the manipulation of linear combinations of worst-case metrics related to executions of the algorithms with varying degrees of parallelism. This justifies the restriction to static algorithms, since, as anticipated at the beginning of the section, the variation of the metrics with the input instances would make the derivations invalid. However, based on the fact that the linear combinations involve a logarithmic number of terms, the proof of the theorem can be extended to nonstatic algorithms by increasing the gap between optimality in the evaluation model and optimality in the execution machine model by an extra $O(\log p)$ factor. Specifically, for arbitrary algorithms, after a straightforward reinterpretation of the quantities in a worst-case sense, the summation on the right-hand side of Equation (6), although not necessarily equal to $D_{\mathcal{C}}(n, p, \boldsymbol{g}, \boldsymbol{\ell})$, can be shown to be a factor at most $O(\log p)$ larger.

The complexity metrics adopted in this article target exclusively interprocessor communication, and thus a (sequential) network-oblivious algorithm specified on $M(v)$ but using only one of the virtual processors would clearly be optimal with respect to these metrics. For meaningful applications of the theorem, the class $\mathscr{C}$ must be suitably defined to exclude such degenerate cases and to contain algorithms where the work is sufficiently well balanced among the processing elements. In addition, one could argue that the effectiveness of our framework is confined only to very regular algorithms, because of the wiseness hypothesis and the fact that the evaluation model uses the maximum number of messages sent/received by a processor as the key descriptor for communication costs, thus disregarding the overall communication volume. However, it has to be remarked that wiseness can be achieved even under communication patterns that are globally unbalanced, as long as some balancing is locally guaranteed within some cluster. Additionally, since the quest for optimality requires evaluating an algorithm at different levels of granularity, communication patterns with the same maximum message count at a processor but different overall communication volume may be discriminated, to some extent, by their different communication costs at coarser granularities.

Some of the issues encountered in establishing the optimality theorem have an analog in the context of memory hierarchies. For example, time in the hierarchical memory model (HMM) can be linked to I/O complexity as discussed in Aggarwal et al. [1987] so that optimality of the latter for different cache sizes implies the optimality of the former for wide classes of functions describing the access time to different memory locations. Although, to the best of our knowledge, the question has not been explicitly addressed in the literature, a careful inspection of the arguments of Aggarwal et al. [1987] shows that some restriction to the class of algorithms is required to guarantee that the maximum value of the I/O complexity for different cache sizes is simultaneously reached for the same input instance. (For example, the optimality of HMM time does not follow for the class of arbitrary comparison-based sorting algorithms, as the known I/O complexity lower bound for this problem [Aggarwal and Vitter 1988] may not be simultaneously reachable for all relevant cache sizes.) Moreover, the monotonicity that we have assumed for the g_i and the ℓ_i/g_i sequences has an analog in the assumption that the function used in Aggarwal et al. [1987] to model the memory access time is polynomially bounded.

In the cache-oblivious framework, the equivalent of our optimality theorem requires algorithms to satisfy the *regularity condition* [Frigo et al. 2012, Lemma 6.4], which requires that the number of cache misses decreases by a constant factor when the cache size is doubled. On the other hand, our optimality theorem gives the best bound when the network-oblivious algorithm is $(\Theta(1), p)$-wise—that is, when the communication complexity decreases by a constant factor when the number of processors is doubled. Although the regularity condition and wiseness cannot be formalized in a similar fashion due to the significant differences between the cache- and network-oblivious frameworks, we observe that both assumptions require the oblivious algorithms to react seamlessly and smoothly to small changes of the machine parameters.

4. ALGORITHMS FOR FUNDAMENTAL PROBLEMS

In this section, we illustrate the use of the proposed framework by developing efficient network-oblivious algorithms for a number of fundamental computational problems: matrix multiplication (Section 4.1), FFT (Section 4.2), and sorting (Section 4.3). All of our algorithms exhibit $\Theta(1)$-optimality on the D-BSP for wide ranges of the machine parameters. In Section 4.4, we also present network-oblivious algorithms for stencil computations. These latter algorithms run efficiently on the D-BSP, although they do not achieve $\Theta(1)$-optimality, which appears to be a hard challenge in this case. In Section 4.5, we also establish a negative result by proving that there cannot exist a network-oblivious algorithm for broadcasting that is simultaneously $\Theta(1)$-optimal on two sufficiently different $M(p, \sigma)$ machines.

As prescribed by our framework, the performance of the network-oblivious algorithms on the D-BSP is derived by analyzing their performance on the evaluation model. Optimality is assessed with respect to classes of algorithms where the computation is not excessively unbalanced among the processors, namely algorithms where an individual processor cannot perform more than a constant fraction of the total minimum work for the problem. For this purpose, we exploit some recent lower bounds that rely on mild assumptions on work distributions and strengthen previous bounds based on stronger assumptions [Scquizzato and Silvestri 2014]. Finally, we want to stress that all of our algorithms are also work optimal.

4.1. Matrix Multiplication

The *n-MM problem* consists of multiplying two $\sqrt{n} \times \sqrt{n}$ matrices, A and B, using only semiring operations. A result in Kerr [1970] shows that any static algorithm for the n-MM problem that uses only semiring operations must compute all $n^{3/2}$ *multiplicative terms*—that is, the products $A[i, k] \cdot B[k, j]$, with $0 \le i, j, k < \sqrt{n}$.

Let $\mathscr{C}$ denote the class of static algorithms for the n-MM problem such that any $\mathcal{A} \in \mathscr{C}$ for v processing elements satisfies the following properties: (i) no entry of A or B is initially replicated (however, the entries of A and B are allowed to be initially distributed among the processing elements in an arbitrary fashion); (ii) no processing element computes more than $n^{3/2}/\min\{v, 11^3\}$ multiplicative terms;[5] and (iii) all of the foldings of $\mathcal{A}$ on 2^j processing elements, for each $1 \le j < \log v$, also belong to $\mathscr{C}$. The following lemma establishes a lower bound on the communication complexity of the algorithms in $\mathscr{C}$.

LEMMA 4.1. *The communication complexity of any n-MM algorithm in $\mathscr{C}$ when executed on $M(p, \sigma)$ is $\Omega(n/p^{2/3} + \sigma)$.*

PROOF. The bound for $\sigma = 0$ is proved in Theorem 2 of Scquizzato and Silvestri [2014], and it clearly extends to the case $\sigma > 0$. The additive σ term follows since at least one message is sent by some processing element. □

We now describe a static network-oblivious algorithm for the n-MM problem, which follows from the parallelization of the respective cache-oblivious algorithm [Frigo et al. 2012]. Then, we prove its optimality in the evaluation model, for wide ranges of the parameters, and in the execution model through the optimality theorem. For convenience, we assume that n is a power of 2^3 (the general case requires minor yet tedious modifications). The algorithm is specified on $M(n)$ and requires that the input and output matrices be evenly distributed among the n VPs. We denote with A, B, and C the two input matrices and the output matrix, respectively, and with A_{hk}, B_{hk}, and C_{hk}, with $0 \le h, k \le 1$, their four quadrants. The network-oblivious algorithm adopts the following recursive strategy:

(1) Partition the VPs into eight segments $S_{hk\ell}$, with $0 \le h, k, \ell \le 1$, containing the same number of consecutively numbered VPs. Replicate and distribute the inputs so that the entries of $A_{h\ell}$ and $B_{\ell k}$ are evenly spread among the VPs in $S_{hk\ell}$.
(2) In parallel, for each $0 \le h, k, \ell \le 1$, recursively compute the product $M_{hk\ell} = A_{h\ell} \cdot B_{\ell k}$ within $S_{hk\ell}$.
(3) In parallel, for each $0 \le i, j < \sqrt{n}$, the VP responsible for $C[i, j]$ collects $M_{hk0}[i', j']$ and $M_{hk1}[i', j']$, with $h = \lfloor 2i/\sqrt{n} \rfloor$, $k = \lfloor 2j/\sqrt{n} \rfloor$, $i' = i \mod (\sqrt{n}/2)$ and $j' = j \mod (\sqrt{n}/2)$, and computes $C[i, j] = M_{hk0}[i', j'] + M_{hk1}[i', j']$.

At the i-th recursion level, with $0 \le i \le (\log n)/3$, 8^i $(n/4^i)$-MM subproblems are solved by distinct $M(n/8^i)$'s formed by distinct segments of VPs. The recursion stops at $i = (\log n)/3$ when each VP sequentially solves an $n^{1/3}$-MM subproblem. By unfolding the recursion, we get that the algorithm comprises a constant number of $3i$-supersteps at the i-th recursive level, where each VP sends/receives $O(2^i)$ messages. To easily claim that the algorithm is $(\Theta(1), n)$-wise, we may assume that in each $3i$-superstep, VP_j sends 2^i dummy messages to $\mathrm{VP}_{j+n/2^{3i+1}}$, for $0 \le j < n/2^{3i+1}$. These messages do not affect the asymptotic communication complexity and communication time exhibited by the algorithm in the evaluation and execution machine models. (In fact, constant wiseness is already achieved by the original communication pattern, but a direct proof would have required a more convoluted argument than resorting to dummy messages. Indeed, we will use the same trick in the other network-oblivious algorithms presented in the article.)

[5]The min term follows from the lower bound in Theorem 2 of Scquizzato and Silvestri [2014], which applies to computations where each processor computes at most $n^{3/2}/\min\{v, 11^3\}$ multiplicative terms on a BSP with v processors. Clearly, weakening the assumption for Theorem 2 of Scquizzato and Silvestri [2014] automatically translates into a weaker property (ii).

THEOREM 4.2. *The communication complexity of the preceding n-MM network-oblivious algorithm when executed on $M(p, \sigma)$ is*

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

for every $1 < p \leq n$ and $\sigma \geq 0$. The algorithm is $(\Theta(1), n)$-wise and $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any $M(p, \sigma)$ with $1 < p \leq n$ and $\sigma = O(n/(p^{2/3} \log p))$.

PROOF. When executed on $M(p, \sigma)$, the preceding algorithm decomposes the problem into eight subproblems that are solved by eight distinct $M(p/8, \sigma)$ machines and each processor sends/receives $O(n/p)$ messages in $O(1)$ supersteps for processing the inputs and outputs of the eight subproblems. The communication complexity satisfies the recurrence relation:

$$H_{\mathrm{MM}}(n, p, \sigma) = \begin{cases} H_{\mathrm{MM}}(n/4, p/8, \sigma) + O(n/p + \sigma) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

By unrolling the recurrence, we get

$$H_{\mathrm{MM}}(n, p, \sigma) = O\left(\sum_{i=0}^{(\log p)/3} \left(\frac{n2^i}{p} + \sigma\right)\right) = O\left(\frac{n}{p^{2/3}} + \sigma \log p\right).$$

As anticipated, the wiseness is guaranteed by the dummy messages introduced in each superstep. Finally, it is easy to see that the algorithm satisfies the three requirements for belonging to $\mathscr{C}$, and hence its optimality follows from Lemma 4.1. □

COROLLARY 4.3. *The preceding n-MM network-oblivious algorithm is $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ machine with $1 < p \leq n$, nonincreasing g_i's and ℓ_i/g_i's, and $\ell_0/g_0 = O(n/p)$.*

PROOF. Since the network-oblivious algorithm is $(\Theta(1), n)$-wise and belongs to $\mathscr{C}$, the corollary follows by plugging $p^\star = n$, $\sigma_i^m = 0$, and $\sigma_i^M = \Theta(n/((i+1)2^{2i/3}))$ into Theorem 3.4. □

4.1.1. Space-Efficient Matrix Multiplication. Observe that the network-oblivious algorithm described earlier incurs an $O(n^{1/3})$ memory blow-up per VP. As described next, the recursive strategy can be modified to incur only a constant memory blow-up, at the expense of an increased communication complexity. The resulting network-oblivious algorithm turns out to be $\Theta(1)$-optimal with respect to the class of algorithms featuring constant memory blow-up.

We assume, as before, that the entries of A, B, and C be evenly distributed among the VPs. The VPs are (recursively) divided into four segments that solve the eight $(n/4)$-MM subproblems in two rounds: in the first round, the segments compute $A_{00} \cdot B_{00}$, $A_{01} \cdot B_{11}$, $A_{11} \cdot B_{10}$, and $A_{10} \cdot B_{01}$ (one product per segment), whereas in the second round, they compute $A_{01} \cdot B_{10}$, $A_{00} \cdot B_{01}$, $A_{10} \cdot B_{00}$, and $A_{11} \cdot B_{11}$ (again, one product per segment). The recursion ends when each VP sequentially solves a 1-MM subproblem. By unfolding the recursion, we get that for every $0 \leq i < \log n/2$, the algorithm executes $\Theta(2^i)$ $2i$-supersteps where each VP sends/receives $\Theta(1)$ messages. At any time, each VP contains only $O(1)$ matrix entries, but the recursion requires it to handle a stack of $O(\log n)$ entries. However, it is easy to see that only a constant number of bits are needed for each stack entry, and hence, under the natural assumption that each matrix entry occupies a constant number of $\Theta(\log n)$-bit words, the entire stack at each VP requires storage proportional to $O(1)$ matrix entries. Therefore, the algorithm incurs only a constant memory blow-up. As before, the algorithm can be easily made $(\Theta(1), n)$-wise by adding suitable dummy messages.

When executed on $M(p, \sigma)$, the preceding space-efficient algorithm exhibits a communication complexity, denoted with $H_{\text{MM-space}}(n, p, \sigma)$, that satisfies the recurrence relation:

$$H_{\text{MM-space}}(n, p, \sigma) = \begin{cases} 2H_{\text{MM-space}}(n/4, p/4, \sigma) + O(n/p + \sigma) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

By unrolling the relation, we get $H_{\text{MM-space}}(n, p, \sigma) = O(n/\sqrt{p} + \sigma\sqrt{p})$.

Let $\mathcal{C}'$ denote the class of static algorithms for the n-MM problem such that any $\mathcal{A} \in \mathcal{C}'$ for v processing elements satisfies the following properties: (i) the local storage required at each processing element is $O(n/v)$, and (ii) all of the foldings of $\mathcal{A}$ on 2^j processing elements, for each $1 \leq j < \log v$, also belong to $\mathcal{C}'$. Since it is proved in Irony et al. [2004] that any n-MM algorithm in $\mathcal{C}'$ when running on $M(p, 0)$ must exhibit an $\Omega(n/\sqrt{p})$ communication complexity, the preceding network-oblivious algorithm is $\Theta(1)$-optimal with respect to $\mathcal{C}'$ on any $M(p, \sigma)$ with $1 < p \leq n$ and $\sigma = O(n/p)$. Consequently, Theorem 3.4 yields optimality of the algorithm on any D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ machine with $1 < p \leq n$, nonincreasing g_i's and ℓ_i/g_i's, and $\ell_0/g_0 = O(n/p)$.

4.2. Fast Fourier Transform

The *n-FFT problem* consists of computing the discrete Fourier transform of n values using the n-input FFT DAG, where a vertex is a pair $\langle w, l\rangle$, with $0 \leq w < n$ and $0 \leq l < \log n$, and there exists an arc between two vertices $\langle w, l\rangle$ and $\langle w', l'\rangle$ if $l' = l + 1$, and either w and w' are identical or their binary representations differ exactly in the l-th bit [Leighton 1992].

Let $\mathcal{C}$ denote the class of static algorithms for the n-FFT problem such that any $\mathcal{A} \in \mathcal{C}$ for v processing elements satisfies the following properties: (i) each DAG node is evaluated exactly once (i.e., recomputation is not allowed); (ii) no input value is initially replicated; (iii) no processing element computes more than $\epsilon n \log n$ DAG nodes, for some constant $0 < \epsilon < 1$; and (iv) all of the foldings of $\mathcal{A}$ on 2^j processing elements, for each $1 \leq j < \log v$, also belong to $\mathcal{C}$. Note that, as in the preceding section, the class of algorithms that we are considering makes no assumptions on the input and output distributions. The following lemma establishes a lower bound on the communication complexity of the algorithms in $\mathcal{C}$.

LEMMA 4.4. *The communication complexity of any n-FFT algorithm in $\mathcal{C}$ when executed on $M(p, \sigma)$ is $\Omega((n \log n)/(p \log(n/p)) + \sigma)$.*

PROOF. The bound for $\sigma = 0$ is proved in Theorem 11 of Scquizzato and Silvestri [2014], and it clearly extends to the case $\sigma > 0$. The additive σ term follows since at least one message is sent by some processing element. □

We now describe a static network-oblivious algorithm for the n-FFT problem and then prove its optimality in the evaluation and execution models. The algorithm is specified on $M(n)$ and exploits the well-known decomposition of the FFT DAG into two sets of $\sqrt{n}$-input FFT subDAGs, with each set containing $\sqrt{n}$ such subDAGs [Aggarwal et al. 1987]. For simplicity, to ensure integrality of the quantities involved, we assume $n = 2^{2^k}$ for some integer $k \geq 0$. We assume that at the beginning, the n inputs are evenly distributed among the n VPs. In parallel, each of the $\sqrt{n}$ segments of $\sqrt{n}$ consecutively numbered VPs recursively computes the assigned subDAG. Then, the outputs of the first set of subDAGs are permuted in a 0-superstep so as to distribute the inputs of each subDAGs of the second set among the VPs of a distinct segment. The permutation pattern is equivalent to the transposition of a $\sqrt{n} \times \sqrt{n}$ matrix. Finally, each segment recursively computes the assigned subDAG.

At the i-th recursion level, with $0 \leq i < \log\log n$, $n^{1-1/2^i}$ $n^{1/2^i}$-FFT subproblems are solved by $n^{1-1/2^i}$ $M(n^{1/2^i})$ models formed by distinct segments of VPs. The recurrence stops at $i = \log\log n$ when each segment of two VPs computes a 2-input subDAG. It is easy to see, by unfolding the recursion, that the algorithm comprises $O(2^i)$ supersteps with label $(1 - 1/2^i)\log n$ at the i-th recursive level, where each VP sends/receives $O(1)$ messages. As before, to enforce wiseness without affecting the algorithm's asymptotic performance, we assume that in each $(1 - 1/2^i)\log n$-superstep, VP_j sends a dummy message to $\mathrm{VP}_{j+n^{1/2^i}/2}$, for each $0 \leq j < n^{1/2^i}/2$.

THEOREM 4.5. *The communication complexity of the preceding n-FFT network-oblivious algorithm when executed on $M(p, \sigma)$ is*

$$H_{\mathrm{FFT}}(n, p, \sigma) = O\left(\left(\frac{n}{p} + \sigma\right)\frac{\log n}{\log(n/p)}\right),$$

for every $1 < p \leq n$ and $\sigma \geq 0$. The algorithm is $(\Theta(1), n)$-wise and $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any $M(p, \sigma)$ with $1 < p \leq n$ and $\sigma = O(n/p)$.

PROOF. When executed on $M(p, \sigma)$, the preceding algorithm decomposes the problem into two sets of $\sqrt{n}$ subproblems that are solved by $\sqrt{n}$ distinct $M(p/\sqrt{n}, \sigma)$ machines and each processor sends/receives $O(n/p)$ messages in $O(1)$ supersteps for processing the inputs and outputs of the $2\sqrt{n}$ subproblems. The communication complexity satisfies the recurrence relation:

$$H_{\mathrm{FFT}}(n, p, \sigma) = \begin{cases} 2H_{\mathrm{FFT}}(\sqrt{n}, p/\sqrt{n}, \sigma) + O(n/p + \sigma) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

By unrolling the recurrence, we get

$$H_{\mathrm{FFT}}(n, p, \sigma) = O\left(\sum_{i=0}^{\log(\log n/\log(n/p))} 2^i \left(\frac{n}{p} + \sigma\right)\right) = O\left(\left(\frac{n}{p} + \sigma\right)\frac{\log n}{\log(n/p)}\right).$$

The wiseness is ensured by the dummy messages, and since the algorithm satisfies the requirements for belonging to $\mathscr{C}$, its optimality follows from Lemma 4.4. □

We now apply Theorem 3.4 to show that the network-oblivious algorithm is $\Theta(1)$-optimal on the D-BSP for wide ranges of the machine parameters.

COROLLARY 4.6. *The preceding n-FFT network-oblivious algorithm is $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any D-BSP$(p, \mathbf{g}, \boldsymbol{\ell})$ machine with $1 < p \leq n$, nonincreasing g_i's and ℓ_i/g_i's, and $\ell_0/g_0 = O(n/p)$.*

PROOF. Since the network-oblivious algorithm is $(\Theta(1), n)$-wise and belongs to $\mathscr{C}$, we get the claim by plugging $p^\star = n$, $\sigma_i^m = 0$, and $\sigma_i^M = \Theta(n/2^i)$ in Theorem 3.4. □

We observe that although we described the network-oblivious algorithm assuming $n = 2^{2^k}$, to ensure integrality of the quantities involved, the preceding results can be generalized to the case of n arbitrary power of 2. In this case, the FFT DAG is recursively decomposed into a set of $2^{\lfloor \log \sqrt{n} \rfloor}$-input FFT subDAGs and a set of $n/2^{\lfloor \log \sqrt{n} \rfloor}$-input FFT subDAGs. The optimality of the resulting algorithm in both the evaluation and execution machine models can be proved in a similar fashion as before.

4.3. Sorting

The *n-sort problem* requires labeling n (distinct) input keys with their ranks, using only comparisons, where the *rank* of a key is the number of smaller keys in the input sequence.

Let $\mathscr{C}$ denote the class of static algorithms for the n-sort problem such that any $\mathcal{A} \in \mathscr{C}$ for v processing elements satisfies the following properties: (i) initially, no input key is replicated and, during the course of the algorithm, only a constant number of copies per key are allowed at any time; (ii) no processing element performs more than $\epsilon n \log n$ comparisons, for an arbitrary constant $0 < \epsilon < 1$; and (iii) all of the foldings of $\mathcal{A}$ on 2^j processing elements, $1 \le j < \log v$, also belong to $\mathscr{C}$. We make no assumptions on how the keys are distributed among the processing elements at the beginning and at the end of the algorithm. The following lemma establishes a lower bound on the communication complexity of the algorithms in $\mathscr{C}$.

Lemma 4.7. *The communication complexity of any n-sort algorithm in $\mathscr{C}$ when executed on $M(p, \sigma)$ is $\Omega((n \log n)/(p \log(n/p)) + \sigma)$.* □

Proof. The bound for $\sigma = 0$ is proved in Theorem 8 of Scquizzato and Silvestri [2014], and it clearly extends to the case $\sigma > 0$. The additive σ term follows since at least one message is sent by some processing element. □

We now present a static network-oblivious algorithm for the n-sort problem and then prove its optimality in the evaluation and execution models. The algorithm implements a recursive version of the *Columnsort* strategy, as described in Leighton [1985]. Consider the n input keys as an $r \times s$ matrix, with $r \cdot s = n$ and $r \ge s^2$. Columnsort is organized into eight *phases* numbered from 1 to 8. During Phases 1, 3, 5, and 7, the keys in each column are sorted recursively (in Phase 5, adjacent columns are sorted in reverse order). During Phases 2, 4, 6, and 8, the keys of the matrix are permuted: in Phase 2 (respectively, Phase 4), a transposition (respectively, diagonalizing permutation [Leighton 1985]) of the $r \times s$ matrix is performed maintaining the $r \times s$ shape; in Phase 6 (respectively, Phase 8), an $r/2$-cyclic shift (respectively, the reverse of the $r/2$-cyclic shift) is done.[6] Columnsort can be implemented on $M(n)$ as follows. For convenience, assume that $n = 2^{(3/2)^d}$ for some integer $d \ge 0$, and set $r = n^{2/3}$ and $s = n/r$ (the more general case is discussed later). The algorithm starts with the input keys evenly distributed among the n VPs. In the odd phases, the keys of each column are evenly distributed among the VPs of a distinct segment of r consecutively numbered VPs, which form an independent $M(r)$. Then, each segment recursively solves the subproblem corresponding to the column it received. The even phases entail a constant number of 0-supersteps of constant degree. At the i-th recursion level, with $0 \le i \le \log_{3/2} \log n$, each segment of $n^{(2/3)^i}$ consecutively numbered VPs forming an independent $M(n^{(2/3)^i})$ solves 4^i subproblems of size $n^{(2/3)^i}$. The recurrence stops at $i = \log_{3/2} \log n$ when each VP solves, sequentially, a subproblem of constant size. It is easy to see, by unfolding the recursion, that the algorithm consists of $\Theta(4^i)$ supersteps with label $(1 - (2/3)^i) \log n$ at the i-th recursive level, where each VP sends/receives $O(1)$ messages. As before, to enforce wiseness without affecting the algorithm's asymptotic performance, we assume that in each $(1 - (2/3)^i) \log n$-superstep, VP_j sends a dummy message to $\text{VP}_{j+n^{(2/3)^i}/2}$, for each $0 \le j < n^{(2/3)^i}/2$.

[6]In the original paper [Leighton 1985], the shift in Phase 6 is not cyclic: a new column is added containing the $r/2$ overflowing keys and $r/2$ large dummy keys, whereas the first column is filled with $r/2$ small dummy keys. However, it is easy to see that a cyclic shift suffices if the first $r/2$ keys in the first column are considered smaller than the last $r/2$ keys.

THEOREM 4.8. *The communication complexity of the preceding network-oblivious algorithm for n-sort when executed on $M(p,\sigma)$ is*

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

for every $1 < p \le n$ and $\sigma \ge 0$. The algorithm is $(\Theta(1), n)$-wise and is $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any $M(p,\sigma)$ with $p = O(n^{1-\delta})$, for any arbitrary constant $\delta \in (0,1)$, and $\sigma \ge 0$.

PROOF. When executed on $M(p,\sigma)$, the preceding algorithm decomposes the problem into four sets of $n^{1/3}$ subproblems that are solved in four phases by $n^{1/3}$ distinct $M(p/n^{1/3},\sigma)$ machines and each processor sends/receives $O(n/p)$ messages in $O(1)$ supersteps for processing the inputs and outputs of the $4n^{1/3}$ subproblems. The communication complexity satisfies the recurrence relation:

$$H_{\text{sort}}(n, p, \sigma) = \begin{cases} 4H_{\text{sort}}(n^{2/3}, p/n^{1/3}, \sigma) + O(n/p + \sigma) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

By unrolling the recurrence, we get

$$H_{\text{sort}}(n, p, \sigma) = O\left(\sum_{i=0}^{\log_{3/2}(\log n/\log(n/p))} 4^i \left(\frac{n}{p} + \sigma\right)\right) = O\left(\left(\frac{n}{p} + \sigma\right)\left(\frac{\log n}{\log(n/p)}\right)^{\log_{3/2} 4}\right).$$

The wiseness is guaranteed by the dummy messages. Since the algorithm satisfies the three requirements to be in $\mathscr{C}$, its optimality follows from Lemma 4.7. □

COROLLARY 4.9. *The above n-sort network-oblivious algorithm is $\Theta(1)$-optimal with respect to $\mathscr{C}$ on any D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ machine with $p = O(n^{1-\delta})$, for some arbitrary constant $\delta \in (0,1)$, and non-increasing g_i's and ℓ_i/g_i's.*

PROOF. Since the network-oblivious algorithm is $(\Theta(1), n)$-wise and belongs to $\mathscr{C}$, we get the claim by plugging $p^\star = n$, $\sigma_i^m = 0$, and $\sigma_i^M = +\infty$ in Theorem 3.4. □

Consider now the more general case when n is an arbitrary power of 2. Now, the input keys must be regarded as the entries of an $r \times s$ matrix, where r is the smallest power of 2 greater than or equal to $n^{2/3}$. Simple yet tedious calculations show that the results stated in Theorem 4.8 and Corollary 4.9 continue to hold in this case.

Finally, we remark that the preceding network-oblivious sorting algorithm turns out to be $\Theta(1)$-optimal on any D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$, as long as $p = O(n^{1-\delta})$ for constant δ, with respect to a wider class of algorithms that satisfy requirements (i), (ii), and (iii), specified earlier for $\mathscr{C}$, but need not be static. By applying the lower bound for sorting in Scquizzato and Silvestri [2014] on two processors, it is easy to show that $\Omega(n)$ messages must cross the bisection for this class of algorithms. Therefore, we get an $\Omega(g_0 n/p)$ lower bound on the communication time on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$, which is matched by our network-oblivious algorithm.

4.4. Stencil Computations

A *stencil* defines the computation of any element in a d-dimensional spatial grid at time t as a function of neighboring grid elements at time $t-1, t-2, \ldots, t-\rho$, for some integers $\rho \ge 1$ and $d \ge 1$. Stencil computations arise in many contexts, ranging from iterative finite-difference methods for the numerical solution of partial differential equations to algorithms for the simulation of cellular automata, as well as in dynamic programming algorithms and in image-processing applications. In addition, the simulation of a d-dimensional mesh [Bilardi and Preparata 1997] can be envisioned as a stencil computation.

In this section, we restrict our attention to stencil computations with $\rho = 1$. To this purpose, we define the (n, d)-*stencil* problem, which represents a wide class of stencil computations (e.g., see Frigo and Strumpen [2005]). Specifically, the problem consists of evaluating all nodes of a DAG of n^{d+1} nodes, each represented by a distinct tuple $\langle i_0, i_1, \ldots, i_d \rangle$, with $0 \leq i_0, i_1, \ldots, i_d < n$, where each node $\langle i_0, i_1, \ldots, i_d \rangle$ is connected, through an outgoing arc, to (at most) 3^d neighbors, namely $\langle i_0 + \delta_0, i_1 + \delta_1, \ldots, i_{d-1} + \delta_{d-1}, i_d + 1 \rangle$ for each $\delta_0, \delta_1, \ldots, \delta_{d-1} \in \{0, \pm 1\}$ (whenever such nodes exist). We suppose n to be a power of 2. Intuitively, the (n, d)-stencil problem consists of n timesteps of a stencil computation on a d-dimensional spatial grid of side n, where each DAG node corresponds to a grid element (first d coordinates) at a given timestep (coordinate i_d).

Let $\mathscr{C}_d$ denote the class of static algorithms for the (n, d)-stencil problem such that any $\mathcal{A} \in \mathscr{C}_d$ for v processing elements satisfies the following properties: (i) each DAG node is evaluated once (i.e., recomputation is not allowed); (ii) no processing element computes more than ϵn^{d+1} DAG nodes, for some constant $0 < \epsilon < 1$; and (iii) all of the foldings of $\mathcal{A}$ on 2^j processing elements, $1 \leq j < \log v$, also belong to $\mathscr{C}_d$. Note that as before, this class of algorithms makes no assumptions on the input and output distributions. The following lemma establishes a lower bound on the communication complexity of the algorithms in $\mathscr{C}_d$.

LEMMA 4.10. *The communication complexity of any (n, d)-stencil algorithm in $\mathscr{C}_d$ when executed on $M(p, \sigma)$ is $\Omega(n^d / p^{(d-1)/d} + \sigma)$.*

PROOF. The bound for $\sigma = 0$ is proved in Theorem 5 of Scquizzato and Silvestri [2014], and it clearly extends to the case $\sigma > 0$. The additive σ term follows since at least one message is sent by some processing element. □

In what follows, we develop efficient network-oblivious algorithms for the (n, d)-stencil problem, for the special cases of $d = \{1, 2\}$. The generalization to values $d > 2$, and to other types of stencils, is left as an open problem.

4.4.1. The $(n, 1)$-Stencil Problem. The $(n, 1)$-stencil problem consists of the evaluation of a DAG shaped as a two-dimensional array of side n. We reduce the solution of the stencil problem to the computation of a diamond DAG. Specifically, we define a *diamond DAG* of side n as the intersection of a $(2n - 1, 1)$-stencil DAG with the following four half-planes: $i_0 + i_1 \geq (n - 1)$, $i_0 - i_1 \leq (n - 1)$, $i_0 - i_1 \geq -(n - 1)$, and $i_0 + i_1 \leq 3(n - 1)$ (i.e., the largest diamond included in the stencil).[7] It follows that an $(n, 1)$-stencil DAG can be partitioned into five full or truncated diamond DAGs of side less than n that can be evaluated in a suitable order, with the outputs of one DAG evaluation providing the inputs for subsequent DAG evaluations.

Our network-oblivious algorithm for the $(n, 1)$-stencil is specified on $M(n)$ and consists of five *stages*, where in each stage the whole $M(n)$ machine takes care of the evaluation of a distinct diamond DAG (full or truncated) according to the aforementioned partition. We require that all of the $O(n)$ inputs necessary for the evaluation of a diamond DAG are evenly distributed among the n VPs at the start of the stage in charge of the DAG. No matter how the inputs are assigned to the VPs at the beginning of the algorithm, the data movement required to guarantee the correct input distribution at the various stages can be accomplished in $O(1)$ 0-supersteps where each VP sends/receives $O(n)$ messages.

[7]We observe that our definition of diamond DAG is consistent with the one in Bilardi and Preparata [1997], whose edges are a superset of those of the diamond DAG defined in Aggarwal et al. [1990].

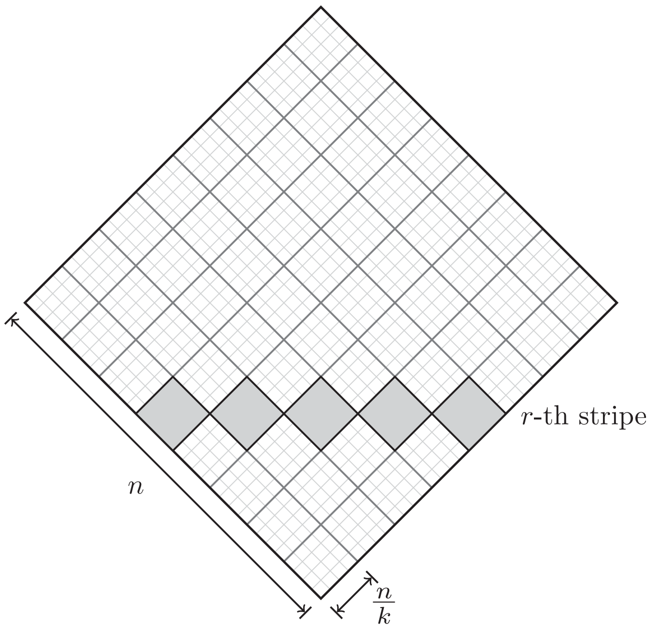

Fig. 1. The decomposition of the diamond DAG performed by our algorithm.

We now focus on the evaluation of the individual diamond DAGs. For ease of presentation, we consider the evaluation of a full diamond DAG of side n on $M(n)$. Simple yet tedious modifications are required for dealing with truncated or smaller diamond DAGs. We exploit the fact that this DAG can be decomposed recursively into smaller diamonds. Parallel algorithms for stencil computations based on this or similar decompositions are known [Chowdhury and Ramachandran 2008; Frigo and Strumpen 2009; Tang et al. 2011], but their focus is on optimizing processor cache efficiency rather than interprocessor communications.

Let $k = 2^{\lceil\sqrt{\log n}\rceil}$. The diamond DAG is partitioned into $2k-1$ horizontal stripes, each containing up to k diamonds of side n/k, as depicted in Figure 1. The DAG evaluation is accomplished into $2k-1$ nonoverlapping *phases*. In the r-th such phase, with $0 \leq r < 2k-1$, the diamonds in the r-th stripe are evaluated in parallel by distinct $M(n/k)$ submachines formed by disjoint segments of consecutively numbered VPs.[8] At the beginning of each phase, a 0-superstep is executed to provide the VPs of each $M(n/k)$ submachine with the appropriate input—that is, the immediate predecessors (if any) of the diamond assigned to the submachine. In this superstep, each VP sends/receives $O(1)$ messages. In each phase, the diamonds of side n/k are evaluated recursively.

In general, at the i-th recursive level, with $i \geq 1$, a total of $(2k-1)^i$ nonoverlapping phases are executed where diamonds of side $n_i = n/k^i$ are evaluated in parallel by distinct $M(n_i)$ submachines. Each such phase starts with a superstep of label $(i-1)\cdot\log k$ to provide each $M(n_i)$ with the appropriate input. In turn, the evaluation of a diamond of side n_i within an $M(n_i)$ submachine is performed recursively by partitioning its nodes into $2k-1$ horizontal stripes of diamonds of side $n_{i+1} = n/k^{i+1}$ that are evaluated

[8]We observe that some $M(n/k)$ submachines may not be assigned to subproblems, as the number of diamonds in a stripe could be smaller than k. To comply with the requirement that in the algorithm execution the sequence of superstep labels is the same at each processing element, we assume that idle $M(n/k)$ submachines are assigned dummy diamonds of side n/k to be evaluated.

in $2k-1$ nonoverlapping phases by $M(n_{i+1})$ submachines, with each phase starting with a superstep of label $i \cdot \log k$ where each VP sends/receives $O(1)$ messages (and thus each processor sends/receives $O(n/p)$ messages). The recursion ends at level $\tau = \lfloor \log_k n \rfloor$, which is the first level where the diamond of side n_τ becomes smaller than k. If $n_\tau > 1$, each diamond of side n_τ assigned to an $M(n_\tau)$ submachine is evaluated straightforwardly in $2n_\tau - 1$ supersteps of label $\tau \cdot \log k$. Instead, if $n_\tau = 1$, at recursion level τ each VP independently evaluates a 1-node diamond, and no communication is required.

By unfolding the recursion, one can easily see that the evaluation of a diamond DAG of side n entails, overall, $(2k-1)^i$ supersteps of label $(i-1)\cdot \log k$, for $1 \le i \le \tau$, and if $n_\tau > 1$, $(2k-1)^\tau n_\tau$ supersteps of label $\tau \cdot \log k$. In each of these supersteps, every VP sends/receives $O(1)$ messages.

To guarantee $(\Theta(1), n)$-wiseness of our algorithm, we assume that suitable dummy messages are added in each superstep to make each VP exchange the same number of messages.

THEOREM 4.11. *The communication complexity of the preceding network-oblivious algorithm for the* $(n, 1)$*-stencil problem when executed on* $M(p, \sigma)$ *is*

$$H_{\text{1-stencil}}(n, p, \sigma) = O\left(n 4^{\sqrt{\log n}}\right),$$

for every $1 < p \le n$ *and* $0 \le \sigma = O(n/p)$*. The algorithm is* $(\Theta(1), n)$*-wise and* $\Omega(1/4^{\sqrt{\log n}})$*-optimal with respect to* $\mathscr{C}_1$ *on any* $M(p, \sigma)$ *with* $1 < p \le n$ *and* $\sigma = O(n/p)$*.*

PROOF. As observed earlier, the communication required at the beginning of each of the five stages contributes an additive factor $O(n)$ to the communication complexity, and hence it is negligible. Let us then concentrate on the communication complexity for one diamond DAG evaluation. Recall that $\tau = \lfloor \log_k n \rfloor$. First suppose that $p \le k^\tau$. Observe that at every recursion level i, with $0 \le i < \lceil \log_k p \rceil$, the evaluation of each diamond of side $n_i = n/k^i$ is performed by $p/k^i > 1$ processors, and each processor sends/receives $O(n/p)$ messages in $O(1)$ supersteps for processing the inputs and outputs of these subproblems; on the other hand, at every recursion level i, with $\lceil \log_k p \rceil \le i \le \tau$, each diamond of side n/k^i is evaluated by a single processor of $M(p, \sigma)$ and no communication takes place. Thus, the communication complexity satisfies the recurrence relation:

$$H_{\text{1-stencil}}(n, p, \sigma) = \begin{cases} (2k-1) H_{\text{1-stencil}}(n/k, p/k, \sigma) + O(n/p + \sigma) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

This recurrence has the following solution,

$$\begin{aligned} H_{\text{1-stencil}}(n, p, \sigma) &= O\left(\sum_{i=0}^{\lceil \log_k p \rceil - 1} (2k-1)^{i+1} \left(\frac{n}{p} + \sigma \right) \right) \\ &= O\left((2k)^{\log_k p + 1} \frac{n}{p} \right) \\ &= O\left(n 2^{\log_k p} k \right) \\ &= O\left(n 4^{\sqrt{\log n}} \right), \end{aligned}$$

where we exploited the upper bound on σ. Instead, if $k^\tau < p \le n$, we have that at every recursion level i, with $0 \le i \le \tau$, the evaluation of each diamond of side $n_i = n/k^i$ is performed by $p/k^i > 1$ processors. Then, by the preceding discussion and recalling that for $i = \tau$, diamonds of side $n_\tau = n/k^\tau$ are evaluated straightforwardly in $2n_\tau - 1$ supersteps, we obtain

$$\begin{aligned} H_{1\text{-stencil}}(n, p, \sigma) &= O\left(\sum_{i=0}^{\tau-1}(2k-1)^{i+1}\left(\frac{n}{p}+\sigma\right)\right) + O\left((2k-1)^\tau \frac{n}{k^\tau}\left(\frac{n}{p}+\sigma\right)\right) \\ &= O\left((2k)^\tau \frac{n}{k^\tau}\frac{n}{p}\right) \\ &= O(n2^\tau k) \\ &= O\left(n4^{\sqrt{\log n}}\right), \end{aligned}$$

where we exploited the upper bound on σ and the fact that $p > k^\tau$, and hence, by definition of τ, $n/p < k$. The wiseness is ensured by the dummy messages. It is easy to see that the algorithm complies with the requirements for belonging to $\mathscr{C}_1$, and hence the claimed optimality is a consequence of Lemma 4.10, and the theorem follows. □

Finally, we show that the network-oblivious algorithm for the $(n, 1)$-stencil problem achieves $\Omega(1/4^{\sqrt{\log n}})$-optimality on the D-BSP as well, for wide ranges of machine parameters.

Corollary 4.12. *The preceding network-oblivious algorithm for the $(n, 1)$-stencil problem is $\Omega(1/4^{\sqrt{\log n}})$-optimal with respect to $\mathscr{C}_1$ on any D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ machine with $1 < p \leq n$, nonincreasing g_i's and ℓ_i/g_i's, and $\ell_0/g_0 = O(n/p)$.*

Proof. The corollary follows by Theorem 4.11 and by applying Theorem 3.4 with $p^\star = n$, $\sigma_i^m = 0$, and $\sigma_i^M = \Theta(n/2^i)$. □

We remark that a tighter analysis of the algorithm and/or the adoption of different values for the recursion degree k, still independent of p and σ, may yield slightly better efficiency. The two techniques recently proposed in Tang et al. [2015] to improve the parallelism of recursive cache-efficient dynamic programming algorithms might also have the potential to lead to improved bounds. However, it is an open problem to devise a network-oblivious algorithm that is $\Theta(1)$-optimal on the D-BSP for wide ranges of the machine parameters.

4.4.2. The $(n, 2)$-Stencil Problem. In this section, we present a network-oblivious algorithm for the $(n, 2)$-stencil problem, which requires the evaluation of a DAG shaped as a three-dimensional array of side n. Both the algorithm and its analysis are a suitable adaptation of the ones for the $(n, 1)$-stencil problem. To evaluate a three-dimensional domain, we make use of two types of subdomains that intuitively play the same role as the diamond for the $(n, 1)$-stencil: the octahedron and the tetrahedron. An octahedron of side n is the intersection of a $(2n - 1, 2)$-stencil with the following eight half-spaces: $i_0 + i_2 \geq (n-1)$, $i_0 - i_2 \leq (n-1)$, $i_0 - i_2 \geq -(n-1)$, $i_0 + i_2 \leq 3(n-1)$, $i_0 + i_1 \geq (n-1)$, $i_0 - i_1 \leq (n-1)$, $i_0 - i_1 \geq -(n-1)$, and $i_0 + i_1 \leq 3(n-1)$; a tetrahedron of side n is the intersection of a $(2n-1, 2)$-stencil with the following four half-spaces: $i_0 + i_1 \geq (n-1)$, $i_0 - i_1 \geq (n-1)$, $i_1 + i_2 \leq 2(n-1)$, and $i_1 - i_2 \leq 0$.

As shown in Bilardi and Preparata [1997], a three-dimensional array of side n can be partitioned into 17 instances of (possibly truncated) octahedra or tetrahedra of side n (see Figure 6 of Bilardi and Preparata [1997]). Our network-oblivious algorithm exploits this partition and is specified on $M(n^2)$. It consists of 17 stages, where in each stage the VPs take care of the evaluation of one polyhedra of the partition. We assume that at the beginning of the algorithm, the inputs are evenly distributed among the

n^2 VPs and also impose that the inputs of each stage be evenly distributed among the VPs. The data movement required to guarantee the correct input distribution for each stage can be accomplished in $O(1)$ 0-supersteps, where each VP sends/receives $O(1)$ messages.

Let $k = 2^{\lceil\sqrt{\log n}\rceil}$. An octahedron of side n can be partitioned into octahedra and tetrahedra of side n/k in $\log k$ steps, where the i-th such step, with $1 \leq i \leq \log k$, refines a partition of the initial octahedron into octahedra or tetrahedra of side $n/2^{i-1}$ by decomposing each of these polyhedra into smaller ones of side $n/2^i$, according to the scheme depicted in Figure 5 of Bilardi and Preparata [1997]. The final partition is obtained at the end of the $\log k$-th step. The octahedra and tetrahedra of the final partition can be grouped in horizontal stripes in such a way that the polyhedra of each stripe can be evaluated in parallel. Consider first the set of octahedra of the partition. It can be seen that the projection of these octahedra on the (i_0, i_2)-plane coincides with the decomposition of the diamond DAG depicted in Figure 1. As a consequence, we can identify $2k-1$ horizontal stripes of octahedra, where each stripe contains up to k^2 octahedra of side n/k. Moreover, the interleaving of octahedra and tetrahedra in the basic decompositions of Figure 5 of Bilardi and Preparata [1997] implies that there is a stripe of tetrahedra between each pair of consecutive stripes of octahedra. Hence, there are also $(2k-1)-1$ horizontal stripes of tetrahedra, each containing up to k^2 tetrahedra of side n/k. Overall, the octahedron of side n is partitioned into $4k-3$ horizontal stripes of at most k^2 polyhedra of side n/k each, where stripes of octahedra are interleaved with stripes of tetrahedra. With a similar argument, one can derive a partition of a tetrahedron of side n into $2k-1 \leq 4k-3$ horizontal stripes of at most k^2 polyhedra of side n/k each, where stripes of octahedra are interleaved with stripes of tetrahedra.

Once the preceding preliminaries have been established, the network-oblivious algorithm to evaluate a three-dimensional array of side n on $M(n^2)$ follows closely from the recursive strategy used for the $(n,1)$-stencil problem: the evaluation of an octahedron is accomplished in $4k-3$ nonoverlapping phases, in each of which the polyhedra (either octahedra or tetrahedra) of side n/k in one horizontal stripe of the partition described earlier are recursively evaluated in parallel by distinct $M(n^2/k^2)$ submachines formed by disjoint segments of consecutively numbered VPs; a tetrahedron of side n can be evaluated through a recursive strategy similar to the one for the octahedron within the same complexity bounds. As usual, we add to each superstep $O(1)$ dummy messages per VP to guarantee $(\Theta(1), n^2)$-wiseness.

THEOREM 4.13. *The communication complexity of the preceding network-oblivious algorithm for the $(n,2)$-stencil problem when executed on $M(p,\sigma)$ is*

$$H_{\text{2-stencil}}(n, p, \sigma) = O\left(\frac{n^2}{\sqrt{p}} 8^{\sqrt{\log n}}\right),$$

for every $1 < p \leq n^2$ and $0 \leq \sigma = O(n^2/p)$. The algorithm is $(\Theta(1), n^2)$-wise and $\Omega(1/8^{\sqrt{\log n}})$-optimal with respect to $\mathscr{C}_2$ on any $M(p,\sigma)$ with $1 < p \leq n^2$ and $\sigma = O(n^2/p)$.

PROOF. Let $H_{\text{octahedron}}(n, p, \sigma)$ be the communication complexity required by the recursive strategy presented earlier for the evaluation of an octahedron of side n, when executed on $M(p,\sigma)$. The recursion depth of that strategy is $\tau = \lfloor \log_k n \rfloor$. First suppose that $p \leq k^{2\tau}$. At every recursion level i, with $0 \leq i < \lceil (\log_k p)/2 \rceil$, the evaluation of each polyhedron of side $n_i = n/k^i$ is performed by $p/k^{2i} > 1$ processors, and each processor sends/receives $O(n^2/p)$ messages in $O(1)$ supersteps for processing the inputs and outputs of these subproblems; on the other hand, at every recursion level i, with

$\lceil(\log_k p)/2\rceil \le i \le \tau$, each polyhedron of side n/k^i is evaluated by a single processor of $M(p,\sigma)$ and no communication takes place. Thus, the communication complexity satisfies the recurrence relation:

$$H_{\text{octahedron}}(n,p,\sigma) = \begin{cases} (4k-3)H_{\text{octahedron}}(n/k, p/k^2, \sigma) + O\big(n^2/p+\sigma\big) & \text{if } p > 1, \\ 0 & \text{otherwise.} \end{cases}$$

This recurrence has the following solution,

$$\begin{aligned} H_{\text{octahedron}}(n,p,\sigma) &= O\left(\sum_{i=0}^{\lceil(\log_k p)/2\rceil-1} (4k-3)^{i+1}\left(\frac{n^2}{p}+\sigma\right)\right) \\ &= O\left((4k)^{(\log_k p)/2+1}\frac{n^2}{p}\right) \\ &= O\left(\frac{n^2}{\sqrt{p}}2^{\log_k p}k\right) \\ &= O\left(\frac{n^2}{\sqrt{p}}4^{\sqrt{\log n}}\right), \end{aligned}$$

where we used the hypothesis $\sigma = O(n^2/p)$. Instead, when $k^{2\tau} < p \le n^2$, we have that at every recursion level i, with $0 \le i \le \tau$, the evaluation of each polyhedron of side $n_i = n/k^i$ is performed by $p/k^{2i} > 1$ processors. Then, since for $i = \tau$ the polyhedra of side $n_\tau = n/k^\tau$ are evaluated straightforwardly in $\Theta(n_\tau)$ supersteps, we obtain

$$\begin{aligned} H_{\text{octahedron}}(n,p,\sigma) &= O\left(\sum_{i=0}^{\tau-1}(4k-3)^{i+1}\left(\frac{n^2}{p}+\sigma\right)\right) + O\left((4k-3)^\tau\frac{n}{k^\tau}\left(\frac{n^2}{p}+\sigma\right)\right) \\ &= O\left((4k)^\tau\frac{n}{k^\tau}\frac{n^2}{p}\right) \\ &= O\left(\frac{n^2}{\sqrt{p}}4^\tau k\right) \\ &= O\left(\frac{n^2}{\sqrt{p}}8^{\sqrt{\log n}}\right), \end{aligned}$$

where we used the hypothesis $\sigma = O(n^2/p)$ and the inequalities $n/k^\tau < k$ and $k^{2\tau} < p$. Similar upper bounds on the communication complexity can be proved for the evaluation of a tetrahedron of side n and for the evaluation of truncated octahedra or tetrahedra.

Recall that the algorithm for the $(n,2)$-stencil problem consists of 17 stages, where in each stage the VPs take care of the evaluation of one (possibly truncated) octahedron or tetrahedron of side n, and that the data movement that ensures the correct input distribution for each stage can be accomplished in $O(1)$ 0-supersteps, where each VP sends/receives $O(1)$ messages. This implies that

$$H_{\text{2-stencil}}(n,p,\sigma) = O\left(\frac{n^2}{\sqrt{p}}8^{\sqrt{\log n}}\right).$$

Since the strategies for the evaluation of (possibly truncated) octahedra or tetrahedra can be made $(\Theta(1), n^2)$-wise, through the introduction of suitable dummy messages, the overall algorithm is also $(\Theta(1), n^2)$-wise. Moreover, the algorithm complies with the requirements for belonging to $\mathscr{C}_2$, and hence the claimed optimality is a consequence of Lemma 4.10. □

Corollary 4.14. *The preceding network-oblivious algorithm for the* $(n, 2)$*-stencil problem is* $\Omega(1/8^{\sqrt{\log n}})$*-optimal with respect to* $\mathscr{C}_2$ *on any D-BSP*$(p, \boldsymbol{g}, \boldsymbol{\ell})$ *machine with* $1 < p \leq n^2$*, nonincreasing* g_i*'s and* ℓ_i/g_i*'s, and* $\ell_0/g_0 = O(n^2/p)$*.*

Proof. The corollary follows by Theorem 4.13 and by applying Theorem 3.4 with $p^\star = n^2$, $\sigma_i^m = 0$, and $\sigma_i^M = \Theta(n^2/2^i)$. □

4.5. Limitations of the Oblivious Approach

In this section, we establish a negative result by showing that for the broadcast problem, defined next, a network-oblivious algorithm can achieve $O(1)$-optimality on $M(p, \sigma)$ only for very limited ranges of σ. Let $V[0, 1, \ldots, n-1]$ be a vector of n entries. The *n-broadcast problem* requires copying the value $V[0]$ into all other $V[i]$'s. Let $\mathscr{C}$ denote the class of static algorithms for the n-broadcast problem such that any $\mathcal{A} \in \mathscr{C}$ for v processing elements satisfies the following properties: (i) at least ϵv processing elements hold entries of V, for some constant $0 < \epsilon \leq 1$, and the distribution of the entries of V among the processing elements cannot change during the execution of the algorithm, and (ii) all of the foldings of $\mathcal{A}$ on 2^j processing elements, $1 \leq j < \log v$, also belong to $\mathscr{C}$. The following theorem establishes a lower bound on the communication complexity of the algorithms in $\mathscr{C}$.

Theorem 4.15. *The communication complexity of any n-broadcast algorithm in* $\mathscr{C}$ *when executed on* $M(p, \sigma)$*, with* $1 < p \leq n$ *and* $\sigma \geq 0$*, is* $\Omega(\max\{2, \sigma\} \log_{\max\{2,\sigma\}} p)$*.*

Proof. Let $\mathcal{A}$ be an algorithm in $\mathscr{C}$. Suppose that the execution of $\mathcal{A}$ on $M(p, \sigma)$ requires t supersteps, and let p_i denote the number of processors that "know" the value $V[0]$ by the end of the i-th superstep, for $1 \leq i \leq t$. Clearly, $p_0 = 1$ and $p_t \geq \epsilon p$, since by definition of $\mathscr{C}$, at least ϵp processors hold entries of V to be updated with the value $V[0]$. During the i-th superstep, $p_i - p_{i-1}$ new processors get to know $V[0]$. Since at the beginning of this superstep only p_{i-1} processors know the value, we conclude that the superstep involves an h-relation with $h \geq \lceil (p_i - p_{i-1})/p_{i-1} \rceil$. Therefore, the communication complexity of $\mathcal{A}$ is

$$H_{\mathcal{A}}(n, p, \sigma) \geq \sum_{i=1}^{t} \left(\left\lceil \frac{p_i - p_{i-1}}{p_{i-1}} \right\rceil + \sigma \right) = \sum_{i=1}^{t} \left(\left\lceil \frac{p_i}{p_{i-1}} \right\rceil - 1 + \sigma \right).$$

Assuming without loss of generality that the p_i's are strictly increasing, we obtain

$$H_{\mathcal{A}}(n, p, \sigma) = \Omega\left(t \max\{2, \sigma\} + \sum_{i=1}^{t} \frac{p_i}{p_{i-1}} \right).$$

Since $\prod_{i=1}^{t} p_i/p_{i-1} = p_t$, it follows that $\sum_{i=1}^{t} p_i/p_{i-1}$ is minimized for $p_i/p_{i-1} = (p_t)^{1/t} \geq (\epsilon p)^{1/t}$, for $1 \leq i \leq t$. Hence,

$$H_{\mathcal{A}}(n, p, \sigma) = \Omega\left(t \left(\max\{2, \sigma\} + p^{1/t} \right) \right). \tag{7}$$

Standard calculus shows that the right-hand side is minimized (to within a constant factor) by choosing $t = \Theta(\log_{\max\{2,\sigma\}} p)$, and the claim follows. □

The preceding lower bound is tight. Consider the following $M(p, \sigma)$ algorithm for n-broadcast. Let the entries of V be evenly distributed among the processors, with

$V[0]$ held by processor P_0. For convenience, we assume that n is a power of 2. Let κ be the smallest power of 2 greater than or equal to $\max\{2, \sigma\}$. The algorithm consists of $\lceil \log_\kappa p \rceil$ supersteps: in the i-th superstep, with $0 \le i < \lceil \log_\kappa p \rceil$, each P_{jp/κ^i}, with $0 \le j < \kappa^i$, sends the value $V[0]$ to $P_{(\kappa j+\ell)p/\kappa^{i+1}}$, for each $0 \le \ell < \kappa$. (When $\log_\kappa p$ is not an integer value, in the last superstep only values of ℓ that are multiples of κ^{i+1}/p are used.) It is immediate to see that the algorithm belongs to $\mathscr{C}$ and that its communication complexity on $M(p, \sigma)$ is

$$H_{\text{broad}_\kappa}(n, p, \sigma) = O((\kappa + \sigma)\log_\kappa p) = O\big(\max\{2, \sigma\}\log_{\max\{2,\sigma\}} p\big).$$

Therefore, the algorithm is $O(1)$-optimal. Observe that the algorithm is aware of parameter σ, and, in fact, this knowledge is crucial to achieve optimality. To see this, we prove that *any* network-oblivious algorithm for n-broadcast can be $\Theta(1)$-optimal on $M(p, \sigma)$, only for limited ranges of σ. Let $H^\star(n, p, \sigma)$ denote the best communication complexity achievable on $M(p, \sigma)$ by an algorithm for n-broadcast belonging to $\mathscr{C}$. By the preceding discussion, we know that $H^\star(n, p, \sigma) = \Theta(\max\{2, \sigma\}\log_{\max\{2,\sigma\}} p)$. Let $\mathcal{A} \in \mathscr{C}$ be a network-oblivious algorithm for n-broadcast specified on $M(v(n))$. For every $1 < p \le v(n)$ and $0 \le \sigma_1 \le \sigma_2$, we define the maximum slowdown incurred by $\mathcal{A}$ with respect to the best $M(p, \sigma)$-algorithm in $\mathscr{C}$, for $\sigma \in [\sigma_1, \sigma_2]$, as

$$\text{GAP}_\mathcal{A}(n, p, \sigma_1, \sigma_2) = \max_{\sigma_1 \le \sigma \le \sigma_2} \left\{ \frac{H_\mathcal{A}(n, p, \sigma)}{H^\star(n, p, \sigma)} \right\}.$$

THEOREM 4.16. *Let $\mathcal{A} \in \mathscr{C}$ be a network-oblivious algorithm for n-broadcast specified on $M(v(n))$. For every $1 < p \le v(n)$ and $0 \le \sigma_1 \le \sigma_2$, we have*

$$\text{GAP}_\mathcal{A}(n, p, \sigma_1, \sigma_2) = \Omega\left(\frac{\log \max\{2, \sigma_2\}}{\log \max\{2, \sigma_1\} + \log\log \max\{2, \sigma_2\}} \right).$$

PROOF. The definition of function GAP implies that

$$\text{GAP}_\mathcal{A}(n, p, \sigma_1, \sigma_2) = \Omega\left(\frac{H_\mathcal{A}(n, p, \sigma_1)}{H^\star(n, p, \sigma_1)} + \frac{H_\mathcal{A}(n, p, \sigma_2)}{H^\star(n, p, \sigma_2)} \right).$$

Let t be the number of supersteps executed by the folding of $\mathcal{A}$ on $M(p, \sigma)$, and note that since $\mathcal{A}$ is network oblivious, this number cannot depend on σ. By arguing as in the proof of Theorem 4.15 (see Inequality (7)), we get that $H_\mathcal{A}(n, p, \sigma) = \Omega(t(\max\{2, \sigma\} + p^{1/t}))$, for any σ, and hence $\text{GAP}_\mathcal{A}(n, p, \sigma_1, \sigma_2)$ is bounded from below by

$$\Omega\left(\frac{t(\max\{2, \sigma_1\} + p^{1/t})}{\max\{2, \sigma_1\}\log_{\max\{2,\sigma_1\}} p} + \frac{t(\max\{2, \sigma_2\} + p^{1/t})}{\max\{2, \sigma_2\}\log_{\max\{2,\sigma_2\}} p} \right),$$

which is minimized for $t = \Theta(\log p/(\log \max\{2, \sigma_1\} + \log\log \max\{2, \sigma_2\}))$. Substituting this value of t in the preceding formula yields the stated result. □

An immediate consequence of the preceding theorem is that if a network-oblivious algorithm for n-broadcast is $\Theta(1)$-optimal on $M(p, \sigma)$, it cannot be simultaneously $\Theta(1)$-optimal on an $M(p, \sigma')$, for any σ' sufficiently larger than σ. A similar limitation of the optimality of a network-oblivious algorithm for n-broadcast can be argued with respect to its execution on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$.

5. EXTENSION TO THE OPTIMALITY THEOREM

The optimality theorem of Section 3 makes crucial use of the wiseness property. Broadly speaking, a network-oblivious algorithm is $(\Theta(1), p)$-wise when the communication performed in the various supersteps is somewhat balanced in the sense that the maximum number of messages sent/received by a virtual processor does not differ significantly from the average number of messages sent/received by other virtual processors belonging to the same region of suitable size. Although there exist $(\Theta(1), p)$-wise network-oblivious algorithms for a number of important problems, as shown in Section 4, there are cases where wiseness may not be guaranteed.

As a simple example of poor wiseness, consider a network-oblivious algorithm $\mathcal{A}$ for $M(n)$ consisting of one 0-superstep where VP_0 sends n messages to $\mathrm{VP}_{n/2}$. Fix p with $2 \leq p \leq n$. Clearly, for each $1 \leq j \leq \log p$, we have that $H_{\mathcal{A}}(n, 2^j, 0) = n$, and hence the algorithm is (α, p)-wise only for $\alpha = O(1/p)$. When executed on a D-BSP$(p, \boldsymbol{g}, \mathbf{0})$, the communication time of the algorithm is ng_0. However, as already observed in Bilardi et al. [2007a], under reasonable assumptions the communication time of the algorithm's execution on the D-BSP can be improved by first evenly spreading the n messages among clusters of increasingly larger size that include the sender, then gathering the messages within clusters of increasingly smaller size that include the receiver. Motivated by this observation, we introduce a more effective protocol to execute network-oblivious algorithms on the D-BSP. By employing this protocol, we are able to prove an alternative optimality theorem that requires a much weaker property than wiseness at the expense of a slight (polylogarithmic) loss of efficiency.

Let $\mathcal{A}$ be a network-oblivious algorithm specified on $M(v(n))$, and consider its execution on a D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$, with $1 \leq p \leq v(n)$. As before, each D-BSP processor P_j, with $0 \leq j < p$, carries out the operations of the $v(n)/p$ consecutively numbered VPs of $M(v(n))$ starting with $\mathrm{VP}_{j(v(n)/p)}$. However, the communication required by each superstep is now performed on D-BSP more effectively by enforcing a suitable balancing. More precisely, each i-superstep s of $\mathcal{A}$, with $0 \leq i < \log p$, is executed on the D-BSP through the following protocol, which we will call the *ascend-descend protocol*:

(1) *Computation phase*: Each D-BSP processor performs the local computations of its assigned virtual processors.
(2) *Ascend phase*: For $k = \log p - 1$ down to $i + 1$: within each k-cluster Γ_k, the messages that originate in Γ_k but are destined outside Γ_k are evenly distributed among the $p/2^k$ processors of Γ_k.
(3) *Descend phase*: For $k = i$ to $\log p - 1$: within each k-cluster Γ_k, the messages currently residing in Γ_k are evenly distributed among the processors of the $(k+1)$-clusters inside Γ_k that contain their final destinations.

Observe that each iteration of the ascend/descend phases requires a prefix-like computation to assign suitable intermediate destinations to the messages to guarantee their even distribution in the appropriate clusters.

LEMMA 5.1. *Let $\mathcal{A}$ be a network-oblivious algorithm specified on $M(v(n))$, and consider its execution on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$, with $1 < p \leq v(n)$, using the ascend-descend protocol. Let s be an i-superstep, for some $0 \leq i < \log p$, and let ξ_s be the sequence of supersteps employed by the protocol for executing s. Then, for every $i < k < \log p$, ξ_s comprises $O(1)$ k-supersteps of degree $O(2^k h^s_{\mathcal{A}}(n, 2^k)/p)$ and $O(\log p)$ k-supersteps each of constant degree.*

PROOF. Consider iteration k of the ascend phase of the protocol, with $i + 1 \leq k \leq \log p - 1$, and a k-cluster Γ_k. As invariant at the beginning of the iteration, we have that the at most $h^s_{\mathcal{A}}(n, 2^{k+1})$ messages originating in each $k+1$-cluster Γ' included in Γ_k and destined outside Γ_k are evenly distributed among the processors of Γ'. Hence, the even distribution of these messages among the $p/2^k$ processors of Γ_k requires a prefix-like computation and an $O(\lceil 2^{k+1} h^s_{\mathcal{A}}(n, 2^{k+1})/p \rceil)$-relation within Γ_k. Consider now iteration k of the descend phase of the protocol, with $i \leq k \leq \log p - 1$, and a k-cluster Γ_k. As invariant at the beginning of the iteration, we have that the at most $2h^s_{\mathcal{A}}(n, 2^{k+1})$ messages to be moved in the iteration are evenly distributed among the processors of Γ_k. Since each $(k+1)$-cluster included in Γ_k receives at most $h^s_{\mathcal{A}}(n, 2^{k+1})$ messages, the iteration requires a prefix-like computation and an $O(\lceil 2^{k+1} h^s_{\mathcal{A}}(n, 2^{k+1})/p \rceil)$-relation within Γ_k. The lemma follows, as each prefix-like computation in a k-cluster can be performed in $O(\log p)$ k-supersteps of constant degree (e.g., using a straightforward tree-based strategy [JáJá 1992]). □

We now define the notion of *fullness*, which is weaker than wiseness but which still allows us to port the optimality of network-oblivious algorithms with respect to the evaluation model onto the execution machine model, at the price of some loss of efficiency.

Definition 5.2. A static network-oblivious algorithm $\mathcal{A}$ specified on $M(v(n))$ is said to be (γ, p)*-full*, for some $\gamma > 0$ and $1 < p \leq v(n)$, if the folding of $\mathcal{A}$ on $M(2^j, 0)$ satisfies

$$\sum_{i=0}^{j-1} F^i_{\mathcal{A}}(n, 2^j) \geq \gamma \frac{p}{2^j} \sum_{i=0}^{j-1} S^i_{\mathcal{A}}(n),$$

for every $1 \leq j \leq \log p$ and input size n.

It is easy to see that a $(\Theta(1), p)$-wise network-oblivious algorithm $\mathcal{A}$ is also $(\Theta(1), p)$-full as long as $h^s_{\mathcal{A}}(n, p) \geq 1$, for every i-superstep s of $\mathcal{A}$ and every $1 < p \leq v(n)$. On the other hand, a $(\Theta(1), p)$-full algorithm is not necessarily $(\Theta(1), p)$-wise, as witnessed by the previously mentioned network-oblivious algorithm consisting of a single 0-superstep where VP_0 sends n messages to $\mathrm{VP}_{n/2}$, which is $(\Theta(1), p)$-full but not $(\Theta(1), p)$-wise, for any $2 \leq p \leq n$. In this sense, (γ, p)-fullness is a weaker condition than $(\Theta(1), p)$-wiseness.

The following theorem shows that when (γ, p)-full algorithms are executed on the D-BSP using the ascend-descend protocol, optimality in the evaluation model is preserved on the D-BSP within a polylogarithmic factor. As in Section 3, let $\mathcal{C}$ denote a class of static algorithms solving a problem Π, with the property that for any algorithm $\mathcal{A} \in \mathcal{C}$ for v processing elements, all of its foldings on 2^j processing elements, for each $1 \leq j < \log v$, also belong to $\mathcal{C}$.

THEOREM 5.3. *Let $\mathcal{A} \in \mathcal{C}$ be a $(\gamma, p^\star)$-full network-oblivious algorithm for some $\gamma > 0$ and $p^\star$ a power of 2. Let also $\{\sigma^m_0, \sigma^m_1, \ldots, \sigma^m_{\log p^\star - 1}\}$ and $\{\sigma^M_0, \sigma^M_1, \ldots, \sigma^M_{\log p^\star - 1}\}$ be two vectors of nonnegative values, with $\sigma^m_j \leq \sigma^M_j$, for every $0 \leq j < \log p^\star$. If $\mathcal{A}$ is β-optimal on $M(2^j, \sigma)$ with respect to $\mathcal{C}$, for $\sigma^m_{j-1} \leq \sigma \leq \sigma^M_{j-1}$ and $1 \leq j \leq \log p^\star$, then for every p power of 2, $p \leq p^\star$, $\mathcal{A}$ is $\Theta(\beta/((1 + 1/\gamma)\log^2 p))$-optimal on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ with respect to $\mathcal{C}$ when executed with the ascend-descend protocol, as long as*

—*the execution of $\mathcal{A}$ on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ using the ascend-descend protocol is in $\mathcal{C}$;*
—*$g_i \geq g_{i+1}$ and $\ell_i/g_i \geq \ell_{i+1}/g_{i+1}$, for $0 \leq i < \log p - 1$;*
—*$\max_{1 \leq k \leq \log p}\{\sigma^m_{k-1} 2^k/p\} \leq \ell_i/g_i \leq \min_{1 \leq k \leq \log p}\{\sigma^M_{k-1} 2^k/p\}$, for $0 \leq i < \log p$.*

PROOF. Consider the execution of $\mathcal{A}$ on a D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ using the ascend-descend protocol. Let $\tilde{\mathcal{A}}$ denote the actual sequence of supersteps performed on the D-BSP in this execution of $\mathcal{A}$. Note that once the D-BSP parameters are fixed, $\tilde{\mathcal{A}}$ can be regarded as a network-oblivious algorithm specified on $M(p)$. Clearly, any optimality considerations on the communication time of the execution of $\tilde{\mathcal{A}}$ (regarded as a network-oblivious algorithm) on D-BSP($p, \boldsymbol{g}, \boldsymbol{\ell}$) using the standard protocol will also apply to the communication time of the execution of $\mathcal{A}$ on D-BSP($p, \boldsymbol{g}, \boldsymbol{\ell}$) using the ascend-descend protocol, since the communication time on $\mathcal{A}$ and $\tilde{\mathcal{A}}$ is the same.

We will assess the degree of optimality of the communication time of $\tilde{\mathcal{A}}$ by resorting to Theorem 3.4. This entails analyzing the communication complexity of $\tilde{\mathcal{A}}$ on $M(2^j, \sigma)$, for any $1 \le j \le \log p$, and determining its wiseness. Focus on $M(2^j, \sigma)$ for some $1 \le j \le \log p$, and consider an arbitrary i-superstep s of $\mathcal{A}$, for some $0 \le i < j$. Let ξ_s be the sequence of supersteps in $\tilde{\mathcal{A}}$ executed in the ascend and descend phases associated with superstep s. From Lemma 5.1, we know that for every $i < k < \log p$, ξ_s comprises $O(1)$ k-supersteps of degree $O(2^k h^s_{\mathcal{A}}(n, 2^k)/p)$ and $O(\log p)$ k-supersteps each of constant degree. Now, in the execution on $M(2^j, \sigma)$, a k-superstep with $k \ge j$ becomes local to the processors and does not contribute to the communication complexity. Since each processor of $M(2^j, \sigma)$ corresponds to $p/2^j$ processors of $M(p)$, the communication complexity on $M(2^j, \sigma)$ contributed by the sequence ξ_s is

$$O\left(\sum_{k=i+1}^{j-1}\left(\frac{p}{2^j}\left(\frac{2^k}{p}h^s_{\mathcal{A}}(n, 2^k) + \log p\right) + \sigma \log p\right)\right).$$

Therefore, since $h^s_{\mathcal{A}}(n, 2^k) \le 2^{j-k} h^s_{\mathcal{A}}(n, 2^j)$, the preceding summation is upper bounded by

$$O\left(\sum_{k=i+1}^{j-1}\left(h^s_{\mathcal{A}}(n, 2^j) + \frac{p\log p}{2^j} + \sigma \log p\right)\right) = O\Big(\Big(h^s_{\mathcal{A}}(n, 2^j) + \frac{p}{2^j} + \sigma\Big)\log^2 p\Big).$$

Recall that $L^i_{\mathcal{A}}(n)$ denotes the set of i-supersteps executed by $\mathcal{A}$, and $S^i_{\mathcal{A}}(n) = |L^i_{\mathcal{A}}(n)|$. Thus, the communication complexity of $\tilde{\mathcal{A}}$ on $M(2^j, \sigma)$ can be written as

$$\begin{aligned}
H_{\tilde{\mathcal{A}}}(n, 2^j, \sigma) &= O\left(\sum_{i=0}^{j-1}\sum_{s\in L^i_{\mathcal{A}}(n)}\Big(h^s_{\mathcal{A}}(n, 2^j) + \frac{p}{2^j} + \sigma\Big)\log^2 p\right)\\
&= O\left(\log^2 p\left(\sum_{i=0}^{j-1}\sum_{s\in L^i_{\mathcal{A}}(n)}\left(h^s_{\mathcal{A}}(n, 2^j) + \sigma\right) + \sum_{i=0}^{j-1}\sum_{s\in L^i_{\mathcal{A}}(n)}\frac{p}{2^j}\right)\right)\\
&= O\left(\log^2 p\left(H_{\mathcal{A}}(n, 2^j, \sigma) + \frac{p}{2^j}\sum_{i=0}^{j-1} S^i_{\mathcal{A}}(n)\right)\right)\\
&= O((1 + 1/\gamma)\log^2 p \cdot H_{\mathcal{A}}(n, 2^j, \sigma)),
\end{aligned}$$

where the last inequality follows by the (γ, p^*)-fullness of $\mathcal{A}$.

The preceding inequality shows that algorithm $\tilde{\mathcal{A}}$ is $\beta/((1 + 1/\gamma)\log^2 p)$-optimal as a consequence of the β-optimality of $\mathcal{A}$. Let us now assess the wiseness of $\tilde{\mathcal{A}}$. Consider again the sequence ξ_s of supersteps of $\tilde{\mathcal{A}}$ associated with an arbitrary i-superstep s of $\mathcal{A}$, for some $0 \le i < \log p$. We know that for every $i < k < \log p$, ξ_s comprises $O(1)$ k-supersteps of degree $O(2^k h^s_{\mathcal{A}}(n, 2^k)/p)$ and $O(\log p)$ k-supersteps each of constant degree. Moreover, we can assume that suitable dummy messages are added so that in a k-superstep of degree $O(2^k h^s_{\mathcal{A}}(n, 2^k)/p)$ (respectively, degree $O(1)$) all processors of a $(k+1)$-cluster send $\Theta(2^k h^s_{\mathcal{A}}(n, 2^k)/p)$ (respectively, $\Theta(1)$) messages to the sibling $(k+1)$-cluster included in the same k-cluster. It is easy to see that the preceding considerations about the optimality of $\tilde{\mathcal{A}}$ remain unchanged, whereas $\tilde{\mathcal{A}}$ becomes $(\Theta(1), p)$-wise. Finally, we recall that $\tilde{\mathcal{A}}$ belongs to class $\mathscr{C}$ by hypothesis, and this is so even forcing it into being wise. Therefore, by applying Theorem 3.4 to $\tilde{\mathcal{A}}$, we can conclude that $\tilde{\mathcal{A}}$, and hence $\mathcal{A}$, is $\Theta(\beta/((1+1/\gamma)\log^2 p))$-optimal on a D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ with parameters satisfying the initial hypotheses. □

As remarked earlier, the fullness requirement is considerably less stringent than wiseness. Algorithmic strategies that could benefit from this weaker requirement might be, for example, those designed for processor networks characterized by low-bandwidth decompositions into subnets. Typical communication patterns arising in these strategies may not feature constant wiseness since at each level of the decomposition a small fraction of boundary processors communicates across subnets, whereas they may exhibit constant fullness as long as a sufficiently large number of messages are exchanged among these boundary processors.

We conclude this section by observing that the relation stated by Theorem 5.3 between optimality in the evaluation model and optimality in D-BSP can be tightened when the g_i and ℓ_i parameters of the D-BSP decrease geometrically. In this case, it is known that a prefix-like computation within a k-cluster, for $0 \leq k < \log p$, can be performed in $O(g_k + \ell_k)$ communication time (e.g., see Proposition 2.2.2 in Bilardi et al. [2007a]). Then, by a similar argument used to prove Theorem 5.3, it can be shown that a (γ, p)-full algorithm $\mathcal{A}$ that is β-optimal in the evaluation model becomes $\Theta(\beta/((1+1/\gamma)\log p))$-optimal when executed on the D-BSP, thus reducing by a factor $\log p$ the gap between the two optimality factors.

6. CONCLUSIONS

We introduced a framework to explore the design of network-oblivious algorithms—that is, algorithms that run efficiently on machines with different processing power and different bandwidth/latency characteristics, without making explicit use of architectural parameters for tuning performance. In the framework, a network-oblivious algorithm is written for $v(n)$ virtual processors (specification model), where n is the input size and $v(\cdot)$ a suitable function. Then, the performance of the algorithm is analyzed in a simple model (evaluation model) consisting of $p \leq v(n)$ processors and where the impact of the network topology on communication costs is accounted for by a latency parameter σ. Finally, the algorithm is executed on the D-BSP model [de la Torre and Kruskal 1996; Bilardi et al. 2007a] (execution machine model), which describes reasonably well the behavior of a large class of point-to-point networks by capturing their hierarchical structures. A D-BSP consists of $p \leq v(n)$ processors, and its network topology is described by the $\log p$-size vectors $\boldsymbol{g}$ and $\boldsymbol{\ell}$, which account for bandwidth and latency costs within nested clusters, respectively. We have shown that for static network-oblivious algorithms, where the communication requirements depend only on the input size and not on the specific input instance (e.g., algorithms arising in DAG computations), the optimality on the evaluation model for certain ranges of p and σ translates into optimality on the D-BSP model for corresponding ranges of the model's parameters. This result justifies the introduction of the evaluation model that allows for a simple analysis of network-oblivious algorithms while effectively bridging the performance analysis to D-BSP, which more accurately models the communication infrastructure of parallel platforms through a logarithmic number of parameters.

We devised $\Theta(1)$-optimal static network-oblivious algorithms for prominent problems such as matrix multiplication, FFT, and sorting, although in the case of sorting, optimality is achieved only when the available parallelism is polynomially sublinear in the input size. In addition, we devised suboptimal, yet efficient, network-oblivious algorithms for stencil computations, and we explored limitations of the oblivious approach by showing that for the broadcasting problem, optimality in D-BSP can be achieved by a network-oblivious algorithm only for rather limited ranges of the parameters. Similar negative results were also proved in the realm of cache-oblivious algorithms (e.g., see Bilardi and Peserico [2001], Brodal and Fagerberg [2003], and Silvestri [2006, 2008]). Despite these limitations, the pursuit of oblivious algorithms appears worthwhile even when the outcome is a proof that no such algorithm can be $\Theta(1)$-optimal on an ample class of target machines. Indeed, the analysis behind such a result is likely to reveal what kind of adaptivity to the target machine is necessary to obtain optimal performance.

The present work can be naturally extended in several directions, some of which are briefly outlined next. First, it would be useful to further assess the effectiveness of our framework by developing novel efficient network-oblivious algorithms for prominent problems beyond the ones of this article. Some progress in this direction has been done in Chowdhury et al. [2013] and Demmel et al. [2013]. For the problems considered here, particularly sorting and stencil computations, it would be very interesting to investigate the potentiality of the network-oblivious approach at a fuller degree. More generally, it would be interesting to develop lower-bound techniques to limit the level of optimality that network-oblivious algorithms can reach on certain classes of target platforms. Another challenging goal concerns the generalization of the results of Theorems 3.4 and 5.3 to a wider class of algorithms, such as by removing the restriction to static algorithms and/or by weakening the assumptions (wiseness or fullness) required to prove these theorems. It would be also useful to identify other classes of machines for which network-oblivious algorithms can be effective. Another open problem is to augment our framework by incorporating memory constraints in the evaluation model to study the interplay between communication, parallelism, and memory. In this context, it is important to devise suitable schedulers that map network-oblivious algorithms on the evaluation model without violating the memory constraints and to study the inherent trade-offs for fundamental problems. Preliminary results in these directions include space-bounded schedulers for multicores (e.g., Chowdhury et al. [2013] and Simhadri et al. [2014]) and trade-offs for linear algebra problems (e.g., Irony et al. [2004] and Ballard et al. [2011, 2012]). More in general, it would be very interesting to generalize our work to apply to computing scenarios, such as traditional time-shared systems and emerging global computing environments, where the amount of resources devoted to a specific application can itself vary dynamically over time, in the same spirit as Bender et al. [2014] generalized the cache-oblivious framework to environments in which the amount of memory available to an algorithm can fluctuate.

Finally, we observe that some of the network-oblivious algorithms presented in this article share a similar structure with their cache-oblivious counterparts (e.g., see the matrix multiplication and FFT algorithms). It would be interesting to explore whether there is a deeper relation between the two kinds of obliviousness. We conjecture that cache-oblivious algorithms can be obtained by simulating network-oblivious ones using a suitable adaptation of the technique developed in Pietracaprina et al. [2006]. However, the other direction seems far more challenging, as cache-oblivious algorithms do not have to exhibit parallelism necessarily. The ultimate goal would be represented by the integration of cache- and network obliviousness in a unified framework for the development of *machine*-oblivious computations. The results obtained by Blelloch et al. [2010] and Chowdhury et al. [2013] in the context of shared-memory platforms could be a source of inspiration toward this goal.

APPENDIX

A. LIST OF NOTATIONS AND SYMBOLS

The following table summarizes the most important notations and symbols used in the article.

Notation/Symbol	Meaning
n	Input size.
$M(v)$	Computational model that underlies the specification, evaluation, and execution models. It consists of v processing elements.
$M(v(n))$	Specification model with $v(n)$ virtual processors.
$M(p, \sigma)$	Evaluation model with p processors.
D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$	Execution model with p processors.
v	Number of processing elements in the underlying model. The symbol v can thus refer to any (specification, evaluation, or execution) model.
$v(n)$	Number of virtual processors in the specification model.
p	Number of processors in the evaluation or execution models.
σ	Latency parameter in the evaluation model $M(p, \sigma)$.
$\boldsymbol{g} = (g_0, g_1, \ldots, g_{\log p-1})$	Bandwidth parameters of the execution model D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$.
$\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_{\log p-1})$	Latency parameters of the execution model D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$.
$L^i_{\mathcal{A}}(I)$ (respectively, $L^i_{\mathcal{A}}(n)$)	Set of i-supersteps executed by an algorithm $\mathcal{A}$ on input I (respectively, by a static algorithm $\mathcal{A}$ on an input of size n).
$S^i_{\mathcal{A}}(I)$ (respectively, $S^i_{\mathcal{A}}(n)$)	$S^i_{\mathcal{A}}(I) = \lvert L^i_{\mathcal{A}}(I)\rvert$ (respectively, $S^i_{\mathcal{A}}(n) = \lvert L^i_{\mathcal{A}}(n)\rvert$).
$h^s_{\mathcal{A}}(I, p)$ (respectively, $h^s_{\mathcal{A}}(n, p)$)	Maximum number of messages sent or received by a processor of $M(p, \sigma)$ or D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ during superstep s of an algorithm $\mathcal{A}$ on input I (respectively, of a static algorithm $\mathcal{A}$ on an input of size n). It is also called *degree of the superstep*.
$F^i_{\mathcal{A}}(I, p)$ (respectively, $F^i_{\mathcal{A}}(n, p)$)	Cumulative degree of all i-supersteps of an algorithm $\mathcal{A}$ on input I (respectively, of a static algorithm $\mathcal{A}$ on an input of size n).
$H_{\mathcal{A}}(n, p, \sigma)$	Communication complexity of an algorithm $\mathcal{A}$ on $M(p, \sigma)$ with input size n.
$D_{\mathcal{A}}(n, p, \boldsymbol{g}, \boldsymbol{\ell})$	Communication time of an algorithm $\mathcal{A}$ on D-BSP$(p, \boldsymbol{g}, \boldsymbol{\ell})$ with input size n.
$\mathscr{C}$	Class of algorithms that solve a given computational problem.
β-optimality	Characterization of the optimality of algorithms in the evaluation model and in the execution model (see Definitions 2.1 and 2.2, respectively).
(α, p)-wiseness, (γ, p)-fullness	Characterizations of the communication pattern in the evaluation model of a network-oblivious algorithm (see Definitions 3.2 and 5.2, respectively).

ACKNOWLEDGMENTS

The authors would like to thank Vijaya Ramachandran for helpful discussions and the anonymous reviewers for useful comments.

REFERENCES

Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. 1987. A model for hierarchical memory. In *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC'87)*. 305–314.

Alok Aggarwal, Ashok K. Chandra, and Marc Snir. 1987. Hierarchical memory with block transfer. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science (FOCS'87)*. 204–216.

Alok Aggarwal, Ashok K. Chandra, and Marc Snir. 1990. Communication complexity of PRAMs. *Theoretical Computer Science* 71, 1, 3–28.

Alok Aggarwal and Jeffrey S. Vitter. 1988. The input/output complexity of sorting and related problems. *Communications of the ACM* 31, 9, 1116–1127.

Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. 2012. Brief announcement: Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'12)*. 77–79.

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2011. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications* 32, 3, 866–901.

Armin Bäumker, Wolfgang Dittrich, and Friedhelm Meyer auf der Heide. 1998. Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model. *Theoretical Computer Science* 203, 2, 175–203.

Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. 2014. Cache-adaptive algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. 958–971.

Sandeep N. Bhatt, Gianfranco Bilardi, and Geppino Pucci. 2008. Area-time tradeoffs for universal VLSI circuits. *Theoretical Computer Science* 408, 2–3, 143–150.

Gianfranco Bilardi and Enoch Peserico. 2001. A characterization of temporal locality and its portability across memory hierarchies. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP'01)*. 128–139.

Gianfranco Bilardi and Andrea Pietracaprina. 2011. Theoretical models of computation. In *Encyclopedia of Parallel Computing*, D. A. Padua (Ed.). Springer, 1150–1158.

Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci. 1999. 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 (Euro-Par'99)*. 543–551.

Gianfranco Bilardi, Andrea Pietracaprina, and Geppino Pucci. 2007a. Decomposable BSP: A bandwidth-latency model for parallel and hierarchical computation. In *Handbook of Parallel Computing: Models, Algorithms and Applications*, J. Reif and S. Rajasekaran (Eds.). CRC Press, Boca Raton, FL, 277–315.

Gianfranco Bilardi, Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri. 2007b. Network-oblivious algorithms. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*. 1–10.

Gianfranco Bilardi and Franco Preparata. 1995. Horizons of parallel computation. *Journal of Parallel and Distributed Computing* 27, 2, 172–182.

Gianfranco Bilardi and Franco Preparata. 1997. Processor-time tradeoffs under bounded-speed message propagation: Part I, upper bounds. *Theory of Computing Systems* 30, 6, 523–546.

Gianfranco Bilardi and Franco Preparata. 1999. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory of Computing Systems* 32, 5, 531–559.

Gianfranco Bilardi and Geppino Pucci. 2011. Universality in VLSI computation. In *Encyclopedia of Parallel Computing*, D. A. Padua (Ed.). Springer, 2112–2118.

Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Harsha Vardhan Simhadri. 2011. Scheduling irregular parallel computations on hierarchical caches. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11)*. 355–366.

Guy E. Blelloch, Phillip B. Gibbons, and Harsha Vardhan Simhadri. 2010. Low depth cache-oblivious algorithms. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'10)*. 189–199.

Gerth S. Brodal and Rolf Fagerberg. 2003. On the limits of cache-obliviousness. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC'03)*. 307–315.

Rezaul A. Chowdhury and Vijaya Ramachandran. 2008. Cache-efficient dynamic programming algorithms for multicores. In *Proceedings of the 20th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'08)*. 207–216.

Rezaul A. Chowdhury, Vijaya Ramachandran, Francesco Silvestri, and Brandon Blakeley. 2013. Oblivious algorithms for multicores and networks of processors. *Journal of Parallel and Distributed Computing* 73, 7, 911–925.

Richard Cole and Vijaya Ramachandran. 2010. Resource oblivious sorting on multicores. In *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP'10)*. 226–237.

Richard Cole and Vijaya Ramachandran. 2012a. Efficient resource oblivious algorithms for multicores with false sharing. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS'12)*. 201–214.

Richard Cole and Vijaya Ramachandran. 2012b. Revisiting the cache miss analysis of multithreaded algorithms. In *Proceedings of the 10th Latin American Theoretical Informatics Symposium (LATIN'12)*. 172–183.

David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Eunice E. Santos, Klaus E. Schauser, Ramesh Subramonian, and Thorsten von Eicken. 1996. LogP: A practical model of parallel computation. *Communications of the ACM* 39, 11, 78–85.

Pilar de la Torre and Clyde P. Kruskal. 1996. Submachine locality in the bulk synchronous setting. In *Proceedings of the 2nd International Euro-Par Conference on Parallel Processing (Euro-Par'96)*. 352–358.

James Demmel, David Eliahu, Armando Fox, Shoaib Kamil, Ben Lipshitz, Oded Schwartz, and Omer Spillinger. 2013. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*. 261–272.

Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 2012. Cache-oblivious algorithms. *ACM Transactions on Algorithms* 8, 1, Article No. 4.

Matteo Frigo and Volker Strumpen. 2005. Cache oblivious stencil computations. In *Proceedings of the 19th International Conference on Supercomputing (ICS'05)*. 361–366.

Matteo Frigo and Volker Strumpen. 2009. The cache complexity of multithreaded cache oblivious algorithms. *Theory of Computing Systems* 45, 2, 203–233.

Phillip B. Gibbons, Yossi Matias, and Vijaya Ramachandran. 1999. Can a shared-memory model serve as a bridging model for parallel computation? *Theory of Computing Systems* 32, 3, 327–359.

Kieran T. Herley. 2011. Network obliviousness. In *Encyclopedia of Parallel Computing*, D. A. Padua (Ed.). Springer, 1298–1303.

Dror Irony, Sivan Toledo, and Alexandre Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing* 64, 9, 1017–1026.

Joseph JáJá. 1992. *An Introduction to Parallel Algorithms*. Addison Wesley Longman.

Ben H. H. Juurlink and Harry A. G. Wijshoff. 1998. A quantitative comparison of parallel computation models. *ACM Transactions on Computer Systems* 16, 3, 271–318.

Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 938–948.

Leslie Robert Kerr. 1970. *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplication*. Ph.D. Dissertation. Cornell University.

Frank T. Leighton. 1985. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers* 34, 4, 344–354.

Frank T. Leighton. 1992. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann.

Charles E. Leiserson. 1985. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers* 34, 10, 892–901.

Charles E. Leiserson and Bruce M. Maggs. 1988. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica* 3, 1–4, 53–77.

Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. 2012. Space-round tradeoffs for MapReduce computations. In *Proceedings of the 26th ACM International Conference on Supercomputing (ICS'12)*. 235–244.

Andrea Pietracaprina, Geppino Pucci, and Francesco Silvestri. 2006. Cache-oblivious simulation of parallel programs. In *Proceedings of the 8th IEEE IPDPS Workshop on Advances in Parallel and Distributed Computational Models (APDCM'06)*. 1–8.

John E. Savage. 1998. *Models of Computation: Exploring the Power of Computing*. Addison Wesley Longman.

Michele Scquizzato and Francesco Silvestri. 2014. Communication lower bounds for distributed-memory computations. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS'14)*. 627–638.

Francesco Silvestri. 2006. On the limits of cache-oblivious matrix transposition. In *Proceedings of the 2nd Symposium on Trustworthy Global Computing (TGC'06)*. 233–243.

Francesco Silvestri. 2008. On the limits of cache-oblivious rational permutations. *Theoretical Computer Science* 402, 2–3, 221–233.

Harsha Vardhan Simhadri, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Aapo Kyrola. 2014. Experimental analysis of space-bounded schedulers. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'14)*. 30–41.

Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The Pochoir stencil compiler. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11)*. 117–128.

Yuan Tang, Ronghui You, Haibin Kan, Jesmin Jahan Tithi, Pramod Ganapathi, and Rezaul A. Chowdhury. 2015. Cache-oblivious wavefront: Improving parallelism of recursive dynamic programming algorithms without losing cache-efficiency. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15)*. 205–214.

Alexandre Tiskin. 1998. The bulk-synchronous parallel random access machine. *Theoretical Computer Science* 196, 1–2, 109–130.

Leslie G. Valiant. 1990. A bridging model for parallel computation. *Communications of the ACM* 33, 8, 103–111.

Leslie G. Valiant. 2011. A bridging model for multi-core computing. *Journal of Computer and System Sciences* 77, 1, 154–166.

Received April 2014; revised July 2015; accepted August 2015