

# A Lower Bound Technique for Communication in BSP\*

Gianfranco Bilardi<sup>†</sup>

Michele Scquizzato<sup>‡</sup>

Francesco Silvestri<sup>†</sup>

November 28, 2017

## Abstract

Communication is a major factor determining the performance of algorithms on current computing systems; it is therefore valuable to provide tight lower bounds on the communication complexity of computations. This paper presents a lower bound technique for the communication complexity in the bulk-synchronous parallel (BSP) model of a given class of DAG computations. The derived bound is expressed in terms of the switching potential of a DAG, that is, the number of permutations that the DAG can realize when viewed as a switching network. The proposed technique yields tight lower bounds for the fast Fourier transform (FFT), and for any sorting and permutation network. A stronger bound is also derived for the periodic balanced sorting network, by applying this technique to suitable subnetworks. Finally, we demonstrate that the switching potential captures communication requirements even in computational models different from BSP, such as the I/O model and the LPRAM.

## 1 Introduction

A substantial fraction of the time and energy cost of a parallel algorithm is due to the exchange of information between processing and storage elements. As in all endeavors where performance is pursued, it is important to be able to evaluate the distance from optimality of a proposed solution. In this spirit, we consider lower bounds on the amount of communication that is required to solve some computational problems on a distributed-memory parallel system. We model the machine using the standard bulk-synchronous parallel (BSP) model of computation [49], which consists of a collection of  $p$  processors, each equipped with an unbounded private memory and communicating with each other through a communication network.

We focus on a metric, called *BSP communication complexity* and denoted  $H$ , defined as the sum, over all the supersteps of a BSP algorithm, of the maximum number of messages sent or received by any processor (a quantity usually referred to as the *degree* of a superstep). This metric captures a relevant component of the cost of BSP computations. We propose the *switching potential* technique to derive lower bounds on the BSP communication complexity, which is applicable under a number of assumptions. (a) The computation can be modeled in terms of a directed acyclic graph (DAG), whose nodes represent operations (of both input/output and functional type) and whose arcs represent data dependencies. The resulting lower bound holds for all BSP evaluations of the given DAG, which vary depending on the superstep and the processor chosen for the evaluation of an operation, and the way (routing path and schedule of

---

\*This work was supported, in part, by MIUR of Italy under project AMANDA, and by University of Padova under projects STPD08JA32, CPDA121378/12 and CPDA152255/15. A preliminary version of this work [18] was presented at the *18th International European Conference on Parallel and Distributed Computing (Euro-Par 2012)*.

<sup>†</sup>University of Padova. E-mail: bilardi@dei.unipd.it, silvestri@dei.unipd.it.

<sup>‡</sup>KTH Royal Institute of Technology. E-mail: mscq@kth.se.

the message along such a path) in which a value is sent from the processor that computes it to a processor that utilizes it. (b) The internal nodes of the DAG are restricted to have the same number of incoming and outgoing arcs, so that they can be thought of as switches that can establish any one-to-one relation between the incoming arcs and the outgoing arcs. The switching potential is the number of permutations that can be established, by means of arc-disjoint paths, between the arcs incident on the input nodes and those incident on the output nodes. (c) During the BSP computation, each node of the DAG is evaluated exactly once (*no recomputation*). (d) The number of input/output nodes that are mapped to the same BSP processor satisfies a suitable upper bound, essentially ensuring that the computation is distributed over at least two processors.

We illustrate the versatility of the switching potential technique in several ways. We apply it to derive tight lower bounds on the BSP communication complexity of the fast Fourier transform (FFT), and of sorting and permutation networks. We also show how, for some DAGs, the lower bound on communication can be boosted by composing the results provided by the technique for suitable parts of the DAG. Finally, we demonstrate that the switching potential of a DAG captures communication requirements which can lead to lower bounds even in models different from BSP.

## 1.1 A Perspective on Previous Work

The impact of communication on performance has been extensively investigated. Even a concise review of all the relevant work would go far beyond the scope of this paper. Here, we will simply attempt to place our work in the broader context and then review more closely the results more directly comparable to ours.

A first division can be drawn between studies that consider the communication requirements inherent to computational problems and studies that consider the communication requirements of specific algorithms. Examples of results that apply to computational problems are the crossing-sequence lower bounds on time for Turing machines [30], Grigoriev’s flow lower bounds on space-time tradeoffs [43], and information-flow lower bounds on the area-time trade-off in VLSI [53, 46, 16]. One ingredient of some of the lower bounds in our paper is based on the information flow of the cyclic shift problem, originally studied in [51].

Our switching potential approach targets lower bounds for classes of implementations of specific algorithms, typically modeled by *computation DAGs*, where nodes represent input and functional operations, and arcs represent data dependencies. A significant distinction lies in whether the rules defining the class of implementations allow for *recomputation*, that is, for the ability to evaluate a given functional node of the DAG multiple times. A number of efforts have focused on data movement between levels of the memory hierarchy, e.g., [31, 2, 43, 13, 42] (with recomputation) and [17, 42, 5, 6] (without recomputation). Other papers have investigated data movement between processing elements in distributed computing, e.g., [40, 1, 39] (with recomputation), and [28, 17, 33, 5, 6, 45] (without recomputation). Re-execution of operations is of interest because it is known that, in some models of computation, it can be exploited in order to asymptotically improve performance. For example, recomputation is known to have the potential to reduce the space requirements of computations in the context of classical pebbling games (see, e.g., [44]), to enhance the performance of simulations among networks (see [36] and references therein), to enable area-universal VLSI computations [10] with constant slowdown, and to reduce the number of write operations between levels of the memory hierarchy [19]. However, as in most of the present paper, recomputation is often ruled out, as a simplifying assumption that still affords the development of insights on the problem, when the general case proves hard to tackle.

The metrics capturing communication requirements are typically quite sensitive to the underlying model of computation; even within the same model, several metrics can be meaningful. In this paper, we will focus on models where the computation is carried out by a set of processors with local storage, interconnected by a communication medium. Examples of such models are bounded degree networks [38], BSP [49], LogP [24]. A pioneering paper in this area [40] introduced two important metrics and analyzed them for the diamond DAG: the total number  $c$  of messages exchanged among the processors, called *communication*, and the maximum number  $d$  of messages exchanged as part of the evaluation of some path of the DAG, called *communication delay*. Both metrics can be trivially minimized to zero by assigning the entire DAG to just one processor. However, interesting tradeoffs arise between the two metrics and the number of steps  $t$  of a parallel schedule. For example, for the diamond DAG with  $n$  inputs,  $c = \Omega(n^3/t)$  and  $d = \Omega(n^2/t)$ , allowing for recomputation.

Another interesting metric is the maximum number  $\hat{H}_{rc}$  of messages received by any processor. Clearly,  $\hat{H}_{rc} \geq c/p$ , since each message is received by some processor. Next, we outline a lower bound technique for  $\hat{H}_{rc}$ , based on the graph-theoretic notion of dominator. If  $W$  and  $U$  are sets of nodes of a given DAG,  $W$  is called a *dominator* of  $U$  if every path from an input node to a node of  $U$  includes a node of  $W$ . Let  $D(k)$  be the maximum size of a set  $U$  of nodes that has a dominator  $W$  of size  $k$ . In a seminal paper [31], formulating a framework for the study I/O complexity in hierarchical memories, the number  $Q(S)$  of data transfers between a “cache” of size  $S$  and “main memory” is shown to satisfy the lower bound  $Q(S) \geq \lfloor \nu S / D(2S) \rfloor$ , where  $\nu$  is the number of (input and functional) nodes. The dominator technique can be adapted to establish that, if the DAG is evaluated by  $p$  processors, each of which initially stores at most  $q$  inputs, then at least one processor receives  $\hat{H}_{rc} \geq D^{-1}(\nu/p) - q$  messages. The argument goes as follows: (a) a processor  $P$  can receive, as part of the input or from other processors, the value of at most  $(q + \hat{H}_{rc})$  nodes of the DAG; (b) these nodes must dominate all the nodes whose value is computed by  $P$ ; (c) by definition of  $D$ , at most  $D(q + \hat{H}_{rc})$  DAG nodes are dominated by any given set of  $(q + \hat{H}_{rc})$  nodes; (d) at least one processor must compute no fewer than  $\nu/p$  of the  $\nu$  nodes in the DAG. Combining these premises one obtains that  $D(q + \hat{H}_{rc}) \geq \nu/p$ , whence the stated bound on  $\hat{H}_{rc}$ . Variants of the outlined argument have been used, for matrix multiplication, in [1, 33, 45]. Dominator-based lower bounds do allow for recomputation.

By definition, the BSP communication complexity  $H$  satisfies  $H \geq \hat{H}_{rc}$ . In particular,  $H$  can be larger than  $\hat{H}_{rc}$ , if different processors receive messages in different supersteps. This difference can play a role, as in the case of the BSP computation of an  $n$ -input radix-two FFT DAG [23], with  $\nu = n \log(2n)$  nodes,<sup>1</sup> a case which has motivated our work. An FFT implementation on BSP is known [49] for which

$$H = O(\hat{H}_{rc}) = O\left(\frac{n \log n}{p \log(n/p)}\right), \quad (1)$$

for  $1 \leq p \leq n/2$ , with each processor initially holding  $n/p$  (consecutive) inputs. Valiant’s analysis is based on the well known property that, if  $s$  divides  $n$  (a power of 2), then the  $n$ -input FFT DAG can be viewed as a sequence of  $\lceil \log n / \log s \rceil$  stages, each (i) consisting of  $n/s$  disjoint  $s$ -input FFTs (possibly incomplete in the last stage) and (ii) being connected to the next one by a permutation. Letting  $s = n/p$  and assigning the  $p$  ( $n/p$ )-input FFTs within a stage to different processors, the resulting BSP algorithm executes  $K = \lceil \log n / \log(n/p) \rceil$  supersteps, each of degree  $O(n/p)$ , yielding bound (1). Making use of the result [31] that in the FFT DAG  $k$  nodes dominate at most  $D(k) \leq 2k \log k$  nodes,<sup>2</sup> the dominator technique yields

<sup>1</sup>In this paper  $\log x$  denotes the logarithm to the base two.

<sup>2</sup>In the Appendix we show that this bound can be improved to  $D(k) \leq k \log 2k$ .

the lower bound

$$H \geq \hat{H}_{\text{rc}} = \Omega\left(\frac{n \log n}{p \log((n/p) \log n)}\right), \quad (2)$$

assuming that at most  $q = \beta(n \log n)/(p \log((n/p) \log n))$  inputs are initially available to any processor, for a suitably small constant  $\beta$ . We observe that the lower bound (2) does not match the upper bound (1), for either  $H$  or  $\hat{H}_{\text{rc}}$ , when  $p = n/2^{o(\log \log n)}$ .

An alternate dominator-based lower bound can be obtained when the local memory of each processor has at most  $m$  locations [4]. During an interval in which it receives  $m$  messages, a processor can evaluate the at most  $D(2m)$  nodes dominated by the (at most)  $m$  nodes whose value is received together with the (at most)  $m$  nodes whose value is locally stored at the beginning of the interval. Then, if  $\nu$  nodes are evaluated by  $p$  processors,  $\hat{H}_{\text{rc}} = \Omega(m \lfloor \nu / (pD(2m)) \rfloor)$ . For the FFT,  $\nu = n \log(2n)$  and  $D(2m) \leq 2m \log(4m)$ , hence  $\hat{H}_{\text{rc}} = \Omega(m \lfloor (n \log n) / (p2m \log(4m)) \rfloor)$ . If we let  $m^*$  denote the value of  $m$  for which the argument of the floor equals 1, we can see that, for  $m \leq m^*$ ,  $\hat{H}_{\text{rc}} = \Omega((n \log n)/(p \log m))$ . In particular, for  $m = \Theta(n/p)$ , the lower bound matches the upper bound (1). For  $m > m^*$ , the bound vanishes. At first, it may be puzzling that, for large enough  $m$ , the memory based bound does not reproduce bound (2); the reason is that, unlike the latter, the former bound does not depend upon the assumption that, initially, only a limited amount of input is available to each processor.

A model with some similarities to BSP is the LPRAM of Aggarwal et al. [1], where  $p$  processors with unbounded local storage are synchronized by a global clock and communicate via a shared memory, rather than directly through a network. The communication metric is the number of steps (cycles)  $T_c$  in which at least one processor reads from or writes to the shared memory. A straightforward adaptation of a well-known decomposition strategy for the FFT achieves for  $T_c$  an upper bound of the same form as (1). A lower bound of the same form is also established. In addition to being developed for a different model, the argument follows a route different than ours: a lower bound of the same form is first established for sorting (assuming no input element is ever kept by two processors at the same time), then claimed (by analogy) for permutation networks, and finally adapted to the FFT network, by exploiting the property established in [52] that the cascade of three FFT networks has the topology of a full permutation network.

Finally, we mention that, motivated by the investigation of area-time trade-offs in VLSI, a number of lower bounds have been established on the information flow of the (multidimensional) discrete Fourier transform (DFT) computed either exactly, on finite rings, or approximately, on the complex field (see, e.g., [12] and references therein). These results apply to any algorithm, rather than just to the radix-two specialization of the FFT considered in the present paper and related work. When adapted to computing the DFT on BSP, with information measured in words capable of encoding a ring element, the known  $\Omega(n)$  word lower bound on the information flow through the bisection of the system does imply that  $H \geq \hat{H}_{\text{rc}} = \Omega(n/p)$ , assuming that no processor inputs (or outputs) more than  $n/2$  values. This bound is of the same order as the dominator-based bound (2) when  $\log((n/p) \log n) = \Omega(\log n)$  and is weaker otherwise, but it does hold under less stringent constraints on the I/O protocol.

In summary, with respect to the outlined state of the art, our objective is to develop lower bound techniques for the communication complexity in BSP,  $H$ , capable to close the gap between the dominator lower bound (2) and the best known upper bound (1) for the FFT, as well as to weaken the assumptions on the input protocol under which the lower bound is established.

## 1.2 Overview of Results

The main contribution of this paper is the *switching potential technique*, to obtain communication lower bounds for DAG computations in the BSP model. The proposed technique applies to DAGs, named switching DAGs, with  $n$  input nodes where all nodes, except for inputs and outputs, have out-degree equal to the in-degree. Such a graph  $G = (V, E)$  can be viewed as a switching network [44] for which a switching size  $N$  and a switching potential  $\gamma$  are defined. The *switching size*  $N$  is the sum of the out-degrees of the input nodes or, equivalently, the sum of the in-degree of the output nodes. The *switching potential*  $\gamma$  is the number of different ways in which  $N$  tokens initially placed on the  $N$  outgoing arcs of the input nodes can be brought on the  $N$  incoming arcs of the output nodes, by moving them along arc-disjoint paths. Intuitively, the switching potential is a measure of the permuting ability of the graph. Its impact on the BSP communication complexity is quantified in the following theorem.

**Theorem 1.** *Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation a switching DAG  $G = (V, E)$  on a BSP with  $p$  processors. Let  $N$ ,  $\gamma$ , and  $\Delta$  be respectively the switching size, the switching potential, and the maximum out-degree of any node of  $G$ . If the sum of the in-degree of the output nodes evaluated by every processor is at most  $U$ , then the BSP communication complexity of algorithm  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} \geq \begin{cases} \frac{\log(\gamma/(U!)^{N/U})}{\Delta p \log(N/p)} & \text{if } p \leq N/e, \\ \frac{e \log(\gamma/(U!)^{N/U})}{\Delta N \log e} & \text{otherwise.} \end{cases} \quad (3)$$

where, by definition,  $\gamma \leq N!$  and  $N/p \leq U \leq N$ .

For the proof, we introduce the *envelope game*, where a set of envelopes, initially positioned on the input nodes, are moved to the output nodes according to some given rules. The evaluation of a DAG in the BSP model is viewed as a run of this game; a lower bound is derived for any BSP algorithm playing the game.

At the heart of the switching potential technique lies a counting argument in combination with an indistinguishability argument, somewhat similar to the approach used by Aggarwal and Vitter [2] to study the I/O complexity of computations, and later applied to study the complexity of communications in the LPRAM [1]. Our technique has the advantage that it can be directly applied to any specific (switching) DAG, while the former approaches require that a suitable combination of copies of the DAG under consideration yield a full permutation network.

Even when  $\gamma$  is large, bound (3) may become weak due to high values of  $U$ . Intuitively, when  $U$  is high, many permutations can be realized by redistributing the data within the processors, without much interprocessor communication. On the other hand, if the different permutations realizable by a given DAG map somewhat uniformly the inputs to the outputs, then the communication complexity can be considerable, even for rather small values of  $\gamma$ . This phenomenon has been investigated in [51] for classes of permutations that form a transitive group. A special case is the class of the cyclic shifts, which are permutations where all input values are shifted by a given amount (we refer to Section 5 for a formal definition). We establish the following result:

**Theorem 2.** *Let  $G = (V, E)$  be a DAG with  $n$  input nodes and  $n$  output nodes, capable of realizing all the  $n$  cyclic shifts, with respect to some fixed correspondence between inputs and outputs. Let  $\mathcal{A}$  be an algorithm that evaluates  $G$  (possibly, with recomputation) on a BSP with  $p \geq 2$  processors, such that initially each input is available to exactly one processor. Then,*

if some processor initially stores (exactly)  $q$  inputs, or some processor evaluates (exactly)  $q$  outputs, the BSP communication complexity of algorithm  $\mathcal{A}$  satisfies

$$H_{\mathcal{A}} \geq \frac{\min\{q, n - q\}}{2}.$$

Some DAGs of interest happen to both exhibit a high switching potential  $\gamma$  and realize all cyclic shifts. For such DAGs, a combination of Theorems 1 and 2 yields communication lower bounds under very mild assumptions on the input/output protocol. An example is the  $n$ -input FFT, for which  $\gamma = 2^{n(\log n - 1)}$  and the lower bound takes the form, when  $p \leq 2n/e$ ,

$$H_{\text{FFT}} > \frac{n \log(n/2)}{8p \log(2n/p)}, \quad (4)$$

as long as no processor evaluates more than  $n/2$  output nodes, and there is no recomputation. Lower bound (4) is the first lower bound on the BSP communication complexity of the FFT that asymptotically matches upper bound (1) for any number of processors  $p \leq 2n/e$ . The technique based on the capability of realizing all cyclic shifts also enables the extension of the dominator-based lower bound (2) to milder assumptions on the input/output protocol. The dominator-based bound is not asymptotically tight for  $p = n/2^{o(\log \log n)}$ , but it remains of interest when recomputation is allowed.

We illustrate the versatility of the switching potential technique by applying it to computations different from the FFT. Sorting and permutation networks naturally exhibit a high switching potential. The corresponding BSP communication complexity lower bound is asymptotically similar to bound (4), and to the best of our knowledge is the first known result for these computations. An asymptotically equivalent lower bound was previously derived for BSP sorting in [28], but only for algorithms with supersteps of degree  $\Theta(n/p)$  and input evenly distributed among the processors. We also show how the switching potential analysis can some time yield higher lower bounds if separately applied to suitable parts of the DAG; in particular, we prove for the BSP communication complexity of the periodic balanced sorting network [26] a bound higher than the one derived for all sorting networks.

The switching potential technique can be used, with some minor changes, to derive lower bounds in other computational models besides the BSP. Specifically, we apply the technique to a parallel variant of the I/O model which includes, as special cases, both the I/O model and the LPRAM model.

In addition to the well-known general motivations for lower bound techniques, we stress that striving for tight bounds for the whole range of model's parameters has special interest in the study of so-called *oblivious* algorithms, which are specified without reference to such parameters, but are designed with the goal of achieving (near) optimality for wide ranges of the parameters. Notable examples are cache-oblivious algorithms [27], multicore-oblivious algorithms [20], resource-oblivious algorithms [21, 22] and, closer to the scenario of this paper, network-oblivious algorithms [15], where algorithms are designed and analyzed on a BSP-like model. In fact, many BSP algorithms are only defined or analyzed for a number of processors  $p$  that is sufficiently small with respect to the input size  $n$ . For the analysis of the FFT DAG, it is often assumed  $p \leq \sqrt{n}$ , where the complexity is  $\Theta(n/p)$ . Our results allow for the removal of such restrictions.

A preliminary version of this paper appeared in [18]. The current version contains an expanded discussion of previous work, and provides full proofs of all claims, a significantly simpler and slightly improved analysis of the switching potential technique, applications of this technique to more case studies, as well as an adaptation of it to a different model of computation.

### 1.3 Paper Organization

Section 2 introduces the BSP model and the concept of switching DAG. Section 3 formulates the envelope game, a convenient framework for studying the communication occurring when evaluating a switching DAG. Section 4 develops the switching potential technique culminating with the proof of Theorem 1, which provides a lower bound on the BSP communication complexity of a switching DAG, in terms of its switching potential. Section 5 establishes Theorem 2, which provides a lower bound on the BSP communication complexity of a DAG that can realize all cyclic shifts. These two results are then applied, in Section 6, to the FFT DAG and to sorting and permutation networks. Section 7 extends the switching potential technique to computational models different from BSP. Finally, in Section 8, we draw some conclusions and discuss directions for further work.

## 2 Models of Computation

This section introduces the BSP model of parallel computation and the class of computation DAGs for which our lower bound technique applies.

### 2.1 The BSP Model

The *bulk-synchronous parallel* (BSP) model of computation was introduced by Valiant [49] as a “bridging model” for general-purpose parallel computing, providing an abstraction of both parallel hardware and software. It has been widely studied (see, e.g., [48] and references therein) together with a number of variants (such as D-BSP [25, 14], BSP\* [8], E-BSP [34], and BSPRAM [47]) that aim at capturing data and communication locality by basing the cost function on different communication metrics.

The architectural component of the model consists of  $p$  processors  $P_1, P_2, \dots, P_p$ , each equipped with an unbounded local memory, interconnected by a communication medium. The execution of a BSP algorithm consists of a sequence of phases, called *supersteps*: in one superstep, each processor can perform operations on data residing in its local memory, send/receive messages (each occupying a constant number of words) to/from other processing elements and, at the end, execute a global synchronization instruction. Messages sent during a superstep become visible to the receiver at the beginning of the next superstep.

The running time of the  $j$ -th superstep is expressed in terms of two parameters,  $g$  and  $\ell$ , as  $T_j = w_j + h_j g + \ell$ , where  $w_j$  is the maximum number of local operations performed by any processor in the  $j$ -th superstep, and  $h_j$  (usually called the *degree* of superstep  $j$ ) is the maximum number of messages sent or received by any processor in the  $j$ -th superstep. If the time unit is chosen to be the duration of a local operation, then parameter  $g$  is defined to be such that the communication medium can deliver the messages of a superstep of degree  $h$  in  $hg$  units of time, so that  $1/g$  can be viewed as measuring the available bandwidth of the communication medium, whereas parameter  $\ell$  is an upper bound on the time required for global barrier synchronization. The *running time*  $T_{\mathcal{A}}$  of a BSP algorithm  $\mathcal{A}$  is the sum of the times of its supersteps and can be expressed as  $W_{\mathcal{A}} + H_{\mathcal{A}}g + S_{\mathcal{A}}\ell$ , where  $S_{\mathcal{A}}$  is the number of supersteps,  $W_{\mathcal{A}} = \sum_{j=1}^{S_{\mathcal{A}}} w_j$  is the *local computation complexity*, and  $H_{\mathcal{A}} = \sum_{j=1}^{S_{\mathcal{A}}} h_j$  is the *BSP communication complexity*. In this paper, we study the latter metric, which often represents the dominant component of the running time.

## 2.2 Switching DAGs

A *computation DAG*  $G = (V, E)$  is a directed acyclic graph where nodes represent input and functional operations and arcs represent data dependencies. More specifically, an arc  $(u, v) \in E$  indicates that the value produced by the operation associated with  $u$  is one of the operands of the operation associated with  $v$ , and we say that  $u$  is a *predecessor* of  $v$  and  $v$  a *successor* of  $u$ . The number of predecessors of a node  $v$  is called its *in-degree* and denoted  $\delta_{\text{in}}(v)$ , while the number of its successors is called its *out-degree* and denoted  $\delta_{\text{out}}(v)$ . A node  $v$  is called an *input* if  $\delta_{\text{in}}(v) = 0$  and an *output* if  $\delta_{\text{out}}(v) = 0$ . We denote by  $V_{\text{in}}$  and  $V_{\text{out}}$  the set of input and output nodes, respectively. The remaining nodes are said to be *internal* and their set is denoted by  $V_{\text{int}}$ .

Of special interest for our developments are situations where the computation executed by a given algorithm on a given input can be viewed as embedding an *evaluation* of a given computation DAG  $G$ . Informally, this means that, during that execution, all the nodes of  $G$  are evaluated, respecting the dependencies specified by the arcs. A bit more formally, we say that the execution of a BSP algorithm on a specific input  $\mathbf{x}$  *evaluates* a given computation DAG if, to any  $v \in V$ , is associated a set  $\mathcal{S}(v)$  of processor-time pairs such that: (a) if  $(t, P) \in \mathcal{S}(v)$ , then, at time  $t$ , processor  $P$  evaluates node  $v$ , by either an input or a functional operation; (b) if  $(t, P) \in \mathcal{S}(v)$  and  $(u, v) \in E$ , then there is a  $(t', P') \in \mathcal{S}(u)$  such that the result of the evaluation of  $u$  by  $P'$  at time  $t'$  is effectively used as an operand by  $P$  at time  $t$ . Between time  $t'$  and time  $t$ , the value of  $u$  in question may be moved to different processors and memory locations: the sequence of instructions implementing such moves will be denoted as  $\mathcal{S}(t', P', t, P)$ . Taken together, the sets  $\mathcal{S}$  define the processing and communication *schedule* of the evaluation of  $G$ . We say that the evaluation is *without recomputation* if each node of  $G$  is evaluated exactly once, that is, if  $\mathcal{S}(v)$  is a singleton for every  $v$ .

A few observations may help provide some perspective on the above notions of evaluation and schedule. A little reflection will show that an algorithm execution that embeds an evaluation of  $G$  (possibly with recomputation) does not necessarily embed an evaluation of  $G$  without recomputation, whence forbidding recomputation effectively restricts lower bounds results. We remark that an execution of an algorithm that embeds an evaluation of  $G$  may well contain additional operations not modeled by  $G$  (for example the additional operations may be instrumental to constructing the DAG from the input, or the input size). In general, the execution of the same algorithm on different inputs may embed the evaluation of different DAGs or of different schedules of the same DAG. However, there are interesting algorithms that evaluate the same DAG, with the same schedule, for all inputs of the same size  $n$ . Notable examples include the FFT, network algorithms for sorting and permutations, standard matrix multiplication in a semiring, and Strassen's matrix multiplication on a ring.

A number of graph-theoretic properties of the DAG can be related to processing, storage, and communication requirements of the underlying algorithm, as well as to its amount of parallelism. One such property is the switching potential, which we introduce and relate to the communication complexity of a BSP algorithm that evaluates a DAG of the type defined next.

**Definition 1.** A switching DAG  $G = (V, E)$  is a computation DAG where, for any internal node  $v \in V_{\text{int}}$ , we have  $\delta_{\text{out}}(v) = \delta_{\text{in}}(v)$ . We refer to  $n = |V_{\text{in}}|$  as to the input size of  $G$ , and introduce the switching size  $N$  of  $G$  defined as

$$N = \sum_{v \in V_{\text{in}}} \delta_{\text{out}}(v) = \sum_{v \in V_{\text{out}}} \delta_{\text{in}}(v),$$

where the equality between the two summations is easily established.



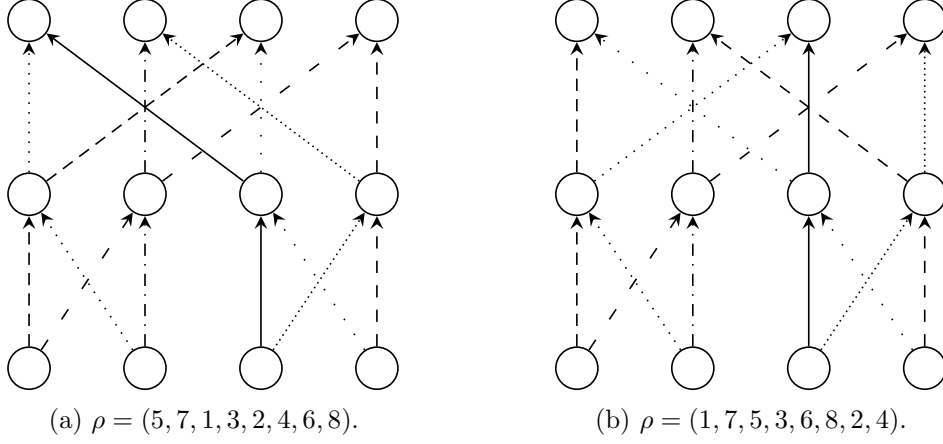


Figure 1: Examples of two different sets of arc-disjoint paths in the FFT DAG on  $n = 4$  inputs and switching size  $N = 8$ . Arcs in  $E_{\text{in}}$  and in  $E_{\text{out}}$  are numbered from 1 to  $N = 8$ , left to right.

Consider the set of arcs  $E_{\text{in}} = E \cap (V_{\text{in}} \times V)$ , outgoing from the input nodes, and the set of arcs  $E_{\text{out}} = E \cap (V \times V_{\text{out}})$ , incoming into the output nodes. Let us now number both the arcs in  $E_{\text{in}}$  and those in  $E_{\text{out}}$  from 1 to  $N$ , in some arbitrarily chosen order. Then, to any partition of  $E$  into a set of arc-disjoint paths there corresponds a permutation  $\rho = (\rho(1), \rho(2), \dots, \rho(N))$  of  $(1, 2, \dots, N)$ , where  $\rho(j)$  is the (number of the) last arc (in  $E_{\text{out}}$ ) of the unique path whose first arc (in  $E_{\text{in}}$ ) is numbered  $j$ . See Figure 1.

**Definition 2.** *The switching potential  $\gamma$  of a switching DAG  $G = (V, E)$  is the number of permutations  $\rho$  corresponding to (one or more) partitions of  $E$  into arc-disjoint paths.*

Intuitively, if we think of each internal node  $v$  of the DAG as a switch that can be configured to connect its incoming arcs to its outgoing arcs in any one-to-one correspondence, then switch configurations uniquely correspond to partitions of  $E$  into arc-disjoint paths. Thus,  $N$  items initially positioned on the input nodes (specifically,  $\delta_{\text{out}}(u)$  items on input  $u$ ) can travel without conflicts and reach the output nodes. Indeed, in the special case where  $\delta_{\text{out}}(u) = 1$  for all input nodes and  $\delta_{\text{in}}(v) = 1$  for all output nodes, one has  $N = n = |V_{\text{in}}| = |V_{\text{out}}|$  and the switching DAG becomes a switching network in the traditional sense [44]. When all permutations can be realized, that is  $\gamma = n!$ , then the switching network is said to be a *permutation* (or, *rearrangeable*) *network* [44]. It is a simple exercise to establish that, for any switching DAG,

$$\gamma \leq \prod_{v \in V \setminus V_{\text{out}}} \delta_{\text{out}}(v) = \prod_{v \in V \setminus V_{\text{in}}} \delta_{\text{in}}(v),$$

where each products equals the number of distinct partitions into arc-disjoint paths. The inequality arises when distinct partitions lead to the same permutation.

### 3 The Envelope Game

In this section we introduce the *envelope game*, to be played on a switching DAG. The goal of the game consists in moving to the output nodes some envelopes, initially placed on the input nodes, according to some rules. Informally, the rules force the envelopes to travel along arc-disjoint paths in a way that all the envelopes that go through a given node must be simultaneously at that node, at some time. The envelope game is meant to provide an abstraction of the evaluation of a DAG without recomputation, which will prove useful in the study of BSP communication

complexity. The spirit is similar to the one that motivated the introduction of the pebble game to study space requirements of computations [41, 32].

**Definition 3.** *The envelope game on a switching DAG  $G$  is defined by the following rules, which characterize the (legal) runs of the game.*

1. *A set of  $N$  distinguishable envelopes is given, with exactly  $\delta_{\text{out}}(u)$  envelopes initially placed on each input node  $u \in V_{\text{in}}$  (hence,  $N$  is the switching size of  $G$ ).*
2. *The set of envelopes remains invariant during the game; at any stage each envelope is at exactly one node of  $G$ .*
3. *One move consists in moving one envelope from a node  $u$  to a node  $v$  along an arc  $(u, v) \in E$ .*
4. *An arc  $(u, v)$  can be used only in one move.*
5. *An envelope can be moved from a node  $u$  only after  $\delta_{\text{in}}(u)$  envelopes (each arriving from a different incoming arc of  $u$ ) have been placed on  $u$ .*
6. *The (run of the) game is completed when all envelopes have reached an output node (i.e., when exactly  $\delta_{\text{in}}(w)$  envelopes are placed on each output node  $w \in V_{\text{out}}$ ).*

According to Rules 1, 2, 3, and 6, during one run, each envelope traverses a path from  $V_{\text{in}}$  to  $V_{\text{out}}$ . Due to Rule 4, paths traversed by different envelopes are arc-disjoint. Due to the property  $\delta_{\text{out}}(v) = \delta_{\text{in}}(v)$  of internal nodes of a switching DAG and to Rule 1 requiring that the number of envelopes equals the switching size  $N$ , the paths traversed by the envelopes yield a partition of the arc set  $E$ . Therefore, each run of the game uniquely identifies a permutation  $\rho$  contributing to the switching potential, according to Definition 2. Finally, we observe that Rule 5 requires that, for each node  $u$ , there is a time when all the envelopes going through  $u$  are on  $u$ .

We observe that, in spite of some similarities, the envelope game differs from the pebbling game in various significant ways. In particular, while the number of pebbles on the DAG can change during the game and the goal is to minimize its (maximum) value, the number of envelopes is constant throughout the game and the goal is to count the mappings between the starting and the ending arcs of the envelope paths by playing the game in all possible ways. Furthermore, at any time, the number of pebbles on a DAG node is either 0 or 1, whereas the number of envelopes on any given node will typically go from 0 to the degree (by unit increments) and then back to 0 (by unit decrements).

Intuitively, a run of the envelope game can be easily augmented into an evaluation of the DAG. We just need to imagine that each envelope carries a (rewritable) card where, when a node  $u$  is computed, its result is written on the card. Since the envelopes leaving from a node  $u$  are distinct even though in the process of DAG evaluation they would carry the same value, the communication of envelopes may result in an overcounting of messages, by a factor at most  $\Delta = \max_{v \in V} \delta_{\text{out}}(v)$ . In fact, there are at most  $\Delta$  envelopes with the same card value moving out from a node. For many DAGs of interest,  $\Delta$  is a small constant (e.g., for the FFT,  $\Delta = 2$ ), thus the overcounting is well bounded. At the same time, the distinguishability of the envelopes simplifies the analysis of the communication requirements of DAG evaluation. The next lemma establishes that communication lower bounds can be transferred from the envelope game to DAG evaluation, by showing how the former can be obtained from the latter.

We say that a BSP algorithm *plays* the envelope game on a switching DAG  $G$  if it satisfies the following conditions: (a) each node  $v \in V$  is assigned to a processor  $P(v)$ ; (b) the envelopes

placed on input node  $u$  are initially in  $P(u)$ ; (c) whenever  $(u, v) \in E$  and  $P(u) \neq P(v)$ , the envelope moved along  $(u, v)$  is sent from  $P(u)$  to  $P(v)$  (possibly via intermediate processors), after  $P(u)$  has received all the envelopes destined to  $u$ .

**Lemma 1.** *Let  $\mathcal{A}$  be any algorithm that evaluates, without recomputation, a switching DAG  $G = (V, E)$  with  $\Delta = \max_{v \in V} \delta_{\text{out}}(v)$ , on a BSP with  $p$  processors, and let  $H_{\mathcal{A}}$  be its BSP communication complexity. Then, there exists an algorithm  $\mathcal{B}$  that plays the envelope game on  $G$  with BSP communication complexity*

$$H_{\mathcal{B}} \leq \Delta H_{\mathcal{A}}.$$

*Proof.* In algorithm  $\mathcal{B}$ , a node  $v \in V$  is assigned to the processor  $P(v)$  where  $v$  is evaluated by  $\mathcal{A}$ , which is unique due to the hypothesis of no recomputation. (1) Initially, for every  $u \in V_{\text{in}}$ ,  $\delta_{\text{out}}(u)$  envelopes are placed on  $u$  (as required by Rule 1) and each envelope is univocally assigned to an outgoing arc of the respective input node. (2) The computation in each internal node  $u$  is replaced with a switch that sequentially forwards the  $\delta_{\text{in}}(u)$  input envelopes to the  $\delta_{\text{out}}(u)$  output arcs according to some permutation. (3) For each arc  $(u, v)$  where  $P(v)$  differs from  $P(u)$ , a message is sent from  $P(u)$  to  $P(v)$  (possibly via intermediate processors) carrying the envelope moved along  $(u, v)$ .

We now show that algorithm  $\mathcal{B}$  plays the envelope game. By construction, the  $N$  envelopes are set on the input nodes as required by Rule 1. Rule 2 is satisfied since recomputation is disallowed (i.e., no new envelope is added) and, in each internal node, all input envelopes are forwarded to the outgoing arcs (i.e., no envelope is deleted). Rules 3, 4, and 5 are complied with since a node  $u$  is computed in  $\mathcal{A}$  only when all the  $\delta_{\text{in}}(u)$  inputs are ready (i.e., every predecessor of  $u$  has sent an envelope to  $u$  in  $\mathcal{B}$ ) and the output values are propagated only to the  $\delta_{\text{out}}(u)$  successors of  $u$ . Rule 6 is also obeyed since all output nodes in  $G$  are computed by  $\mathcal{A}$ .

We now show that the BSP communication complexity of  $\mathcal{B}$  is at most  $\Delta$  times the BSP communication complexity of  $\mathcal{A}$ . The first two modifications do not increase the communication (an envelope can be locally constructed by a processor and no communication is required for evaluating a switch). The third modification can increase the BSP communication complexity as analyzed next. Consider the case where a node  $u$  is processed by  $\mathcal{A}$  on a processor, say  $P_0$ , while  $\ell$  of its successors are processed on a different processor, say  $P_1$ , where  $1 \leq \ell \leq \Delta$ . Then, one message from  $P_0$  to  $P_1$  is necessary and sufficient to send the output value of  $u$  to the  $\ell$  successors in  $P_1$  since the output value of a node is the same for each successor. In contrast, since distinct envelopes are sent by  $u$  to its successors,  $\ell$  messages must be sent by  $\mathcal{B}$  in order to forward the  $\ell$  envelopes from node  $u$  in  $P_0$  to the  $\ell$  successors in  $P_1$ . Therefore, in the worst case, to each message in  $\mathcal{A}$  there correspond  $\Delta$  messages in  $\mathcal{B}$ .  $\square$

Typically, a BSP algorithm  $\mathcal{A}$  that evaluates a DAG will implement the same schedule of operations and messages, for any input of the DAG size  $n$ ; in such a case, the corresponding algorithm  $\mathcal{B}$  that plays the envelope game will be the same for all inputs as well. However, Lemma 1 applies even if DAG  $G$  is evaluated by  $\mathcal{A}$  in different ways for different inputs  $\mathbf{x}$ , in which case the lemma inequality can be more explicitly written as  $H_{\mathcal{B}(\mathbf{x})} \leq \Delta H_{\mathcal{A}(\mathbf{x})}$ .

## 4 The Switching Potential Technique

This section develops the proof of Theorem 1, a lower bound on the communication complexity of any BSP algorithm  $\mathcal{A}$  that evaluates, without recomputation, a switching DAG  $G$  with switching potential  $\gamma$ . Technically, the lower bound individually applies to any execution of the

algorithm that embeds an evaluation of  $G$ . We recall that  $n$  denotes the input size and  $N$  the switching size of  $G$ . Some observations may be useful to build up intuition before entering the technical development, which focuses on the envelope game.

One important point is that the communication requirements captured by Theorem 1 do not simply arise from the data movement implied by the  $\gamma$  permutations that contribute to the switching potential, but rather by the constraint that all those permutations must be realizable under the same schedule, in the sense defined in Section 2.2. In fact, from one run of the envelope game one can always obtain all other runs, by simply changing the envelope permutation locally applied at each node. Then, at any given time during the algorithm, the set of locations that contain envelopes is the same for all runs, although how the envelopes are distributed across those locations will differ from run to run. Similarly, at any given superstep, the set of (source, destination) pairs of the messages sent in that superstep is the same for all runs, although the envelope carried by a given message will generally vary from run to run.

To appreciate the implications of a fixed schedule, let us consider the problem of permuting  $N$  records among  $p$  BSP processors, under an input/output protocol where at most  $q_{\text{in}}$  records are initially held by any processor and at most  $q_{\text{out}}$  records are destined to any processor. A simple BSP algorithm, executing just one superstep where each record is directly sent from the source processor to the destination processor, accomplishes the task with minimum BSP communication complexity  $\max\{q_{\text{in}}, q_{\text{out}}\}$ , assuming that a record fits in one message. In particular, if the I/O protocol is balanced, i.e.,  $q_{\text{in}} = q_{\text{out}} = N/p$ , then each permutation can be accomplished with communication complexity  $N/p$ . However, under this algorithm, each permutation will result in a different set of (source, destination) pairs for the  $N$  messages that carry the records, hence in a different communication schedule. On the other hand, a BSP algorithm that evaluates the DAG corresponding to a sorting or permutation network (see Section 6.2 for more details) has a fixed communication schedule, and the realization of a specific permutation depends only on the content of the messages.

For the BSP model, we center our analysis around the following quantity.

**Definition 4.** *Given an algorithm  $\mathcal{B}$  that plays the envelope game on a switching DAG  $G = (V, E)$ , the redistribution potential at superstep  $j$ , denoted  $\eta_j$ , is the number of different placements of the  $N$  envelopes across the  $p$  processors at the beginning of the  $j$ -th superstep that are achievable, in different runs, while complying with the schedule of  $\mathcal{B}$ . (The order of the envelopes within a processor is irrelevant.)*

We let  $\eta_{K+1}$  denote the number of different placements just after the end of the last superstep. The plan to establish Theorem 1 is along the following lines:

- First, we show that, without loss of generality, we can confine our analysis to algorithms where all supersteps have degree at most 1. This greatly simplifies the subsequent counting arguments.
- Second, we establish that  $\eta_{K+1} \geq \gamma/(U!)^{N/U}$ , for any algorithm  $\mathcal{B}$  that plays the envelope game on DAG  $G$  with switching potential  $\gamma$ , in terms of the maximum number  $U$  of envelopes held by any processor at the end of the game.
- Third, we establish that  $\eta_{K+1} \leq (N/p)^{p^H}$ , due to the structure of the BSP model.
- Finally, Theorem 1 stems from a combination of the upper and the lower bounds on  $\eta_{K+1}$  in the two previous points.

The first three steps of the plan are each carried out by a separate lemma.

**Lemma 2.** *For any BSP algorithm  $\mathcal{A}$  with communication schedule independent of the input, there exists a BSP algorithm  $\mathcal{A}'$  with the following properties.*

- $\mathcal{A}'$  and  $\mathcal{A}$  compute the same function.
- The communication schedule of  $\mathcal{A}'$  is independent of the input; furthermore, in any superstep, each processor sends at most one message and receives at most one message.
- $\mathcal{A}'$  has the same BSP communication complexity as  $\mathcal{A}$ , that is,  $H_{\mathcal{A}'} = H_{\mathcal{A}}$ .

*Proof.* We obtain  $\mathcal{A}'$  from  $\mathcal{A}$  by replacing each superstep  $\mathcal{S}$  of degree  $h \geq 1$  with  $h$  supersteps of degree 1. (Supersteps where  $h = 0$  are left unchanged.) Specifically, let the communication schedule of  $\mathcal{S}$  be modeled by the bipartite *message multigraph*  $M = (S, D, F)$ , where  $S = \{s_1, \dots, s_p\}$ ,  $D = \{d_1, \dots, d_p\}$ , and  $F \subseteq S \times D$  where edge multiset  $F$  contains  $(s_i, d_k)$  with a multiplicity equal to the number of messages sent by  $P_i$  to  $P_k$  during  $\mathcal{S}$ . Clearly, each node in  $S$  and in  $D$  has degree at most  $h$ . It is a simple matter to augment  $M$  with edges so as to obtain a regular bipartite multigraph  $M' = (S, D, F')$  of degree  $h$ . We recall that Hall's theorem [29] ensures that any regular bipartite multigraph does admit a perfect matching. Therefore, by repeated applications of Hall's theorem,  $F'$  can be partitioned into  $h$  perfect matchings, i.e.,  $F' = \sum_{\nu=1}^h F_\nu$ , where  $M_\nu = (S, D, F_\nu)$  is regular of degree 1. Correspondingly, superstep  $\mathcal{S}$  can be replaced by the equivalent sequence of supersteps  $(\mathcal{S}_1, \dots, \mathcal{S}_h)$  where, for  $\nu \in [h]$ ,<sup>3</sup> superstep  $\mathcal{S}_\nu$  includes the messages of  $\mathcal{S}$  corresponding to the edges in  $F_\nu \cap F$ . (No message corresponds to the edges in  $(F' - F)$ , which were added just to make Hall's theorem directly applicable). All computations performed by  $\mathcal{S}$  as well as the receive operations relative to messages sent by the superstep preceding  $\mathcal{S}$  are assigned to  $\mathcal{S}_1$ . It is then straightforward to verify that the sequence  $(\mathcal{S}_1, \dots, \mathcal{S}_h)$  is equivalent to  $\mathcal{S}$  and contributes  $h$  to the BSP communication complexity of  $\mathcal{A}'$ , since each of its  $h$  supersteps has degree 1.  $\square$

**Lemma 3.** *Consider any algorithm  $\mathcal{B}$  for a BSP with  $p$  processors that plays the envelope game on a switching DAG  $G = (V, E)$  in  $K$  supersteps. Let  $N$  and  $\gamma$  be the switching size and the switching potential of  $G$ , respectively. At the end of the algorithm, if each processor holds at most  $U \leq N$  envelopes, the redistribution potential satisfies*

$$\eta_{K+1} \geq \frac{\gamma}{(U!)^{N/U}}. \quad (5)$$

*Proof.* Let  $U_i \leq U$  denote the number of envelopes held by processor  $P_i$  just after the end of the last superstep of algorithm  $\mathcal{B}$ . We know that, when varying the input-output correspondence for each internal node of  $G$  in all possible ways,  $\gamma$  different permutations of the envelopes over the arcs entering the outputs of  $G$  are generated. At most  $\prod_{i=1}^p (U_i!)$  envelope permutations can differ only by a rearrangement of the envelopes among arcs assigned to the same processor and thus result in the same placement of the envelopes across processors. Hence,

$$\eta_{K+1} \geq \frac{\gamma}{\prod_{i=1}^p (U_i!)}. \quad (6)$$

Considering that the quantity  $\prod_{i=1}^p (U_i!)$  is a superlinear function of the  $U_i$ 's and that  $\sum_{i=1}^p U_i = N$ , a convexity argument reveals that the maximum value is reached when  $\lfloor N/U \rfloor$  of the variables are set to the value  $U$ , one variable is set to  $(N \bmod U)$ , and the remaining variables are set to zero. Therefore,

$$\prod_{i=1}^p (U_i!) \leq (U!)^{\lfloor N/U \rfloor} (N \bmod U)! \leq (U!)^{\lfloor N/U \rfloor} (U!)^{(N \bmod U)/U} = (U!)^{N/U},$$

---

<sup>3</sup>Throughout the paper,  $[x]$  denotes the set  $\{1, 2, \dots, x\}$ .

where the second inequality follows since the function  $f(x) = \log(x!)/x$  is increasing when  $x > 0$ , and hence  $f(N \bmod U) \leq f(U)$ . Then, by plugging inequality  $\Pi_{i=1}^p (U_i!) \leq (U!)^{N/U}$  in (6), we obtain bound (5).  $\square$

**Lemma 4.** *Consider any algorithm  $\mathcal{B}$  for a BSP with  $p$  processors that plays the envelope game on a switching DAG  $G = (V, E)$ , with communication complexity  $H_{\mathcal{B}}$ . Let  $N$  be the switching size of  $G$ . At the end of the algorithm, the redistribution potential satisfies*

$$\eta_{K+1} \leq \begin{cases} (N/p)^{pH_{\mathcal{B}}} & \text{if } p \leq N/e, \\ e^{NH_{\mathcal{B}}/e} & \text{otherwise.} \end{cases}$$

*Proof.* For any  $i \in [p]$  and  $j \in [K+1]$ , where  $K$  is the number of supersteps in  $\mathcal{B}$ , we denote with  $t_{i,j}$  the number of envelopes held by processor  $P_i$  at the beginning of the  $j$ -th superstep. Clearly,  $\sum_{i=1}^p t_{i,j} = N$  for every  $j$ , since, by Rule 2 of the envelope game (Definition 3), the number of envelopes is invariant and always equal to  $N$ . (This constraint would not necessarily hold if recomputation were allowed.) Let, for every  $j \in [K]$ ,  $\mathcal{P}'_j$  to be the set of processors holding at least one envelope at the beginning of the  $j$ -th superstep, that is,  $\mathcal{P}'_j = \{P_i : t_{i,j} \geq 1\}$ . We claim that, for every  $j \in [K]$ ,

$$\eta_{j+1}/\eta_j \leq \prod_{i \in [p]: t_{i,j} \geq 1} t_{i,j}. \quad (7)$$

Thanks to Lemma 2, we can assume without loss of generality that all supersteps of  $\mathcal{B}$  have degree at most one. Consider superstep  $j$ , and consider a processor  $P_i \in \mathcal{P}'_j$ . At the beginning of the  $j$ -th superstep,  $P_i$  holds  $t_{i,j} \geq 1$  envelopes. From these,  $P_i$  can choose, in exactly  $t_{i,j} \geq 1$  different ways, the envelope to send in the  $j$ -th superstep, if any. The claim then follows because any of the  $\eta_{j+1}$  envelope placements immediately after superstep  $j$  correspond to one or more combinations of (a) one of the  $\eta_j$  placements achievable immediately before superstep  $j$ , and (b) one communication choice for each processor. Let  $p'_j = |\mathcal{P}'_j|$ . Given that, for every  $j \in [K]$ ,  $\sum_{i \in [p]: t_{i,j} \geq 1} t_{i,j} = N$ , the right-hand side of (7) is maximized when all the  $p'_j$  factors equal  $N/p'_j$ , and hence from (7) we obtain

$$\eta_{j+1}/\eta_j \leq (N/p'_j)^{p'_j}.$$

Standard calculus reveals that the function  $(N/p'_j)^{p'_j}$  has its maximum at  $p'_j = N/e$ . Therefore, since we must have  $p'_j \leq p$ , we have

$$\eta_{j+1}/\eta_j \leq \begin{cases} (N/p)^p & \text{if } p \leq N/e, \\ e^{N/e} & \text{otherwise.} \end{cases}$$

Multiplying both sides of the preceding relation over the  $H_{\mathcal{B}}$  supersteps of degree one; considering that, for a superstep with  $h = 0$ ,  $\eta_{j+1}/\eta_j = 1$ ; and observing that  $\eta_1 = 1$ , since the only placement of envelopes among processors before the first superstep is the one corresponding to the input placement protocol, the claim follows.  $\square$

We are now ready to prove Theorem 1, which we recall for convenience.

**Theorem 1.** *Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation a switching DAG  $G = (V, E)$  on a BSP with  $p$  processors. Let  $N$ ,  $\gamma$ , and  $\Delta$  be respectively the switching size, the switching potential, and the maximum out-degree of any node of  $G$ . If the sum of the in-degree of the output nodes evaluated by every processor is at most  $U$ , then the BSP communication complexity of algorithm  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} \geq \begin{cases} \frac{\log(\gamma/(U!)^{N/U})}{\Delta p \log(N/p)} & \text{if } p \leq N/e, \\ \frac{e \log(\gamma/(U!)^{N/U})}{\Delta N \log e} & \text{otherwise.} \end{cases} \quad (3)$$

where, by definition,  $\gamma \leq N!$  and  $N/p \leq U \leq N$ .

*Proof.* When  $p \leq N/e$ , combining Lemma 3 and Lemma 4 yields

$$(N/p)^{pH_B} \geq \frac{\gamma}{(U!)^{N/U}}.$$

Taking the logarithm of both sides, solving for  $H_B$ , and recalling that, by Lemma 1,  $H_A \geq H_B/\Delta$ , we conclude that

$$H_A \geq \frac{\log(\gamma/(U!)^{N/U})}{\Delta p \log(N/p)}.$$

The case  $p > N/e$  is shown analogously.  $\square$

## 5 The Cyclic Shift Technique

As discussed in Section 1.2, the switching potential lower bound becomes weaker as the maximum number  $U$  of output nodes held by a processor grows. In fact, the larger is  $U$ , the larger is the number of permutations that can be realized without interprocessor communication. However, there are classes of permutations that, in spite of their small cardinality, do require high communication even for large values of  $U$ . One such class is that of the *cyclic shifts* of order  $n$ , i.e., the permutations  $\sigma_0, \sigma_1, \dots, \sigma_{n-1}$  such that, for  $0 \leq k, i \leq n-1$ , we have  $\sigma_k(i) = (i+k) \bmod n$ . Intuitively,  $\sigma_k$  cyclically shifts to the right, by  $k$  positions. We say that a DAG  $G$  can realize all cyclic shifts of order  $n$  if 1)  $|V_{\text{in}}| = |V_{\text{out}}| = n$ ; 2) there exist labelings of the input nodes,  $V_{\text{in}} = \{v_0, v_1, \dots, v_{n-1}\}$ , and of the output nodes,  $V_{\text{out}} = \{v'_0, v'_1, \dots, v'_{n-1}\}$ , such that, for any  $0 \leq k < n$ , there exists a set of  $n$  arc-disjoint paths connecting  $v_i \in V_{\text{in}}$  to  $v'_{\sigma_k(i)} \in V_{\text{out}}$ , for any  $0 \leq i < n$ . Several interesting computational DAGs do realize all cyclic shifts of a given order. We now quantify the communication required by algorithms that evaluate such DAGs.

We need to introduce the following notation. Let  $P_1$  be one BSP processor, and let  $P_0$  be a virtual processor consisting of the other  $p-1$  processors. Denote by  $I_1$  and  $I_0$  the set of input nodes initially held by  $P_1$  and  $P_0$ , respectively. We also denote by  $O_1$  and  $O_0$  the set of output nodes evaluated by  $P_1$  and  $P_0$ , respectively.

**Lemma 5.** *Let  $G = (V, E)$  be a DAG with  $n$  input nodes and  $n$  output nodes, capable of realizing all the  $n$  cyclic shifts, with respect to some fixed labeling of inputs and outputs. Let  $\mathcal{A}$  be an algorithm that evaluates  $G$  (possibly, with recomputation) on a BSP with  $p \geq 2$  processors, such that initially each input is available to exactly one processor. Then the BSP communication complexity of  $\mathcal{A}$  satisfies*

$$H_A \geq \frac{|I_0||O_1| + |I_1||O_0|}{2n}.$$

*Proof.* We use an argument patterned after Vuillemin [51]. There are  $|O_1|$  cyclic shifts that match an input node in  $I_0$  with every output node in  $O_1$ ; similarly, there are  $|O_0|$  cyclic shifts that match an input node in  $I_1$  with every output node in  $O_0$ . Summing over all cyclic shifts, we get that  $|I_0||O_1| + |I_1||O_0|$  input nodes are assigned to output nodes contained in a different processor. As there are  $n$  cyclic shifts, there must be a shift that matches  $F = \lceil (|I_0||O_1| + |I_1||O_0|)/n \rceil$  input nodes to output nodes contained in a different processor. Since at the beginning of the computation each input is available to exactly one processor, a total of  $F$  messages are exchanged between  $P_0$  and  $P_1$ . Therefore,  $P_1$  receives or sends at least  $F/2$  messages, and hence  $H_A \geq F/2$ .  $\square$

As an immediate consequence of this lemma, we obtain the following.

**Theorem 2.** Let  $G = (V, E)$  be a DAG with  $n$  input nodes and  $n$  output nodes, capable of realizing all the  $n$  cyclic shifts, with respect to some fixed correspondence between inputs and outputs. Let  $\mathcal{A}$  be an algorithm that evaluates  $G$  (possibly, with recomputation) on a BSP with  $p \geq 2$  processors, such that initially each input is available to exactly one processor. Then, if some processor initially stores (exactly)  $q$  inputs, or some processor evaluates (exactly)  $q$  outputs, the BSP communication complexity of algorithm  $\mathcal{A}$  satisfies

$$H_{\mathcal{A}} \geq \frac{\min\{q, n - q\}}{2}.$$

*Proof.* Since, by hypothesis, each input is initially available to exactly one processor, we have  $|I_0| + |I_1| = n$ . Moreover, since we can assume without loss of generality that each output node is computed only once,  $|O_0| + |O_1| = n$ . Thus, if we let  $F$  denote the quantity  $(|I_0||O_1| + |I_1||O_0|)/n$ ,

$$F = \frac{(n - |I_1|)|O_1| + |I_1|(n - |O_1|)}{n} \geq \min\{|O_1|, n - |O_1|\}.$$

Therefore, if  $|O_1| = q$ , then  $F \geq \min\{q, n - q\}$ . A symmetric argument yields the same bound if  $|I_1| = q$ . In conclusion, if  $|I_1| = q$  or  $|O_1| = q$ , by applying Lemma 5 we obtain  $H_{\mathcal{A}} \geq F/2 \geq \min\{q, n - q\}/2$ , as desired.  $\square$

## 6 Applications

In this section, we show the versatility of the switching potential technique by applying it to the FFT DAG and to sorting and permutation networks. Further, we show that by applying the switching potential technique to some parts of a particular sorting network it is possible to obtain a lower bound stronger than the one obtained by applying the switching potential technique to the entire DAG.

### 6.1 Fast Fourier Transform

Let  $n$  be a power of two. In the  $n$ -input FFT DAG, a node is a pair  $\langle w, l \rangle$ , with  $0 \leq w < n$  and  $0 \leq l \leq \log n$ , and there exists an arc from node  $\langle w, l \rangle$  to node  $\langle w', l' \rangle$  if and only if  $l' = l + 1$  and either  $w$  and  $w'$  are identical or their binary representations differ exactly in the  $l'$ -th least significant bit. See Figure 2 for an example.

The  $n$ -input FFT DAG is a switching DAG since for any internal node  $v$  we have  $\delta_{\text{in}}(v) = \delta_{\text{out}}(v) = 2$ , and its switching size is  $N = 2n$ . Its switching potential is established by the following lemma.

**Lemma 6.** The FFT DAG of input size  $n$  has switching potential  $\gamma = 2^{n(\log n - 1)}$ .

*Proof.* For each internal node, there exist two possible one-to-one relations between the incoming arcs and the outgoing arcs. A configuration of the internal nodes is given by specifying the relation of each internal node, and each configuration automatically defines a particular set of  $N = 2n$  arc-disjoint paths. Since there are  $n(\log n - 1)$  internal nodes in the FFT DAG, there are  $2^{n(\log n - 1)}$  possible configurations of the internal nodes. No two configurations define the same set: this follows as a corollary of the property that in the FFT DAG there is a unique path between any input node and any output node [37].<sup>4</sup>  $\square$

Now we show that the FFT DAG can realize all cyclic shifts. This will enable the application of the results of Section 5.

---

<sup>4</sup>[37] discusses the property for the Omega network, which is isomorphic to the FFT network.



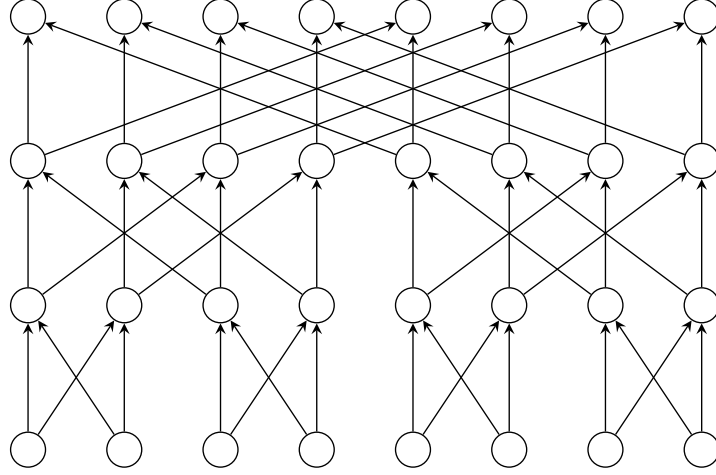


Figure 2: The FFT DAG on  $n = 8$  inputs and switching size  $N = 16$ . The nodes  $\langle w, l \rangle$  are placed so that  $w = 0, \dots, 7$  from left to right and  $l = 0, \dots, 3$  from bottom to top.

**Lemma 7.** *The  $n$ -input FFT DAG can realize all cyclic shifts of order  $n$ .*

*Proof.* In an  $n$ -input FFT DAG there exists a path from  $\langle w, 0 \rangle$  to  $\langle (w + k) \bmod n, \log n \rangle$ , for any  $0 \leq w < n$  and  $0 \leq k < n$ . This path visits  $\langle f(w, l), l \rangle$  for each  $0 \leq l \leq \log n$ , where  $f(w, l) = \lfloor w/2^l \rfloor 2^l + ((w + k) \bmod 2^l)$ . We now show that each one of the above sequences of nodes is a connected path. Clearly, we have  $f(w, 0) = w$  and  $f(w, \log n) = (w + k) \bmod n$ . The values  $f(w, l)$  and  $f(w, l + 1)$  differ at most in the  $(l + 1)$ -th least significant bit: indeed, the two values share the  $l$  least significant bits since  $f(w, l) \equiv f(w, l + 1) \bmod 2^l$ , and they also share the  $\log n - l - 1$  most significant bits since  $\lfloor f(w, l)/2^{l+1} \rfloor = \lfloor f(w, l + 1)/2^{l+1} \rfloor$ . Therefore the DAG nodes  $\langle f(w, l), l \rangle$  and  $\langle f(w, l + 1), l + 1 \rangle$  are connected and the path is well defined. Moreover, no two paths share a node for a given  $k$ : assume by contradiction that there exist two paths  $K_1 = (\langle w, 0 \rangle, \dots, \langle (w + k) \bmod n, \log n \rangle)$  and  $K_2 = (\langle w', 0 \rangle, \dots, \langle (w' + k) \bmod n, \log n \rangle)$ , with  $w \neq w'$ , that share a node; then, there must exist a value  $l$ , with  $0 < l < \log n$ , such that  $f(w, l) = f(w', l)$ ; however, since  $w = f(w, l) - (k \bmod 2^l)$ , it follows that  $w = w'$  which is in contradiction with the initial assumption  $w \neq w'$ ; we can thus conclude that paths  $K_1$  and  $K_2$  do not share any node. Therefore, we have that the FFT DAG can realize cyclic shifts for any  $0 \leq k < n$ , by suitably setting DAG nodes according to the set of  $n$  paths specified by  $k$ .  $\square$

We are now ready to prove Theorem 4, which provides the first lower bound on the BSP communication complexity required for evaluating a  $n$ -input FFT DAG that asymptotically matches upper bound (1) for any number of processors  $p \leq 2n/e$ . Before this, we shall use the cyclic shift technique to complement the dominator technique in order to obtain, for the FFT, a lower bound of the same form of (2) in Section 1.1, but under milder assumptions on the input/output protocol. Since, like the dominator technique, the cyclic shift technique does allow for recomputation, this results in a strengthened lower bound for the general case when recomputation is allowed, and thus is of independent interest.

**Theorem 3.** *Let  $\mathcal{A}$  be an algorithm that evaluates the  $n$ -input FFT DAG (possibly, with recomputation) on a BSP with  $p \geq 2$  processors, such that initially each input is available to exactly one processor. Then, if no processor evaluates more than  $n/\epsilon$  outputs, for some constant  $\epsilon > 1$ , the BSP communication complexity of  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} = \Omega\left(\frac{n \log n}{p \log((n/p) \log n)}\right).$$

*Proof.* Since the  $n$ -input FFT DAG has  $n(\log n - 1)$  internal nodes and  $n$  output nodes, at least one processor has to evaluate  $x = (n \log n)/p$  nodes. Let  $P_1$  be one of such processors, and let  $P_0$  be a virtual processor consisting of the remaining  $p - 1$  processors. Denote by  $I_1$  and  $I_0$  the set of input nodes initially held by  $P_1$  and  $P_0$ , respectively. We also denote by  $O_1$  and  $O_0$  the set of output nodes evaluated by  $P_1$  and  $P_0$ , respectively.

If  $|I_1| \leq \beta x / \log x$ , where  $\beta$  is a suitably small constant, then the lower bound in (2) applies, and the theorem follows. Otherwise, if  $|I_1| > \beta x / \log x$ , we shall leverage the hypothesis whereby no processor evaluates more than  $n/\epsilon$  outputs, for some constant  $\epsilon > 1$ . This gives  $|O_1| \leq n/\epsilon$ , which in turn implies  $|O_0| \geq n - n/\epsilon = n(\epsilon - 1)/\epsilon$ . Then, since by hypothesis each input is initially available to exactly one processor, and since by Lemma 7 the FFT DAG can realize all the  $n$  cyclic shifts, we can apply Lemma 5, obtaining

$$H_A \geq \frac{|I_0||O_1| + |I_1||O_0|}{2n} \geq \frac{|I_1||O_0|}{2n} > \frac{\beta n \log n \cdot n(\epsilon - 1)}{2n \cdot p \log((n/p) \log n) \cdot \epsilon} = \Omega\left(\frac{n \log n}{p \log((n/p) \log n)}\right),$$

as desired.  $\square$

We now use the switching potential technique in synergy with the cyclic shift technique to derive a tight lower bound for the case when recomputation is disallowed. We shall consider only the case  $p \leq N/e$ ; nevertheless, Theorem 1 also encompasses the case  $p > N/e$ . This case can be analyzed in a similar way as we analyze the case  $p \leq N/e$ .

**Theorem 4.** *Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation the  $n$ -input FFT DAG on a BSP with  $p \leq 2n/e$  processors, and let  $q$  be the maximum number of output nodes evaluated by a processor. If initially each input is available to exactly one processor, then the communication complexity of algorithm  $\mathcal{A}$  satisfies*

$$H_A \geq \frac{n \log(n/(8q^2))}{4p \log(2n/p)} + \frac{\min\{q, n - q\}}{4}.$$

Moreover, if  $q \leq n/2$ ,

$$H_A > \frac{n \log(n/2)}{8p \log(2n/p)}.$$

*Proof.* Since the in-degree of output nodes is two, we have that the sum of the in-degree of the output nodes evaluated by each processor is at most  $U \leq 2q$ . As recomputation is ruled out, we can apply Theorem 1 with  $N = 2n$ ,  $\Delta = 2$ ,  $U \leq 2q$ , and  $\gamma = 2^{n(\log n - 1)}$ , obtaining, after some manipulations,

$$H_A \geq \frac{\log(\gamma/(U!)^{N/U})}{\Delta p \log(N/p)} \geq \frac{n \log(n/(8q^2))}{2p \log(2n/p)}.$$

By hypothesis, each input is initially available to exactly one processor, and there exists some processor which evaluates (exactly)  $q$  output nodes; moreover, by Lemma 7, the FFT DAG can realize all cyclic shifts. Therefore, we can apply Theorem 2, which gives  $H_A \geq (1/2) \min\{q, n - q\}$ . By combining these two lower bounds we obtain the first claim of the theorem.

Consider now the case  $q \leq n/2$ . In this case we can write

$$H_A \geq \frac{n \log(n/(8q^2))}{4p \log(2n/p)} + \frac{q}{4} > \frac{n \log(n/(8q^2))}{8p \log(2n/p)} + \frac{q}{4}.$$

Let us denote with  $H'(q)$  the rightmost term of the above inequality, that is,

$$H'(q) = \frac{n \log(n/(8q^2))}{8p \log(2n/p)} + \frac{q}{4}.$$

By deriving  $H'(q)$  with respect to  $q$ , we can see that  $H'(q)$  is non-decreasing for  $q \geq n/(p \log(2n/p))$ . Therefore, since  $q \geq n/p$ , and since  $n/p > n/(p \log(2n/p))$  (because of the  $p \leq 2n/e$  hypothesis), we have that  $H'(q) \geq H'(n/p)$ , and thus

$$H_A > H'(q) \geq H'(n/p) = \frac{n \log(n/2)}{8p \log(2n/p)},$$

which proves the second claim of the theorem.  $\square$

## 6.2 Sorting and Permutation Networks

In this section we apply our technique to bound from below the BSP communication complexity of the computation DAGs that correspond to sorting and permutation networks. These networks, such as the Beneš permutation network [9] and the bitonic [7] and AKS sorting networks [3], can be interpreted as switching DAGs and have the property that they can realize all the possible permutations, and thus the switching potential technique can be naturally applied to them. Since our technique abstracts the DAG under consideration by considering only one general parameter (its switching potential), the lower bound obtained in this section is *universal* in the sense that it holds for *any* sorting or permutation network.

We now briefly recall the definitions of such networks. More complete descriptions can be found in [35, 38, 44]. A *comparator network* is an acyclic circuit of *comparators*. A comparator is a 2-input 2-output operator which returns the minimum of the two inputs on one output, and the maximum on the other. An  $n$ -input comparator network is called a *sorting network* if it produces the same output sequence on all  $n!$  permutations of the inputs. Thus, sorting networks can be seen as a simple model for data-oblivious sorting algorithms, that is, algorithms that perform the same set of operations for all values of the input data. A *routing network* is an acyclic circuit of *switches*. A switch is a 2-input 2-output operator which either passes its two inputs to its outputs, or it swaps them. An  $n$ -input routing network is called a *permutation network* if for each of the  $n!$  permutations of the inputs there exists a setting of the switches that creates  $n$  disjoint paths from the  $n$  inputs to the  $n$  outputs. Observe that, suitably modified to transmit messages, every sorting network is a permutation network, but the converse is not true.

A sorting/permutation network can be naturally modeled as a computation DAG by associating to each comparator/switch a pair of (internal) nodes of the DAG. Both nodes have the same two predecessors, and thus receive in input the same two values  $a$  and  $b$ , and both nodes have out-degree two, but one node computes the function  $x = \min\{a, b\}$  while the other computes the function  $y = \max\{a, b\}$ . The resulting DAG is therefore a switching DAG of input size  $n$ , switching size  $N = 2n$ , and with  $\Delta = 2$ . The following lemma shows that the DAG of any sorting or permutation network has switching potential  $\gamma \geq n!$ , and can realize all cyclic shifts.

**Lemma 8.** *The DAG of any sorting or permutation network with input size  $n$  has switching potential  $\gamma \geq n!$  and can realize all cyclic shifts of order  $n$ .*

*Proof.* Since any sorting or permutation network can perform all the  $n!$  permutations of  $n$  inputs, there exist  $n!$  sets  $\mathcal{S}$  of  $n$  arc-disjoint paths connecting input and output nodes. Each set  $S \in \mathcal{S}$  of  $n$  arc-disjoint paths determines a set of  $n$  paths from  $n$  outgoing arcs of the input nodes to  $n$  incoming arcs of the output nodes. In order to get the claimed switching potential, we have to construct additional  $n$  paths from the  $n$  outgoing arcs of the input nodes to the  $n$  incoming arcs of the output nodes that are not used in  $S$ . We observe that each path in a  $S \in \mathcal{S}$  uses only one of the two incoming arcs of each (non-input) node, and only one of the two

outgoing arcs of each (non-output) node. By exploiting the unused pair of incoming/outgoing arcs in each internal node, it is possible to uniquely construct the missing paths. Therefore, there exists  $n!$  sets of  $N = 2n$  arc-disjoint paths connecting input and output nodes, and the first part of the claim follows.<sup>5</sup> Furthermore, the set  $\mathcal{S}$  contains the set of  $n$  arc-disjoint paths of all the  $n$  cyclic shifts by definition of sorting and permutation networks, and the second part of the claim follows as well.  $\square$

We are therefore in a similar situation as for the FFT DAG, with a slightly different switching potential  $\gamma$ , and it is therefore sufficient to mimic the proof for Theorem 4. We have the following result.

**Theorem 5.** *Let  $\mathcal{N}$  be any sorting or permutation network with  $n$  inputs. Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation the DAG corresponding to  $\mathcal{N}$  on a BSP with  $p \leq 2n/e$  processors, and let  $q$  be the maximum number of output nodes evaluated by a processor. Then the BSP communication complexity of algorithm  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} \geq \frac{n \log(n/(4eq^2))}{4p \log(2n/p)} + \frac{\min\{q, n - q\}}{4}.$$

Moreover, if  $q \leq n/2$ ,

$$H_{\mathcal{A}} > \frac{n \log(n/e)}{8p \log(2n/p)}.$$

*Proof.* Analogous to the proof of Theorem 4, with  $\gamma \geq n!$  in place of  $\gamma = 2^{n(\log n - 1)}$ .  $\square$

We observe that for sorting and permutation networks we are using a lower bound on the switching potential  $\gamma$  which is lower (when  $n \geq 6$ ) than the value of the switching potential of the FFT DAG, and then the resulting lower bound on the BSP communication complexity has a lower constant inside the logarithmic term at the numerator with respect to the one in Theorem 4. This is due to the generality of our argument that applies to the entire family of sorting and permutation networks. Better bounds can be obtained for specific networks. For example, in the case of the Beneš permutation network, the same bound of Theorem 4 applies since the corresponding DAG contains an  $n$ -input FFT DAG.

Finally, we observe that the lower bound of Theorem 5 is asymptotically tight, as the computation DAG corresponding to the Beneš permutation network can be evaluated with the same strategy used for the FFT DAG, yielding a BSP communication complexity of  $O(n \log n / (p \log(n/p)))$ .

### 6.3 Boosting the Switching Potential Technique

Since  $\gamma \leq N!$ , if applied to an entire switching DAG, Theorem 1 cannot yield a lower bound larger than  $\Omega((N \log N)/(p \log(N/p)))$ . However, by applying the theorem to suitable parts of the DAG and composing the results, it is sometime possible to obtain asymptotically larger lower bounds. To illustrate the approach, we study the DAG of the *periodic balanced sorting network* (PBSN) [26], which consists of a sequence of  $\log n$  identical *blocks*. Specifically, we consider the case where  $n$  is a power of two and the block is the *balanced merging network* (BMN), as in [26]. (The analysis and the result would also apply when the block is the odd-even merging network).

The DAG of an  $n$ -input BMN is the following: a node is a pair  $\langle w, l \rangle$ , with  $0 \leq w < n$  and  $0 \leq l \leq \log n$ ; there exists an arc between two nodes  $\langle w, l \rangle$  and  $\langle w', l' \rangle$  if and only if  $l' = l + 1$ , and

---

<sup>5</sup> In a DAG corresponding to a sorting or permutation network with  $n$  inputs and internal nodes of in-degree  $\delta$ , the switching size is  $N = \delta n$  since there are  $\delta$  outgoing edges per input node. Therefore, the switching potential  $\gamma$  can be as large as  $N! = (\delta n)!$ . The present argument shows that  $\gamma$  is at least  $n!$ .

either  $w = w'$  are identical or  $w' = (i+1)n/2^l - j$  where  $i = \lfloor w2^l/n \rfloor$  and  $j = i \bmod (n/2^l)$ . The  $n$ -input BMN is a switching DAG since for any internal node  $v$  we have  $\delta_{\text{in}}(v) = \delta_{\text{out}}(v) = 2$ , and its switching size is  $N = 2n$ .

The DAGs of the BMN and of the FFT are isomorphic, that is, there exists an arc-preserving bijection between the two node sets [11]. Hence, the BMN has the same switching potential of the FFT DAG (i.e.,  $\gamma = 2^{n(\log n - 1)}$ , see Lemma 6) and can realize all cyclic shifts of order  $n$  (see Lemma 7). As a consequence, the lower bounds stated in Theorem 4 for the FFT DAG apply unchanged to the DAG of the BMN.

By separately applying the switching potential technique to each of the  $\log n$  BMN blocks of a PBSN, we obtain the following result.

**Theorem 6.** *Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation the DAG of the  $n$ -input PBSN, where the block is a BMN, on a BSP with  $p \leq 2n/e$  processors. Let  $q$  be the maximum number of output nodes of each block evaluated by a processor. If each input is initially available to exactly one processor and  $q \leq n/2$ , then the BSP communication complexity of  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} > \frac{n \log(n/2)}{8p \log(2n/p)} \left\lceil \frac{\log n}{2} \right\rceil.$$

*Proof.* Consider the sequence  $s_1, s_2, \dots, s_{\log n}$  of BMNs in the PBSN. For any  $1 \leq i < \log n - 1$ , the evaluations of  $s_i$  and  $s_{i+2}$  in algorithm  $\mathcal{A}$  cannot overlap in time: as BMN sorts any bitonic sequence [26], any input value can reach any output; therefore, no input value of  $s_{i+2}$  is ready until all output values of  $s_i$  have been computed. It follows that the evaluations of the  $\lceil \log n/2 \rceil$  odd BMNs cannot overlap in time. Since, by hypothesis, no processor evaluates more than  $q \leq n/2$  output nodes of each BMN, we can apply (the second part of) Theorem 4 to each BMN, and the claim follows.  $\square$

Observe that the PBSN contains  $\Theta(n \log^2 n)$  comparators, which is a factor  $\log n$  more than the optimal value. Therefore, it is natural that the BSP communication complexity for evaluating a PBSN is a factor  $\Omega(\log n)$  larger than the lower bound that holds for any sorting network (Theorem 5). Nevertheless, a lower bound of the form of the one given in Theorem 6 cannot be derived by applying the switching potential technique to the entire DAG of a PBSN. We are not aware of any prior lower bounds of this form for computations that correspond to the evaluation of PBSN.

## 7 The Switching Potential Technique in a Parallel I/O Model

In this section, we show how the switching potential technique can be adapted to yield lower bounds in computational models different from BSP. Specifically, we consider a parallel variant of the *I/O model*, which includes, as special cases, both the I/O model of Hong and Kung [31] and the LPRAM model of Aggarwal et al. [1].

Our parallel I/O model consists of  $p$  synchronous processors, each with a (fast) local memory of  $m$  words, which can access a (slow) shared memory of (potentially) unbounded size. In each step, all the processors perform the same instruction, which can be (i) an operation on data in the local memory, (ii) a move of a word from the shared to the local memory (i.e., a *read* operation) or (iii) vice versa (i.e., a *write* operation).<sup>6</sup> The *I/O complexity*  $H_{\mathcal{A}}$  of an algorithm  $\mathcal{A}$  is the number of steps where a read or write operation occurs.<sup>7</sup> We assume the input and output of

<sup>6</sup>Our lower bound can be trivially adjusted, dividing it by  $b$ , if an instruction can move  $b$  memory words instead of just one.

<sup>7</sup>We use here the same notation  $H_{\mathcal{A}}$  for the I/O complexity as for the BSP communication complexity, to highlight the similar role of the two metrics in the context of the switching potential technique.

algorithm  $\mathcal{A}$  to reside in the shared memory at the beginning and at the end of  $\mathcal{A}$ , respectively; (if this is not the case, our lower bound may still apply, after suitable modifications). We refer to [50] for a survey on algorithms and data structures for the I/O model.

Consider now an algorithm  $\mathcal{B}$ , for the parallel I/O model, playing the envelope game on a switching DAG  $G = (V, E)$ , with switching potential  $\gamma$ . We assume that each envelope occupies one memory word, whence there cannot be more than  $m$  envelopes in any local memory. Due to the restrictions of the I/O model, algorithm  $\mathcal{B}$  can move the first envelope from a node  $u$  only when all the  $\delta_{\text{in}}(u)$  input envelopes are in the local memory of the same processor.

As for the BSP model, a crucial observation is that the switching potential technique does not simply arise from the data movement implied by the  $\gamma$  permutations that contribute to the switching potential, but rather by the constraint that all those permutations must be realizable under the same schedule. Formally, the *schedule* of an algorithm is defined by the sequence of read and write operations, and by the memory locations in the shared and local memories involved in each read and write operation. Since the schedule is given, only the content (not the source and the destination) of a read and write operation can vary across different runs of the envelope game that can result in the realization of different permutations. Thus, at any given time, the memory locations (both shared and local) that contain envelopes are independent of the run of the game, while the mapping of the envelopes to those location will generally differ across runs. This fact allows us to introduce a notion of redistribution potential, appropriate for the parallel I/O model:

**Definition 5.** *Consider an algorithm  $\mathcal{B}$  for the parallel I/O model that plays the envelope game on a switching DAG  $G = (V, E)$ . At any given step of the algorithm, the envelope placement is the specification, for each envelope, of either the (address of the) shared memory location or the (index of the) processor whose local memory contains that envelope. (The exact position of the envelope within a local memory is irrelevant.) The redistribution potential at the  $j$ -th read/write operation, denoted  $\eta_j$ , is the number of different envelope placements, before the  $j$ -th read/write operation, that are achievable in different runs, while complying with the schedule of  $\mathcal{B}$ .*

From the initial and final conditions of the game, we have  $\eta_1 = 1$  and  $\eta_{H_{\mathcal{B}}+1} \geq \gamma$ .

Intuitively, read and local operations do not increase the redistribution potential. On the other hand, the increase due to write operations is bounded by the amount of envelopes held locally by each processor. This amount is naturally bounded by the size  $m$  of the local memory, but it is also effectively bounded by  $N/p$ , since the  $p$  processors together can at most hold  $N$  envelopes, and it turns out that a balanced allocation of envelopes to processors can result in the maximum increase of redistribution potential. These statements are substantiated in the next lemma, leading to a lower bound on the number of write steps needed to bring the redistribution potential from 1 to  $\gamma$ , hence to the I/O complexity.

**Lemma 9.** *The I/O complexity of an algorithm  $\mathcal{B}$  that plays the envelope game on a switching DAG  $G = (V, E)$  in the parallel I/O model with  $p$  processors, with local memory size  $m$ , satisfies*

$$H_{\mathcal{B}} \geq \frac{\log \gamma}{p \log \min\{m, N/p\}},$$

where  $\gamma$  is the switching potential of  $G$ .

*Proof.* An operation on data in the local memories cannot change the redistribution potential, since it does not affect the shared memory and does not change the set of envelopes present in each local memory. A read operation moves memory words from the shared memory to the local memories, in a way that is uniquely prescribed by the schedule of the algorithm. Hence,

to each envelope placement before the operation there correspond a unique placement after the operation, whence the redistribution potential does not increase.

On the other hand, each write operation increases the redistribution potential by at most a factor  $\prod_{i=1}^p N_i$ , where  $N_i$  denotes the number of envelopes currently in the  $i$ -th processor. In fact, the envelope to be moved to shared memory by the  $i$ -th processor can only be one of the  $N_i$  envelopes in its local memory. (For clarity, we observe that the source and the target addresses of the write are actually fixed by the schedule. It is the envelope currently located at the source address that can differ across different runs of the game, due to permutations of envelopes within a local memory.)

We now observe that, on the one hand, for each  $i$ , we have  $N_i \leq m$ , whence  $\prod_{i=1}^p N_i \leq m^p$ . On the other hand, by the invariance of the number of envelopes, we have  $\sum_{i=1}^p N_i = N$ , a constraint under which  $\prod_{i=1}^p N_i \leq (N/p)^p$ , as can be established by standard techniques. We then have  $\eta_{j+1} \leq (\min\{m, N/p\})^p \eta_j$ , whence  $\eta_{H_B+1} \leq \min\{m, N/p\}^{pH_B}$ . Since  $\eta_{H_B+1} \geq \gamma$ , the claim follows.  $\square$

We are now ready to provide a lower bound on the I/O complexity of any algorithm evaluating a switching DAG  $G$  without recomputation.

**Theorem 7.** *Let  $\mathcal{A}$  be any algorithm that evaluates without recomputation a switching DAG  $G = (V, E)$  in the parallel I/O model with  $p$  processors and with local memory size  $m$ . Let  $N$ ,  $\gamma$ , and  $\Delta$  be the switching size of  $G$ , the switching potential of  $G$ , and the maximum out-degree of any node of  $G$ , respectively. Then the I/O complexity of algorithm  $\mathcal{A}$  satisfies*

$$H_{\mathcal{A}} \geq \frac{\log \gamma}{\Delta p \log \min\{\Delta m, N/p\}}.$$

*Proof.* An algorithm  $\mathcal{A}$  that evaluates  $G$  on the parallel I/O model with local memory size  $m$  and  $p$  processors can be transformed into an algorithm  $\mathcal{B}$  that plays the envelope game on the parallel I/O model with local memory size  $\Delta m$  and  $p$  processors. Algorithm  $\mathcal{B}$  is obtained from  $\mathcal{A}$  with the following three changes. (1) Initially, for every  $u \in V_{\text{in}}$ ,  $\delta_{\text{out}}(u)$  envelopes are placed on  $u$  and each envelope is univocally assigned to an outgoing arc of the respective input node. (2) The computation in each internal node  $u$  is replaced with a switch that sequentially forwards the  $\delta_{\text{in}}(u)$  input envelopes to the  $\delta_{\text{out}}(u)$  output arcs according to some permutation. This replacement requires that  $\delta_{\text{out}}(u) \leq \Delta$  words are available to store the envelopes forwarded by  $u$ , whereas only one word was required to store the single output of node  $u$ : since the local memory is  $\Delta$  times larger than the one used by algorithm  $\mathcal{A}$ , there is enough space to store the at most  $\Delta$  envelopes for each of the  $m$  nodes kept in the local memory by  $\mathcal{A}$ . (3) For each arc  $(u, v)$  where  $P(v)$  differs from  $P(u)$ , the envelope is first written in the shared memory by  $P(u)$  and then read by  $P(v)$  (possibly, other processors can read and write the envelope between these two operations). As shown in Lemma 1, the above modifications guarantee that the six rules of the envelope game are satisfied.

We now observe that  $H_{\mathcal{B}} \leq \Delta H_{\mathcal{A}}$ . The first two changes do not increase the I/O complexity. On the other hand, the third change increases the I/O complexity by a factor  $\Delta$ : indeed, for each node  $u$ , the output values on the  $\delta_{\text{out}}(u)$  outgoing edges can be stored by algorithm  $\mathcal{A}$  in one word of the local/shared memory since the output values are indistinguishable; however, algorithm  $\mathcal{B}$  requires  $\delta_{\text{out}}(u) \leq \Delta$  words since envelopes are distinct; therefore the I/O complexity of  $\mathcal{B}$  is at most  $\Delta H_{\mathcal{A}}$ .

The theorem follows since, by Lemma 9, an algorithm  $\mathcal{B}$  playing the envelope game on an  $\Delta m$ -word local memory with  $p$  processors requires  $H_{\mathcal{B}} \geq (\log \gamma)/(p \log \min\{\Delta m, N/p\})$  I/Os. Since  $H_{\mathcal{B}} \leq \Delta H_{\mathcal{A}}$ , the main claim follows.  $\square$

We now analyze the above lower bounds for two special cases: the sequential I/O model [31] and the LPRAM [1]. The I/O model follows by the parallel I/O model by setting  $p = 1$ . If we consider the FFT DAG and assume that  $N \geq 2m$ , we obtain a lower bound of  $n \log(n/2)/(2 \log(2m))$ , which asymptotically matches the lower bound in [31]. On the other hand, the LPRAM is obtained by setting  $m = +\infty$ . In this case, we get for the FFT DAG a lower bound of  $n \log(n/2)/(2p \log(2n/p))$ , which asymptotically matches the lower bound in [1].

Interestingly, we observe that Theorem 7 gives a tight lower bound for the FFT DAG without additionally resorting to the cyclic shift technique, as we did in the BSP model. The reason for such behavior is that the I/O protocol requires envelopes to be stored in the shared memory at the beginning and end of the algorithm. Thus, the redistribution potential cannot be increased without I/O operations. In contrast, in the BSP model, envelopes are contained in the processors' local memories since there is no "external storage" where to store envelopes at the beginning and at the end of the algorithm. Therefore, if each BSP processor contains at most  $U$  of the  $N$  envelopes, it is possible to get  $(U!)^{N/U}$  permutations even without communication, by just rearranging envelopes within each processor (see the proof of Lemma 3). If  $U$  is sufficiently large compared to the switching potential  $\gamma$ , almost all permutations can be reached without communication and we then need the cyclic shift technique to reinforce the lower bound.

## 8 Conclusions

In this paper we have studied some aspects of the complexity of communication of parallel algorithms. We have presented new techniques for deriving lower bounds on communication complexity for computations that can be represented by a certain class of DAGs. We have demonstrated the effectiveness of this technique by deriving novel, mostly tight lower bounds for the FFT and for sorting and permutation networks.

The present work can be naturally extended in several directions, some of which are briefly outlined next. First, it would be interesting to apply the switching potential technique to other DAG computations beyond the few case studies of this paper. One example are DAGs that correspond to merging networks. We conjecture that the switching potential  $\gamma$  of any DAG corresponding to a merging network of input size  $n = 2^k$  satisfies  $\log \gamma = \Omega(n \log n)$ ; we also conjecture that the same bound holds for any network that can realize all cyclic shifts. It is also natural to explore the application of the switching potential technique to other models for distributed and hierarchical computation. Finally, one might ask whether the main lower bounds presented in this paper also hold when recomputation of intermediate values is allowed.

As a broader consideration, our lower bound techniques, as well as others in the literature, crucially exploit the circumstance that the execution of some algorithms embeds the evaluation of the same DAG for different inputs. The development of communication lower bound techniques for algorithms (e.g., heapsort or quicksort) which do not fall in this class remains an open, challenging problem.

**Acknowledgements.** The authors would like to thank Lorenzo De Stefani, Andrea Pietracaprina, and Geppino Pucci for insightful discussions.

## References

- [1] A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoret. Comput. Sci.*, 71(1):3–28, 1990.



- [2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Comm. ACM*, 31(9):1116–1127, 1988.
- [3] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in  $c \log n$  parallel steps. *Combinatorica*, 3(1):1–19, 1983.
- [4] G. Ballard, J. Demmel, A. Gearhart, B. Lipshitz, Y. Oltchik, O. Schwartz, and S. Toledo. Network topologies and inevitable contention. In *Proceedings of the 1st International Workshop on Communication Optimizations in HPC (COMHPC)*, pages 39–52, 2016.
- [5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.
- [6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6), 2012.
- [7] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, 1968.
- [8] A. Bäumer, W. Dittrich, and F. Meyer auf der Heide. Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model. *Theoret. Comput. Sci.*, 203(2):175–203, 1998.
- [9] V. E. Beneš. Permutation groups, complexes, and rearrangeable connecting networks. *Bell System Tech. J.*, 43:1619–1640, 1964.
- [10] S. N. Bhatt, G. Bilardi, and G. Pucci. Area-time tradeoffs for universal VLSI circuits. *Theoret. Comput. Sci.*, 408(2-3):143–150, 2008.
- [11] G. Bilardi. Merging and sorting networks with the topology of the omega network. *IEEE Trans. Comput.*, 38(10):1396–1403, 1989.
- [12] G. Bilardi and C. Fantozzi. New area-time lower bounds for the multidimensional DFT. In *Proceedings of the 17th Computing: The Australasian Theory Symposium (CATS)*, pages 111–120, 2011.
- [13] G. Bilardi, A. Pietracaprina, and P. D’Alberto. On the space and access complexity of computation DAGs. In *Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 47–58, 2000.
- [14] G. Bilardi, A. Pietracaprina, and G. Pucci. Decomposable BSP: a bandwidth-latency model for parallel and hierarchical computation. In *Handbook of Parallel Computing: Models, Algorithms and Applications*, pages 277–315. CRC Press, 2007.
- [15] G. Bilardi, A. Pietracaprina, G. Pucci, M. Scquizzato, and F. Silvestri. Network-oblivious algorithms. *J. ACM*, 63(1), 2016.
- [16] G. Bilardi and F. Preparata. Area-time lower-bound techniques with applications to sorting. *Algorithmica*, 1(1):65–91, 1986.
- [17] G. Bilardi and F. Preparata. Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. *Theory Comput. Syst.*, 32(5):531–559, 1999.
- [18] G. Bilardi, M. Scquizzato, and F. Silvestri. A lower bound technique for communication on BSP with application to the FFT. In *Proceedings of the 18th International European Conference on Parallel and Distributed Computing (Euro-Par)*, pages 676–687, 2012.

- [19] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, and J. Shun. Efficient algorithms with asymmetric read and write costs. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, pages 14:1–14:18, 2016.
- [20] R. A. Chowdhury, V. Ramachandran, F. Silvestri, and B. Blakeley. Oblivious algorithms for multicores and networks of processors. *J. Parallel Distrib. Comput.*, 73(7):911–925, 2013.
- [21] R. Cole and V. Ramachandran. Efficient resource oblivious algorithms for multicores with false sharing. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 201–214, 2012.
- [22] R. Cole and V. Ramachandran. Resource oblivious sorting on multicores. *ACM Trans. Parallel Comput.*, 3(4), 2017.
- [23] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297–301, 1965.
- [24] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. LogP: A practical model of parallel computation. *Comm. ACM*, 39(11):78–85, 1996.
- [25] P. de la Torre and C. P. Kruskal. Submachine locality in the bulk synchronous setting. In *Proceedings of the 2nd International Conference on Parallel Processing (Euro-Par)*, pages 352–358, 1996.
- [26] M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *J. ACM*, 36(4):738–757, 1989.
- [27] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1), 2012.
- [28] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM J. Comput.*, 29(2):416–432, 1999.
- [29] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10(1):26–30, 1935.
- [30] F. C. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8(6):553–578, 1965.
- [31] J.-W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC)*, pages 326–333, 1981.
- [32] J. E. Hopcroft, W. J. Paul, and L. G. Valiant. On time versus space. *J. ACM*, 24(2):332–337, 1977.
- [33] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.
- [34] B. H. H. Juurlink and H. A. G. Wijshoff. A quantitative comparison of parallel computation models. *ACM Trans. Comput. Syst.*, 16(3):271–318, 1998.
- [35] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 2nd edition, 1973.

- [36] R. R. Koch, F. T. Leighton, B. M. Maggs, S. B. Rao, A. L. Rosenberg, and E. J. Schwabe. Work-preserving emulations of fixed-connection networks. *J. ACM*, 44(1):104–147, 1997.
- [37] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, C-24(12):1145–1155, 1975.
- [38] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers Inc., 1992.
- [39] P. D. MacKenzie and V. Ramachandran. Computational bounds for fundamental problems on general-purpose parallel models. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 152–163, 1998.
- [40] C. H. Papadimitriou and J. D. Ullman. A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646, 1987.
- [41] M. S. Paterson and C. E. Hewitt. Comparative schematology. In *Proceedings of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127, 1970.
- [42] D. Ranjan, J. Savage, and M. Zubair. Strong I/O lower bounds for binomial and FFT computation graphs. In *Proceedings of the 17th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 134–145, 2011.
- [43] J. E. Savage. Extending the Hong-Kung model to memory hierarchies. In *Proceedings of the 1st Annual International Conference on Computing and Combinatorics (COCOON)*, pages 270–281, 1995.
- [44] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- [45] M. Scquizzato and F. Silvestri. Communication lower bounds for distributed-memory computations. In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 627–638, 2014.
- [46] D. C. Thompson. *A Complexity Theory for VLSI*. PhD thesis, Carnegie Mellon University, 1980.
- [47] A. Tiskin. The bulk-synchronous parallel random access machine. *Theoret. Comput. Sci.*, 196(1-2):109–130, 1998.
- [48] A. Tiskin. BSP (bulk synchronous parallelism). In *Encyclopedia of Parallel Computing*, pages 192–199. Springer, 2011.
- [49] L. G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, 1990.
- [50] J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.
- [51] J. Vuillemin. A combinatorial limit to the computing power of VLSI circuits. *IEEE Trans. Comput.*, 32(3):294–300, 1983.
- [52] C.-L. Wu and T.-Y. Feng. The universality of the shuffle-exchange network. *IEEE Trans. Comput.*, 30(5):324–332, 1981.

- [53] A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–213, 1979.

## APPENDIX

### A.1 An Improved Dominator Analysis for the FFT DAG

Given a DAG,  $D(k)$  denotes the maximum size of a set  $U$  of nodes that has a dominator  $W$  of size  $k$ . Hong and Kung showed that, for the FFT DAG,  $D(k) \leq 2k \log k$  [31, Theorem 4.1]. In this section we show that this bound can be improved to  $D(k) \leq k \log 2k$ , which is tight for every  $k$  power of two. This can be done by modifying the inductive proof of Hong and Kung accordingly. In the proof we will use the following lemma.

**Lemma 10.** *If  $0 \leq x \leq y$  and  $x + y \leq m$ , then*

$$x \log x + y \log y + 2x \leq m \log m.$$

*Proof.* Let  $m' = x + y$ . Standard calculus shows that, in the interval  $0 \leq x \leq m'/2$ ,

$$x \log x + (m' - x) \log(m' - x) + 2x \leq m' \log m'.$$

Since, by hypothesis,  $m' \leq m$ , we obtain  $x \log x + y \log y + 2x \leq m' \log m' \leq m \log m$ . □

We are now ready to show the result claimed at the beginning of this section.

**Proposition 1.** *For  $k \geq 2$ , any node set  $U$  of the FFT DAG that has a dominator set of size no more than  $k$  can have at most  $k \log 2k$  nodes.*

*Proof.* The proof is by induction on  $k$ . Since  $2k \log k = k \log 2k$  when  $k = 2$ , the base case is the same as in the proof of Hong and Kung.

We now mimic the inductive argument in the proof of Theorem 4.1 of [31]. We partition the nodes of the FFT DAG into three parts,  $A$ ,  $B$ , and  $C$ , defined as follows.

$$\begin{aligned} A &= \{\text{nodes } \langle w, l \rangle \text{ s.t. } 0 \leq w < n/2 \text{ and } l < \log n\}, \\ B &= \{\text{nodes } \langle w, l \rangle \text{ s.t. } n/2 \leq w < n \text{ and } l < \log n\}, \\ C &= \{\text{nodes } \langle w, l \rangle \text{ s.t. } l = \log n\}. \end{aligned}$$

The set of nodes  $\langle w, l \rangle \in C$  such that  $0 \leq w < n/2$  is said to be the upper half of part  $C$ , whereas the set of nodes  $\langle w, l \rangle \in C$  such that  $n/2 \leq w < n$  is said to be the lower half of part  $C$ . (See also the figure depicted in the proof of Theorem 4.1 of [31].) The dominator is partitioned into three parts,  $D_A$ ,  $D_B$ , and  $D_C$ , which have  $d_A$ ,  $d_B$ , and  $d_C$  nodes respectively. Without loss of generality we assume  $d_A \leq d_B$ . The set  $U$  is partitioned into three parts,  $U_A$ ,  $U_B$ , and  $U_C$ , which have  $u_A$ ,  $u_B$ , and  $u_C$  nodes respectively. If  $u_C > d_C + 2d_A$  then either there are more than  $d_A$  nodes of  $U_C \setminus D_C$  in the upper half of part  $C$  or there are more than  $d_A$  nodes of  $U_C \setminus D_C$  in the lower half of part  $C$ . In either case, there are more than  $d_A$  independent paths from the upper half inputs (i.e., the set of input nodes of the FFT DAG such that  $0 \leq w < n/2$ ) to these nodes in  $U_C \setminus D_C$ . Since the set  $D_A$  has only  $d_A$  nodes, this is impossible. Therefore we have

$$u_C \leq d_C + 2d_A.$$

By inductive hypothesis we have

$$\begin{aligned} u_A &\leq d_A \log 2d_A, \\ u_B &\leq d_B \log 2d_B. \end{aligned}$$

Thus

$$|U| \leq d_A \log 2d_A + d_B \log 2d_B + d_C + 2d_A.$$

Combining Lemma 10 with the hypothesis  $x + y \leq m$  yields

$$x \log 2x + y \log 2y + 2x \leq m \log 2m. \tag{8}$$

Since  $0 \leq d_A \leq d_B$  and  $d_A + d_B \leq k - d_C$ , applying (8) yields

$$|U| \leq (k - d_C) \log 2(k - d_C) + d_C \leq k \log 2k,$$

as desired. □