A Lower Bound Technique for Communication on BSP with Application to the FFT *

Gianfranco Bilardi, Michele Scquizzato, and Francesco Silvestri

Department of Information Engineering, University of Padova, Italy
{bilardi,scquizza,silvest1}@dei.unipd.it

Abstract. *Communication complexity* is defined, within the *Bulk Synchronous Parallel* (BSP) model of computation, as the sum of the degrees of all the supersteps. A lower bound to the communication complexity is derived for a given class of DAG computations 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 a novel and tight lower bound for the FFT graph.

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 paper, we consider the *Bulk Synchronous Parallel* (BSP) model of computation [23]. We develop a lower bound technique for a metric, called *communication complexity*, which captures a relevant component of the cost of BSP computations. This technique applies to a class of computations that can be modeled in terms of a *Directed Acyclic Graph* (DAG), whose vertices represent operations (of both input/output and processing type) and whose arcs represent data dependencies. The same DAG computation can be performed in many different ways, depending on the superstep and the processing element chosen for the execution of an operation and the way (routing path and schedule of the message along such a path) in which a value is routed from the processor that computes it to a processor that utilizes it. Our proposed technique further assumes that each operation is executed only once, but the case where repetitions are allowed is also of interest.

The complexity of communication of DAGs on various models of computation has received considerable attention. Lower bounds are often established through adaptations of the techniques of Hong and Kung [13] for hierarchical memory, or by critical path arguments, such as those in [1]. For applications of these and other techniques see [18, 2, 12, 9, 5, 14, 3] as well as [19] and references therein.

* This work was supported, in part, by MIUR-PRIN Project *AlgoDEEP*, by PAT-INFN Project *AuroraScience*, by the University of Padova Projects *STPD08JA32* and *CPDA099949*, and by the IBM *Visiting Scientist Program*.

The resulting bounds are often tight, but not in all cases. A notable example is the computation of an n-input FFT DAG on a BSP with p processors: when inputs are initially evenly distributed among the processors, an adaptation of the *dominator set* technique of [13] yields a lower bound of[1] $\Omega\left(\frac{n \log n}{p \log(n \log n/p)}\right)$ to the communication complexity, which does not match the best known upper bounds when $p = \omega(n/\log n)$. As this example indicates, communication lower bounds deserve further exploration.

The main contribution of this paper is the *switching potential* technique, to obtain communication lower bounds for DAG computations in the BSP model. The communication complexity of a BSP computation is defined as the sum of the degrees of all its supersteps. The proposed technique applies to DAGs with n input nodes where all nodes, except for inputs and outputs, have out-degree equal to the in-degree. Such a graph can be viewed as a *switching network* [19], whose switching potential $\gamma(n)$ is defined as the number of different permutations that it can realize. We show that, for executions of a DAG *without recomputation* of its nodes, the BSP communication complexity satisfies a suitable lower bound expressed in terms of the switching potential. As a corollary of this general result, we obtain a tight bound for the communication complexity of the FFT on the BSP. The bound has the form $\Omega\left(\frac{n \log n}{p \log(n/p)}\right)$ and matches an upper bound of [23] for any $p = O(n)$.[2] A similar bound was derived earlier in [12], for the special class of algorithms performing exclusively supersteps of degree $\Theta(n/p)$.

Our FFT lower bound has the same form as the lower bound derived for the communication complexity of the FFT in the LPRAM model, by Aggarwal, Chandra, and Snir [1]. In addition to being developed for a different model, the argument of [1] follows a different route: the lower bound is first established for sorting, then claimed (by analogy) for permutation networks, and finally adapted to the FFT network, by exploiting the property that, as shown in [24], the cascade of three FFT networks has the topology of a full permutation network. Finally, we observe that while when recomputation is not allowed our FFT lower bound improves on the dominator-set result mentioned above, the latter remains of interest when recomputation is allowed.

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, whose specification does not refer to such parameters, but are designed with the goal of achieving (near) optimality for all values of the parameters. Notable examples are *cache-oblivious algorithms* [11], *multicore-oblivious algorithms* [10] and, closer to the scenario of this paper, *network-oblivious algorithms* [8, 7], 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

[1] We denote by $\log n$ the logarithm in base two, and by $\ln n$ the natural logarithm.

[2] When $p = \Omega(n)$ a suitable adaptation of our argument gives an $\Omega(\log n)$ bound, which is also tight.

DAG, it is often assumed $p^2 \leq n$, where the complexity is $\Theta(n/p)$. Our results allow for the removal of such restrictions.

The rest of the paper is organized as follows. Section 2 introduces the concept of switching DAG and its switching potential. Then, it formulates the envelope game, a convenient framework for studying the communication occurring when evaluating a DAG. Section 3 briefly reviews the BSP model and develops a relationship between the switching potential of a DAG and its communication complexity on BSP, in the form of a mathematical program. The latter is analyzed in Section 4 and the results are applied to the FFT DAG. Finally, in Section 5 we draw some conclusions and discuss future work.

2 The Switching Potential of Computation DAGs

A *computation DAG* $G = (V, E)$ is a directed acyclic graph where nodes represent *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 $\delta_{in}(v)$, while the number of its successors is called its *out-degree* and denoted $\delta_{out}(v)$. A node v is called an *input* if $\delta_{in}(v) = 0$ and an *output* if $\delta_{out}(v) = 0$. We denote by V_{in} and V_{out} the set of input and output nodes, respectively. The remaining nodes are said to be *internal* and their set is denoted by V_{int}.

For many models of computation, the execution of an algorithm on a particular input can be naturally described by a computation DAG. Of particular interest is the case when this DAG is the same for all inputs of the same size n, and can then be denoted as $G(n)$. In fact, a number of graph-theoretic properties of $G(n)$ can be related to processing, storage, and communication requirements of the underlying algorithm, as well as to its amount of parallelism. In this context, we introduce one such property, the switching potential, defined for a class of relevant computation DAGs.

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

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

where the equality between the two summations is easily established.

It is not difficult to see that if, for any internal node of G, a one-to-one relation R is established between the incoming arcs and the outgoing arcs, then a set $\mathcal{R}$ of N arc-disjoint paths naturally arises, where paths are formed by the arcs that belong to the same equivalence class of R^*, the transitive closure of R. Let us now number the arcs incident upon input nodes from 1 to N (in some arbitrarily

chosen order) and do the same for the arcs incident upon the output nodes. Then, to the above set $\mathcal{R}$ there corresponds a permutation $\rho = (\rho(1), \rho(2), \ldots, \rho(N))$ of $(1, 2, \ldots, N)$, where $\rho(j)$ is the (number of the) last arc of the (unique) path in $\mathcal{R}$ whose first arc is numbered j. In terms of these concepts, we now introduce a key property of switching DAGs.

Definition 2. *Given a switching DAG $G = (V, E)$, consider the set Γ of all permutations corresponding to one or more sets of N arc-disjoint paths. The* switching potential *of G is defined as the number $\gamma = |\Gamma|$ of such permutations.*

Intuitively, if the internal nodes are viewed as switches, then items initially positioned on the input nodes can travel without conflicts on arc-disjoint paths and reach the output nodes. Indeed, in the special case where $\delta_{out}(v) = 1$ for all input nodes and $\delta_{in}(v) = 1$ for all output nodes, one has $N = n = |V_{in}| = |V_{out}|$ and the switching DAG can be viewed as a switching network in the traditional sense. Furthermore, if $\gamma = n!$ (all permutations can be realized), then the switching network is said to be a *permutation* (or, *rearrangeable*) *network*.

Next, we define the *envelope game*, to be played on a switching DAG G, based on a given one-to-one relation R between the incoming arcs and the outgoing arcs of each internal node. The game is subject to the following rules.

1. A set of N distinguishable *envelopes* is given, with exactly $\delta_{out}(v)$ envelopes placed on each input node v.
2. The set of envelopes remains invariant during the game and at any stage each envelope is at exactly one node of G.
3. One elementary move consists in moving one envelope along an arc, that is, from one node u to one node v, such that $(u, v) \in E$.
4. No envelope can be moved from a node v before all $\delta_{in}(v)$ envelopes that must be placed on v have actually been placed.
5. The game is completed when all envelopes have reached an output node.

Speaking rather informally, it is easy to see that from the orchestration of the envelope game on a given model of computation one can immediately derive a schedule without recomputation for evaluating a DAG G, on the same model, and viceversa. We just need to imagine that each envelope carries a (rewritable) card where, when a node of G is computed, its result is written on the card of each envelope currently at that node. It is also intuitive that, if nodes u and v of arc (u, v) are processed at different sites, then moving the envelope from u to v will result in some communication. It ought to be observed that given two arcs (u, v) and (u, w) with the same origin, if both v and w are processed at sites different from that of u, then two envelope moves will contribute to communication. This may result in an overcounting, in the case when v and w are processed at the same site, as just one of the two envelopes would be sufficient here, since they carry the same information. However, this overcounting is bounded from above by the maximum out-degree of any node, $\Delta = \max_{v \in V} \delta_{out}(v)$, which is a small constant for many interesting DAGs. The reverse process, of obtaining an execution of the envelope game from an evaluation of the DAG, is also straightforward, with an increase in communication upper bounded by Δ.

While the preceding considerations can be made precise only after having specified a model of computation, they do convey a useful intuition, which will be made rigorous for BSP in the next section, but could prove valuable on other models as well.

In the next section, we show that executing a switching DAG on BSP requires an amount of communication bounded from below by a certain function of its switching potential. This result is of interest, since several relevant computation DAGs are switching DAGs. Examples include the DAGs of networks of switches, of networks of comparators (e.g., for sorting or merging), the DAGs modeling computations of bounded-degree networks (as defined, e.g., in [17]), the DAGs of several stencil computations, and others.

3 Switching Potential and Communication on BSP

The Bulk Synchronous Parallel (BSP) model was introduced by Valiant [23] 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., [20] and references therein) together with a number of variants (such as D-BSP [22, 6], BSP* [4], E-BSP [15], and BSPRAM [21]).

The architectural component of the model consists of p processing elements $P_1, P_2, \ldots, P_p$, each equipped with 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 messages and, at the end, execute a global synchronization instruction. A message sent during a superstep becomes visible to the receiver only at the beginning of the next superstep. The running time of the i-th superstep is expressed in terms of two parameters g and ℓ as $T_i = w_i + h_i g + \ell$, where w_i is the maximum number of local operations performed by any processor and h_i is the maximum number of messages sent or received by any processor (i.e., the i-th superstep performs an h_i-relation). Intuitively, $1/g$ can be interpreted as the available bandwidth per processor, while ℓ as 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, $W_{\mathcal{A}} = \sum_{i=1}^{S_{\mathcal{A}}} w_i$ is the *local computation complexity* and $H_{\mathcal{A}} = \sum_{i=1}^{S_{\mathcal{A}}} h_i$ is the *communication complexity*. In this paper, we study the latter metric, which often represents the dominant component.

We focus on algorithms whose execution can be described by a computation DAG $G(n)$ solely determined by the input size n. The lower bounds are derived under the assumption that $G(n)$ is a switching DAG and that each node of the DAG (operation) is executed only once (no recomputation). In particular, we analyze the envelope game on G. In any given execution of such a game, a given node of G is assigned to a unique BSP processor. If $(u, v) \in E$ is an arc with u assigned to processor P and v assigned to processor $P' \neq P$, then the envelope must be routed from P to P', possibly through intermediate processors. A key

observation is that a BSP execution of the envelope game corresponding to a given relation R on arcs (intuitively, a setting of the switches) can be adapted to any other relation R' without changing the number of superstep of the sources and destinations sent at each superstep. Simply, the messages will carry different envelopes. We now introduce the critical quantity that we analyze.

Definition 3. *Consider the execution of a switching DAG $G(n)$ on the BSP. The* distribution potential *at superstep j, denoted $\eta_j(n,p)$, is defined as the number of different distributions of the N envelopes across the p processors that result at the end of the j-th superstep, when relation R is varied in all possible ways. (The order of the envelopes within a processor is irrelevant.)*

Intuitively, two tradeoffs are captured by the lower bound argument developed below. First, the communication complexity h of a given superstep is bounded from below in terms of the growth of the distribution potential in that superstep. Second, the distribution potential after the last superstep is bounded from below by the switching potential of the DAG.

At the beginning of the computation (after the 0-th superstep), $\eta_0(n,p) = 1$, since the only achievable distribution of envelopes among processors is the one corresponding to the input distribution protocol. Denote by U the maximum number of envelopes held by any processor at the end of the algorithm. If the algorithm completes in K supersteps, then $\eta_K(n,p) \geq \gamma(n)/(U!)^{N/U}$, where $(U!)^{N/U}$ is a corrective term due to the definition of $\eta_K(n,p)$. Let $o_i \leq U$ be the number of envelopes stored at the end of the algorithm in the i-th processor; then, there are at most $\Pi_{i=1}^{p}(o_i!) \leq (U!)^{N/U}$ envelope permutations differing only on the output values held by the same processor which yield the same distribution of the envelopes among processors. We denote the number of envelopes held by the i-th processor after the j-th superstep by $t_{i,j}$, for each $i \in [p]$ and $j \in [K]$, where $[x]$ denotes the set $\{1, 2, \ldots, x\}$. Clearly, by the rules of the envelope game, we have $\sum_{i=1}^{p} t_{i,j} = N$ and $t_{i,j} \geq 0$. (The latter equation would not necessarily hold if the envelope game were extended in order to allow for recomputation.)

Now consider a processor $i \in [p]$ and a superstep $j \in [K]$. The $t_{i,j}$ envelopes held by processor i after the j-th superstep are of two kinds: the $s_{i,j}$ envelopes that will be sent by i to some other processors during the subsequent superstep, and the other $r_{i,j} = t_{i,j} - s_{i,j}$ remaining envelopes. (The quantities $t_{i,j}$, $s_{i,j}$, and $r_{i,j}$ are all functions of n and p, although this dependence is not made explicit in the notation, for better readability. For the same reason, when clarity is not compromised, we will write η_j in place of $\eta_j(n,p)$.) Thus, there are $\binom{t_{i,j}}{s_{i,j}}$ choices of the set of envelopes to send and (given a fixed schedule of the algorithm, i.e., a fixed pattern of communication) these envelopes can be sent in at most $s_{i,j}!$ different ways to the other processors. Hence, at each superstep j each processor i has at most

$$\binom{t_{i,j}}{s_{i,j}} s_{i,j}! = \binom{r_{i,j} + s_{i,j}}{s_{i,j}} s_{i,j}! = \frac{(s_{i,j} + r_{i,j})!}{r_{i,j}!}$$

communications choices. Then, $\eta_j/\eta_{j-1} \leq \Pi_{i=1}^{p}(s_{i,j} + r_{i,j})!/r_{i,j}!$.

Assembling the above observations, we conclude that the communication complexity H of any algorithm for G is no smaller than the value an optimal solution to the following mathematical program.

$$\begin{aligned} H \geq \min & \sum_{j=1}^{K} \max_i s_{i,j} \\ \text{s.t.} & \prod_{j=1}^{K} \prod_{i=1}^{p} \frac{(s_{i,j}+r_{i,j})!}{r_{i,j}!} \geq \gamma(n)/(U!)^{N/U} \\ & \sum_{i=1}^{p} (s_{i,j}+r_{i,j}) = N \qquad \forall j \in [K] \\ & r_{i,j}, s_{i,j} \geq 0 \qquad \forall i \in [p], \forall j \in [K]. \end{aligned}$$

4 Solving the Mathematical Program

We relax the above system by observing that, for each $j \in [K]$,

$$\prod_{i=1}^{p} \frac{(s_{i,j}+r_{i,j})!}{r_{i,j}!} \leq \prod_{i=1}^{p} (s_{i,j}+r_{i,j})^{s_{i,j}}.$$

The relaxation will enable us to exploit the following lemma.

Lemma 1. *Let q and N be two positive integer values. Then, an optimal solution of the following mathematical program*

$$\begin{aligned} \max & \prod_{i=1}^{q} (a_i+b_i)^{a_i} \\ \text{s.t.} & \sum_{i=1}^{q} (a_i+b_i) = N \\ & a_i, b_i \geq 0 \qquad \forall i \in [q] \end{aligned}$$

must satisfy $b_i A = a_i (N-A)$ for each $i \in [q]$, where $A = \sum_{i=1}^{q} a_i$.

Proof. When $A = 0$ or there is just one $a_i \neq 0$ the lemma is straightforward. It is also easy to see that, in an optimal solution, $b_i = 0$ whenever $a_i = 0$. Hence, in the following we assume that $a_i \neq 0$ for each $i \in [q]$.

Let $A > 0$. We first study the case $q = 2$, and then use this as a building-block for determining the solution to the general case. Consider an optimal solution (a_1, b_1, a_2, b_2), and suppose a_1 and a_2 are given. Consider the first derivative in b_1 of the objective function. The constraint of the system imposes

$$b_1 + b_2 = N - (a_1 + a_2) = N - A, \tag{1}$$

hence we have

$$\frac{d}{db_1}(a_1+b_1)^{a_1}(a_2+b_2)^{a_2} = \frac{d}{db_1}(a_1+b_1)^{a_1}(N-a_1-b_1)^{a_2}$$
$$= a_1(a_1+b_1)^{a_1-1}(N-a_1-b_1)^{a_2} - (a_1+b_1)^{a_1}a_2(N-a_1-b_1)^{a_2-1}.$$

Since $a_1, a_2 > 0$, we have that the derivative is non-negative when $a_1(N-a_1-b_1) \geq a_2(b_1+a_1)$, that is, using Equation 1,

$$b_1 \leq \frac{a_1(N-a_1-a_2)}{a_1+a_2} = a_1\left(\frac{N-A}{A}\right).$$

Since the above derivative is first non-negative and then non-positive, the point b_1 where the value of the derivative is zero is unique, and thus must satisfy

$$b_1 = a_1\left(\frac{N-A}{A}\right) \qquad \text{and} \qquad b_2 = a_2\left(\frac{N-A}{A}\right).$$

We now turn our attention to the situation when q is arbitrary. Let (a,b) be an optimal solution, with $a = a_1, a_2, \ldots, a_q$ and $b = b_1, b_2, \ldots, b_q$, and a is given. We claim that $b_i = a_i\left(\frac{N-A}{A}\right)$ for each $i \in [q]$. In fact, suppose this is not true. Then, there must exist an optimal solution $(a,\bar{b}) \neq (a,b)$ and a pair of indices h,k such that $\bar{b}_h/a_h \neq \bar{b}_k/a_k$. We can prove this by contradiction. In fact, if $\bar{b}_h/a_h = \bar{b}_k/a_k$ for each $h \in [q]$, we have the following system of equations with q variables $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_q$ and q constraints:

$$\begin{cases} \dfrac{\bar{b}_h}{a_h} = \dfrac{\bar{b}_{h+1}}{a_{h+1}} & \forall h \in [q-1] \\ \sum_{j=1}^{q}(\bar{b}_j + a_j) = N. \end{cases}$$

To derive its unique solution, we can rewrite the last constraint as $\sum_{j=1}^{q}(\bar{b}_j/a_j + 1)a_j = N$. By the first $q-1$ constraints we have $\bar{b}_h/a_h = \bar{b}_k/a_k$ for each $h,k \in [q]$, and thus

$$\sum_{j=1}^{q}\left(\frac{\bar{b}_h}{a_h}+1\right)a_j = N \qquad \forall h \in [q],$$

that is

$$\left(\frac{\bar{b}_h}{a_h}+1\right)A = N \qquad \forall h \in [q],$$

which implies

$$\bar{b}_h = a_h\left(\frac{N-A}{A}\right) = b_h \qquad \forall h \in [q],$$

a contradiction. Therefore, we have shown that there exists a pair $(\bar{b}_h, \bar{b}_k)$ such that $\bar{b}_h/a_h \neq \bar{b}_k/a_k$. However, as seen for the case $q=2$, we can always find a pair $(\tilde{b}_h, \tilde{b}_k)$ such that $(\tilde{b}_h + a_h)^{a_h}(\tilde{b}_k + a_k)^{a_k} > (\bar{b}_h + a_h)^{a_h}(\bar{b}_k + a_k)^{a_k}$, thus contradicting the optimality of $(\bar{b}, v)$ and (u, v). This pair is

$$\tilde{b}_h = a_h\left(\frac{\bar{b}_h + \bar{b}_k}{a_h + a_k}\right) \qquad \text{and} \qquad \tilde{b}_k = a_k\left(\frac{\bar{b}_k + \bar{b}_h}{a_k + a_h}\right),$$

which is the solution of the system

$$\begin{cases} \dfrac{\tilde{b}_h}{a_h} = \dfrac{\tilde{b}_k}{a_k} \\ \tilde{b}_h + \tilde{b}_k = \bar{b}_h + \bar{b}_k. \end{cases}$$

It remains to check that the mathematical program is not unbounded. Observe that the objective function is real-valued and continuous on a domain which is non-empty, closed, and bounded. By the classical Weierstrass theorem, such a function admits a maximum, and this must be achieved at (a, b). □

We are now ready to prove the main result of the paper; we will establish the desired lower bound for the FFT graph as a corollary. The lower bound in the theorem exhibits a tradeoff between the communication and the maximum number U of envelopes held by a processor at the end of the algorithm: indeed, as U increases the number of envelope permutations differing only on the output values stored in the same processor increases as well, and thus the required communication complexity may decrease.

Theorem 1. *Let $x^\star$ be the value of an optimal solution of the mathematical program of the previous section. Then,*

$$x^\star \geq \frac{\ln(\gamma(n)/(U!)^{N/U})}{p \ln(eN/p)}.$$

Proof. We consider only supersteps where at least one message is sent over the network. (Supersteps without communication do not increase the number of envelope distributions.) We use the following notation: $s_j = \max_i s_{i,j}$ and $S_j = \sum_{i=1}^{p} s_{i,j}$. By setting $a_i = s_{i,j}$, $b_i = r_{i,j}$, and $q = p$ in Lemma 1, we have that, for a given superstep j,

$$\begin{aligned} \prod_{i=1}^{p} (s_{i,j} + r_{i,j})^{s_{i,j}} &\leq \prod_{i=1}^{p} \left(s_{i,j} + s_{i,j} \left(\frac{N - S_j}{S_j} \right) \right)^{s_{i,j}} \\ &= \prod_{i=1}^{p} s_{i,j}^{s_{i,j}} \left(\frac{N}{S_j} \right)^{s_{i,j}} = \left(\frac{N}{S_j} \right)^{S_j} \prod_{i=1}^{p} s_{i,j}^{s_{i,j}}. \end{aligned}$$

We partition the values of the index j into three sets K_1, K_2, and K_3 as follows: $j \in K_1$ iff $s_j > N/p$, $j \in K_2$ iff $N/(ep) < s_j \leq N/p$, $j \in K_3$ iff $s_j \leq N/(ep)$, where e is the base of the natural logarithm. For simplicity, we assume $p \leq N/e$. If $j \in K_1$, we have

$$\left(\frac{N}{S_j} \right)^{S_j} \prod_{i=1}^{p} s_{i,j}^{s_{i,j}} \leq e^{N/e} s_j^N,$$

because function $(N/x)^x$ is increasing in x until $x < N/e$ and $x = N/e$ is the maximum of the function. The constraints on the problem implies $\sum_{i=1}^{p} s_{i,j} \leq N$,

and then $\prod_{i=1}^{p} s_{i,j}^{s_{i,j}}$ is maximized when N/s_j values $s_{i,j}$ are set to s_j and the remaining ones to zero. On the other hand, when $j \in K_2$, we have

$$\left(\frac{N}{S_j}\right)^{S_j} \prod_{i=1}^{p} s_{i,j}^{s_{i,j}} \leq \left(\frac{N}{S_j}\right)^{S_j} s_j^{ps_j} \leq e^{N/e} s_j^{ps_j},$$

since, $s_{i,j} \leq s_j$. Finally, when $j \in K_3$ we have

$$\left(\frac{N}{S_j}\right)^{S_j} \prod_{i=1}^{p} s_{i,j}^{s_{i,j}} \leq \left(\frac{N}{S_j}\right)^{S_j} s_j^{ps_j} \leq \left(\frac{N}{ps_j}\right)^{ps_j} s_j^{ps_j} = \left(\frac{N}{p}\right)^{ps_j},$$

since the function $(N/x)^x$ is increasing in x until $x < N/e$ and the maximum value is $(N/(ps_j))^{ps_j}$ when $S_j \leq ps_j \leq N/e$.

Therefore, the first constraint of the minimization problem that we are studying can be relaxed as follows:

$$\prod_{j \in K_1} e^{N/e} s_j^N \prod_{j \in K_2} e^{N/e} s_j^{ps_j} \prod_{j \in K_3} \left(\frac{N}{p}\right)^{ps_j} \geq \frac{\gamma(n)}{(U!)^{N/U}}.$$

By taking the natural logarithm of both sides we have

$$\sum_{j \in K_1} \left(\frac{N}{e} + N \ln s_j\right) + \sum_{j \in K_2} \left(\frac{N}{e} + ps_j \ln s_j\right) + \sum_{j \in K_3} ps_j \ln(N/p) \geq \ln\left(\frac{\gamma(n)}{(U!)^{N/U}}\right).$$

Since $s_j > N/(ep)$ if $j \in K_1 \cup K_2$, we get

$$\sum_{j \in K_1} N \ln(es_j) + \sum_{j \in K_2} ps_j \ln(es_j) + \sum_{j \in K_3} ps_j \ln(N/p) \geq \ln\left(\frac{\gamma(n)}{(U!)^{N/U}}\right).$$

Let $\widehat{K}_i = \sum_{j \in K_i} s_j$. By the concavity of $\ln(es_j)$ and the convexity of $s_j \ln(es_j)$, we have that the first two summations are maximized when $s_j = N/p$ for each $j \in K_1 \cup K_2$, $|K_1| = \widehat{K}_1/(N/p)$ and $|K_2| = \widehat{K}_2/(N/p)$. Then we get

$$p\widehat{K}_1 \ln(eN/p) + p\widehat{K}_2 \ln(eN/p) + p\widehat{K}_3 \ln(N/p) \geq \ln\left(\frac{\gamma(n)}{(U!)^{N/U}}\right),$$

which yields the sought lower bound to the minimum solution of the problem:

$$\min \sum_{j=1}^{K} s_j \geq \widehat{K}_1 + \widehat{K}_2 + \widehat{K}_3 \geq \frac{\ln(\gamma(n)/(U!)^{N/U})}{p \ln(eN/p)}.$$

□

Corollary 1. *Let $\mathcal{A}$ be any algorithm computing an n-input FFT DAG on a BSP with p processors, and let U be the maximum number of envelopes held by any processor at the end of the algorithm. If $U \leq N/2 = n$ and recomputation is not allowed, then the communication complexity of the algorithm is*

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

Proof (Sketch). The FFT DAG has $n(\log n + 1)$ nodes and can produce at the output nodes $\gamma(n) = 2^{n(\log n+1)}$ distinct permutations of the $N = 2n$ envelopes. Hence, by Theorem 1, we get $H_{\mathcal{A}}(n,p) \geq (n/p)\log(2n/U^2)/\log(n/p)$. Since an FFT DAG can perform any cyclic shift of a vector, an $\Omega(U)$ lower bound follows by an argument based on the information flow of cyclic shifts [19, Lemma 10.5.2]. Therefore, we have

$$H_{\mathcal{A}}(n,p) = \Omega\left(\frac{n\log(2n/U^2)}{p\log(n/p)} + U\right).$$

We have that $U \geq 2n/p$. In this range, the above bound is minimized by setting $U = 2n/p$, yielding the stated bound. □

Several similar results can be obtained, for example for the Beneš permutation network, and for the bitonic and the AKS sorting networks.

5 Conclusions

In this paper, we have studied some aspects of the communication complexity of parallel algorithms. We have developed a new technique for deriving lower bounds on communication complexity for computations that can be represented by a certain kind of DAGs. We have demonstrated the power of this technique on the FFT DAG for which, assuming non-recomputation, the derived lower bound is tight for any possible values of parameters n and p, thus improving previous work.

It is natural to wonder whether our main lower bound holds (asymptotically) when recomputation of intermediate values is allowed. (Re-execution of operations is known, for instance, to enhance some simulations among networks [16].) While it is not difficult to see that our lower bound holds when each node of the DAG can be recomputed $O(1)$ times, in the general case (an adaptation of) our technique yields, for the FFT DAG, the same bound as that of the dominator-set result mentioned in the introduction. We feel that settling this question might shed new light on the role of recomputation in I/O- and communication-efficient computing, which is not yet fully understood.

Acknowledgments. The authors would like to thank Andrea Pietracaprina and Geppino Pucci for insightful discussions.

References

1. Aggarwal, A., Chandra, A.K., Snir, M.: Communication complexity of PRAMs. Theor. Comp. Sci. 71, 3–28 (1990)
2. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Comm. ACM 31(9), 1116–1127 (1988)
3. Ballard, G., Demmel, J., Holtz, O., Schwartz, O.: Graph expansion and communication costs of fast matrix multiplication. In: Proc. 23rd SPAA. pp. 1–12. ACM (2011)

4. Bäumker, A., Dittrich, W., Meyer auf der Heide, F.: Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model. Theor. Comp. Sci. 203(2), 175–203 (1998)
5. Bilardi, G., Pietracaprina, A., D'Alberto, P.: On the space and access complexity of computation DAGs. In: Proc. 26th WG. pp. 47–58. Springer (2000)
6. Bilardi, G., Pietracaprina, A., Pucci, G.: Decomposable BSP: A bandwidth-latency model for parallel and hierarchical computation. In: Handbook of Parallel Computing: Models, Algorithms and Applications, pp. 277–315. CRC Press (2007)
7. Bilardi, G., Pietracaprina, A., Pucci, G., Scquizzato, M., Silvestri, F.: Network-oblivious algorithms (2012), to be submitted
8. Bilardi, G., Pietracaprina, A., Pucci, G., Silvestri, F.: Network-oblivious algorithms. In: Proc. 21st IPDPS. pp. 1–10. IEEE (2007)
9. Bilardi, G., Preparata, F.: Processor-time tradeoffs under bounded-speed message propagation: Part II, lower bounds. Theor. Comp. Syst. 32(5), 531–559 (1999)
10. Chowdhury, R.A., Silvestri, F., Blakeley, B., Ramachandran, V.: Oblivious algorithms for multicores and network of processor. In: Proc. 24th IPDPS. pp. 1–12. IEEE (2010)
11. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. ACM Trans. Algorithms 8(1), 4:1–4:22 (2012)
12. Goodrich, M.T.: Communication-efficient parallel sorting. SIAM J. Computing 29(2), 416–432 (1999)
13. Hong, J.W., Kung, H.T.: I/O complexity: The red-blue pebble game. In: Proc. 13th STOC. pp. 326–333. ACM (1981)
14. Irony, D., Toledo, S., Tiskin, A.: Communication lower bounds for distributed-memory matrix multiplication. J. Par. & Distr. Comp. 64(9), 1017–1026 (2004)
15. Juurlink, B.H.H., Wijshoff, H.A.G.: A quantitative comparison of parallel computation models. ACM Trans. Comput. Syst. 16(3), 271–318 (1998)
16. Koch, R.R., Leighton, F.T., Maggs, B.M., Rao, S.B., Rosenberg, A.L., Schwabe, E.J.: Work-preserving emulations of fixed-connection networks. J. ACM 44(1), 104–147 (1997)
17. Leighton, F.T.: Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers (1992)
18. Papadimitriou, C.H., Ullman, J.D.: A communication-time tradeoff. SIAM J. Computing 16(4), 639–646 (1987)
19. Savage, J.E.: Models of Computation: Exploring the Power of Computing. Addison-Wesley (1998)
20. Tiskin, A.: BSP (bulk synchronous parallelism). In: Encyclopedia of Parallel Computing, pp. 192–199. Springer (2011)
21. Tiskin, A.: The bulk-synchronous parallel random access machine. Theor. Comp. Sci. 196(1-2), 109–130 (1998)
22. de la Torre, P., Kruskal, C.P.: Submachine locality in the bulk synchronous setting. In: Proc. 2nd Euro-Par. LNCS, vol. 1124, pp. 352–358. Springer (1996)
23. Valiant, L.G.: A bridging model for parallel computation. Comm. ACM 33(8), 103–111 (1990)
24. Wu, C.L., Feng, T.Y.: The universality of the shuffle-exchange network. Trans. Computers 30, 324–332 (1981)