Communication-Optimal Parallel Minimum Spanning Tree Algorithms

Extended Abstract

Micah Adler[1] Wolfgang Dittrich[2] Ben Juurlink[3] Mirosław Kutyłowski[4] Ingo Rieping[3]

Abstract

Lower and upper bounds for finding a minimum spanning tree (MST) in a weighted undirected graph on the BSP model are presented. We provide the first non-trivial lower bounds on the communication volume required to solve the MST problem. Let p denote the number of processors, n the number of nodes of the input graph, and m the number of edges of the input graph. We show that in the worst case a total of $\Omega(k \cdot \min(m, pn))$ bits need to be transmitted in order to solve the MST problem, where k is the number of bits required to represent a single edge weight. This implies that if a message can contain at most $O(k)$ bits, any BSP algorithm for finding an MST requires communication time $\Omega(g \cdot \min(m/p, n))$, where g is the gap parameter of the BSP model.

In addition, we present two algorithms whose running times match the lower bounds in different situations. Both algorithms perform linear work and use a number of supersteps independent of the input size. The first algorithm is simple but can employ at most m/n processors efficiently. Hence, it should be applied in situations where the input graph is relatively dense. The second algorithm is a randomized algorithm that performs linear work with high probability, provided that $m \geq n \cdot \log p$. This is the first linear work BSP algorithm for finding an MST in sparse graphs.

[1]Department of Computer Science, University of Toronto, Canada. Email: micah@cs.toronto.edu. Supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada, and by ITRC, an Ontario Centre of Excellence. This research was conducted in part while the author was at the Heinz Nixdorf Institute Graduate College, Paderborn, Germany.

[2]Bosch Telecom GmbH, Dept. UC-ON/ERS, Backnang, Germany. Email: Wolfgang.Dittrich2@pcm.bosch.de. This research was done while the author was working at the University of Paderborn, Germany.

[3]Heinz Nixdorf Institute, Dept. of Computer Science, University of Paderborn, Germany. Email: {benj,inri}@uni-paderborn.de. This research was partially supported by DFG-SFB 376 "Massive Parallelität" and EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

[4]Computer Science Institute, University of Wrocław, Poland. Email: mirekk@tcs.uni.wroc.pl.

1 Introduction

Parallel computation models like the BSP [26], the BSP* [5], the QSM [16], the CGM [12] and the LogP [11] are used more and more often for designing and analyzing parallel algorithms. In contrast to PRAM models, these models account for communication cost by introducing parameters that correspond to considerations such as communication bandwidth, latency or the startup cost of a message transmission. For this reason, algorithms derived using these models often have more practical relevance than PRAM algorithms.

All models mentioned above provide the incentive to design parallel algorithms that use less communication time than computation time. Such algorithms are called *communication-efficient.* For "regular" problems such as matrix multiplication, FFT and LU decomposition, this is relatively easy to achieve. Communication-efficient algorithms have also been obtained for a number of "irregular" problems such as sorting [15, 1, 17, 14], multisearch [5, 4, 18] and parallel priority queues [6]. However, for many graph problems such as list ranking, connected components and minimum spanning tree, it seems difficult to find communication-efficient algorithms. For many of these problems, very fast (linear time) sequential algorithms exist, but efficient parallel algorithms are unknown.

In this paper, we consider one of these problems, that of finding a minimum spanning tree (MST), and use the BSP model [26] as our cost model. Briefly, a BSP computer consists of a set of processor/memory modules, a routing network, and a mechanism for synchronizing the processors in a barrier style. The performance of a BSP computer depends on three parameters that specify the number of processors p, the computation-to-communication throughput ratio g, and the latency/synchronization cost L. A BSP computation consists of a sequence of *supersteps*, separated by barrier synchronizations. In a superstep, a processor can perform local operations, send messages, and receive messages. Messages received during one superstep cannot be used until the next superstep. The cost of a superstep is given by $w + g \cdot h + L$, where w is the maximum number of local operations performed by any processor, and h is the maximum number of messages sent or received by any processor. In this paper we often analyze the three cost components separately, and thus, we let W denote the total amount of *local work* performed by the algorithm, H denote the *communication time*, and S denote the number of supersteps. For more details and motivation for the BSP model, the reader is referred to [26].

Definition of the MST problem. Given a connected undirected graph $G = (V, E)$, each of whose edges has a weight $w(e)$. We let $n = |V|$ denote the number of nodes and

$m = |E|$ denote the number of edges of the input graph. A tree $T = (V, F)$ where $F \subseteq E$ and $|F| = |V| - 1$ is called a *spanning tree* of G if it connects all the vertices. The weight $w(T)$ of the spanning tree is

$$w(T) = \sum_{e \in F} w(e).$$

T is a *minimum spanning tree* (MST) if $w(T) \leq w(T')$ for all other spanning trees T'.

Overview of Results. A natural first step for any parallel MST algorithm is for each processor i to compute a minimum spanning forest on the edges local to i. Once this has been accomplished, the number of edges remaining in the graph is at most $\min(m, np)$. In this paper, we demonstrate that in the BSP model, the quantity $\min(m, np)$ defines the communication requirements of the MST problem. We provide the first non-trivial lower bounds on the communication volume required to solve the MST problem. In particular, we show that for any values of p, m, and n, the total number of bits that need to be communicated, in the worst case, is $\Omega(k \cdot \min(m, pn))$, where k is the number of bits required to represent an edge weight. The lower bound is information theoretic, and holds for Las Vegas randomized as well as deterministic algorithms. The implication of a lower bound on the communication volume required to solve a problem to the running time of a BSP algorithm solving that problem depends on b, the number of message bits contained in each BSP message packet. When $b = k$ and $b = \Omega(\log p)$, then the lower bound implies that the communication time required by any BSP algorithm is $\Omega(g \cdot \min(m/p, n))$. Note that the requirement $b = \Omega(\log p)$ is equivalent to saying that the number of message bits in each packet is not dominated by the number of bits needed to describe the packet destination." Furthermore, the techniques used to prove this bound do not assume the BSP model, and thus can be used to provide lower bounds for other models that limit interprocessor communication, including the LogP [11] and the QSM [16]. The lower bound techniques we develop are also quite general. For example, they can be used to provide similar lower bounds on the communication volume required for the problem of finding a minimum matching.

We also demonstrate upper bounds that asymptotically match the lower bound provided that $m \geq n \cdot \log p$, i.e., on all but very sparse graphs. To do this, we present two different algorithms: one for the case $m \geq np$, which we call MST-DENSE, and one for the case $m \leq np$, which we call MST-SPARSE. Algorithm MST-DENSE requires $W = O(n \cdot \log\log p + m/p)$ local computation time, $H = O(n)$ communication time and $S = O(\log p)$ supersteps. This algorithm requires no special distribution of the edges over the processors, as long as they are distributed evenly. Notice that the communication time H is independent of m (provided that $m \geq np$) and that the number of supersteps is independent of the problem size. Furthermore, when $m \geq np$, the communication time matches the lower bound. However, the main drawback of the algorithm is that it can employ at most m/n processors efficiently.

MST-SPARSE is a randomized algorithm based on the sequential linear-time algorithm presented in [21]. It uses several known ideas, but combines them together in a novel way. For example, it uses a BSP implementation of the algorithm due to Awerbuch and Shiloach [3]. This is a CRCW PRAM algorithm, and thus the best known work-preserving emulation on the BSP model needs $(n + m)/p \geq p^{\epsilon}$ slackness, constant $\epsilon > 0$. We show that the amount of slackness required as well as the number of supersteps can be reduced by keeping the graph in a representation in which all edges incident to a node are stored consecutively. Furthermore, MST-SPARSE uses an observation due to Johnson and Metaxas [19], which states that in order to reduce the number of nodes by a factor of k, it is sufficient to consider only the k cheapest edges incident to each node. In our algorithm, however, edges are discarded much more aggressively in order to keep the communication time low. Overall, MST-SPARSE requires $W = O((n \cdot \log p + m)/p)$ local work, $H = (12+\epsilon)\cdot(m/p) + o(m/p) + O((n \cdot \log p)/p)$ communication time, and $S = O(\log p \cdot \lceil \log p / \log((m + n)/p) \rceil)$ supersteps. Thus, it performs linear work provided that $m \geq n \cdot \log p$, and when $np \geq m \geq n \cdot \log p$, the communication time matches the lower bound.

Previous Work. Many PRAM algorithms for computing the MST exist. The algorithm due to Awerbuch and Shiloach [3] runs in $O(\log n)$ time on an $(n + m)$-processor PRIORITY CRCW PRAM, where the priority of a processor is determined by the weight of the edge assigned to it. This is a very powerful contention resolution rule. Johnson and Metaxas [19] gave an algorithm running in $O(\log^{3/2} n)$ time on an $(n + m)$-processor EREW PRAM. Using the same number of processors, Chong [9] presented an algorithm running in $O(\log n \cdot \log\log n)$ time. Karger [22] was the first to use randomization in parallel MST algorithms. He gave an EREW PRAM algorithm that requires $O(\log n)$ time and uses $m/\log n + n^{1+\epsilon}$ processors. A randomized sequential linear-time algorithm is given in [21]. A parallel version of this algorithm, running in $O(\log n)$ time and performing linear work on a CRCW PRAM, is described in [10].

Concurrent to our work, several groups have investigated the MST problem on different parallel computation models. Dehne and Götz [13] presented MST algorithms for the CGM model. They described a randomized algorithm that can be implemented on the BSP in time $O(\frac{n+m}{p} \cdot \log p + g \cdot \frac{n+m}{p} \cdot \log p + L \cdot \log p)$. If bit operations on the edge weights are possible, the algorithm can be made deterministic. They also gave an algorithm for dense graphs. If $m = \Omega(np)$, this algorithm takes $O(\frac{n+m}{p} + g \cdot \frac{n+m}{p} + L \cdot \log p)$ time on the BSP model. In comparison, algorithm MST-DENSE has an extra term $O(n \cdot \log\log p) = O((m/p) \cdot \log\log p)$ in its running time. On the other hand, the communication time of MST-DENSE is independent of m, and we also show that a slight modification of MST-DENSE needs only a constant number of supersteps. Furthermore, Dehne and Götz always assume that sufficient slackness is available ($\frac{n+m}{p} \geq p$), whereas we assume arbitrary slackness ($n + m \geq p$). Moreover, their algorithms are not work-efficient for sparse input graphs.

Poon and Ramachandran [24] considered the MST problem on the EREW PRAM. They designed a randomized algorithm that performs linear expected work and runs in $\tilde{T} = O(\log n \cdot \log\log n \cdot 2^{\log^* n})$ expected time. This implies that when $np \geq m$, linear-work and communication-optimality can also be achieved by simulating their algorithm on the BSP model. However, algorithm MST-SPARSE has three advantages. First, it requires at most S supersteps, where $\Omega(\log p) \leq S \leq O(\log^2 p)$, which is independent of the input size and therefore important in practice. A direct simulation of the Poon-Ramachandran algorithm needs $\tilde{T}$ supersteps on the BSP model. Second, it is designed directly for the BSP model, which obviates the need to hash the memory address space. Third, we introduce some novel techniques in order to reduce the constant in the term $O(g \cdot m/p)$ of the communication time required by algorithm MST-SPARSE. In addition, when $np \leq m$, simulating the Poon-Ramachandran algorithm no longer leads to a communication-optimal BSP algorithm. An advantage of the Poon-Ramachandran algorithm is that it does not require $m \geq n \cdot \log p$ in order to achieve linear work.

Organization. This paper is organized as follows. In Section 2 the lower bound technique is described and applied to the problem of finding the MST on the BSP model. In Section 3, BSP algorithms are presented for some basic operations that are used frequently in our algorithms. Algorithm MST-DENSE is described in Section 4. A top-level description of algorithm MST-SPARSE is given in Section 5. It employs three subalgorithms which are described in Sections 6 through Section 8. The full version of this paper is available as a technical report [2].

2 Communication Lower Bounds for MST

Communication Volume. We here present a general, model independent, framework for proving lower bounds on the amount of communication required by an algorithm. We then demonstrate the power of this framework by describing how to use it to obtain lower bounds on the amount of time required for communication in a BSP algorithm. We here introduce the concept of *communication volume*. This takes into account the amount of information processors obtain both *directly*, i.e., from bits that are sent and received, as well as *indirectly*, i.e. from information such as "no message was received at time t", or "the third message received came from processor i".

Our only assumption about the model of computation is that each processor communicates with the other processors through some communication facility, which we call here a *router*. For any model M, algorithm A, processor P and problem input I, we define a *transcript* $T(M, A, I, P)$, a string of bits which encodes all the information discernible to P and to any other processor regarding the flow of information between P and the router. Any encoding to binary strings is allowed, provided that the following condition is adhered to: for any algorithm A and two inputs I_1 and I_2 such that on algorithm A any processor is able to differentiate between I_1 and I_2 based solely on the flow of information between P and the router, $T(M, A, I_1, P) \neq T(M, A, I_2, P)$.

Let $|T(M, A, I, P)|$ be the number of bits in the transcript $T(M, A, I, P)$. When the input I' is chosen randomly from a distribution of inputs $\mathcal{I}$, then $|T(M, A, I', P)|$ is a random variable, and has an expectation $E[|T(M, A, \mathcal{I}, P)|]$, that depends on the method of encoding the transcript. We say that the *per processor communication volume* of a processor P in model M running algorithm A on distribution $\mathcal{I}$ is the minimum, over all encodings of the transcript, of $E[|T(M, A, \mathcal{I}, P)|]$. Thus, for example, when $\mathcal{I}$ consists of k equally likely inputs, all of which must have a different transcript for processor P, then the per processor communication volume of processor P must be at least $\log k$. Note that the per processor communication volume is essentially equivalent to the entropy of the communication between P and the remainder of the processors, on an input chosen from $\mathcal{I}$. The *total communication volume* is the sum of communication volumes of all processors.

The crucial feature of the above definition is that if we find a lower bound v on communication volume at processor P (by choosing an appropriate probability distribution of the inputs), then for each method of encoding the transcripts there is an input with the transcript at P of length at least v. (Note that which input has this property may depend on the method of encoding the transcript — one can adjust a communication protocol to favor a given input). Therefore, if we have a lower bound of v on communication volume at processor P, then we may less formally say that "there is an input, where communication volume at P is at least v".

We next show how to apply this idea to the BSP model.

Lemma 1 *A lower bound of m on the per processor communication volume, for any processor, implies a lower bound of $\Omega(\frac{gm}{b})$ on the time required for communication in the BSP model, provided that $L \geq g$ and that the BSP model computes with messages consisting of b bits, where $\log p = O(b)$.*

Note that in most implementations, the destination address is included in a packet that is sent to the router. When this is the case, the assumption that $\log p = O(b)$ holds whenever the contents of the message is not dominated by the bits that describe the packet destination.

Proof: We show that any execution of a BSP algorithm that completes in time $\frac{gm}{b}$ can be encoded into a transcript of length $O(m)$, from which the Lemma follows directly. Each superstep is represented by a sequence of messages sent or received by P. Each message is encoded by a triple: source address, destination address and message bits. The sequences describing supersteps are separated by special end-of-superstep marks. This allows, for example, processors to identify supersteps where they do not receive messages. By the assumption $L \geq g$, the computation considered may consist of at most $\frac{m}{b}$ supersteps. Hence, separating codes of supersteps requires $O(m/b)$ bits. The total number of messages sent in time $\frac{gm}{b}$ is at most m/b, and thus the total number of message bits is at most m. Also, by the assumption that each message has length $\Omega(\log p)$, the total number of bits required for source and destination addresses is at most $O(m)$. □

Lower bounds for Vector Minimum Problem. We assume that when an algorithm terminates, at least one processor outputs a correct answer and no processors outputs a false answer. To show a lower bound for the MST problem we first show a lower bound on a related problem, and then reduce this problem to the MST problem.

Definition 1 *Given n sets $X_1, \ldots, X_n$, where each set X_i consists of p keys, $x_{i,0}, \ldots x_{i,p-1}$.* The vector minimum *of X_i's is the vector* $(\min(X_1), \min(X_2), \ldots, \min(X_n))$. *We assume that there are p processors $P_0, \ldots, P_{p-1}$, and, for every $j < p$, processor P_j holds the numbers $x_{1j}, \ldots, x_{nj}$ (one number from each X_i). The vector minimum (VM) problem is to compute the vector minimum of $X_1, \ldots, X_n$.*

Consider the VM problem where each element of the sets X_i is a k-bit number. For the majority of inputs, the total communication volume of this problem is far below $\Theta(npk)$. However, it turns out that the worst case complexity is quite large:

Theorem 1 *Consider the VM problem where each of n sets X_i consists of k-bit numbers and the number of processors is p. For any deterministic algorithm solving this problem, there is an input for which the total communication volume is $\Omega(npk)$. Furthermore, for any randomized algorithm that always answers correctly, there is an input where the expected total communication volume is $\Omega(npk)$.*

Proof: We consider the behavior of deterministic algorithms running on an input drawn from a distribution β which we define below. We show that according to distribution β the expected communication volume is large. So for any deterministic algorithm and a method of encoding the transcript, there must exist some input which also requires a large communication volume. Furthermore, from Yao's Lemma [27], this also implies the stated result about randomized algorithms.

An input from the distribution β is chosen as follows.

- We take n seeds $s_1, \ldots, s_n$, each chosen independently and uniformly at random from the set of all $k-1$ bit binary strings. For $1 \leq i \leq n$ and $0 \leq j < p$, let $x_{ij} = s_i$.
- With probability $\frac{1}{2}$ we choose one processor P_u uniformly at random, and replace each seed held by P_u with a completely random $k-1$ bit string. This is referred to as *scrambling* processor P_u.
- For each j, $0 \leq j < p$, we append a 1 to each string x_{ij}, $1 \leq i \leq p$, except for strings x_{hj}, such that $h = j \bmod p$, to which we append a 0. (The bit appended is the least significant bit.)

Lemma 2 *For any processor P_j, and for the distribution on inputs to the VM problem β, the per processor communication volume at processor P_j is $\Omega(kn)$.*

From the linearity of expectation, Lemma 2 gives us that the expected total communication volume is $\Omega(kpn)$. Thus, Theorem 1 follows from Lemma 2.

Proof of Lemma 2. We use a variant of a technique developed by Yao for the study of communication complexity [28]. We consider the actions of processor P_j under the assumption that no other processor has been scrambled. Since this occurs with probability at least $\frac{1}{2}$, this can increase the expected number of bits by at most a factor of 2.

We allow the processors different from P_j to determine all the seed values $s_1, \ldots, s_n$ with no charge (this can only decrease the complexity of the problem.) We call the string that processor P_j would hold if it has not been scrambled its *standard string*, and denote this by S. We denote by (S,u) the input where the standard string is S and processor P_j has not been scrambled, and by $(S,\bar{S})$ the input where the standard string is S and P_j has been scrambled so that it holds $\bar{S}$. We consider pairs of inputs, called *distinguished pairs*, of the form (S,u) and (S',u), where S and S' differ on some bit in s_i, where $i \neq j \bmod p$.

Claim. The communication transcript between processor P_j and the remaining processors must differ for inputs (S,u) and (S',u), if (S,u) and (S',u) is a distinguished pair.

We assume that (S,u) and (S',u) are a distinguished pair such that the communication between P_j and the rest is the same for inputs (S,u) and (S',u). We show that there is an input where some processor produces a false answer. Let $s'_1, \ldots, s'_n$ be the seeds corresponding to S'. Let s_i be a seed such that $s_i \neq s'_i$ and $i \neq j \bmod p$, and assume w.l.o.g. that $s'_i < s_i$. Now, consider the input (S,S'). Let $a_i = \min X_i$. If P_j outputs the value a_i on the input (S,S'), then it must output the same value on the input (S',u), since computations on (S,S') and (S',u) are indistinguishable for P_j. However, on input (S,S'), $a_i = s'_i1$ and on input (S',u), $a_i = s'_i0$. So some other processor must output a_i on (S,S'). But, this processor must output the same value on input (S,u), since P_j behaves in exactly the same way while communicating with the remaining processors for (S,S') and (S,u). However, on input (S,S'), $a_i = s'_i1$ and on input (S,u), $a_i = s_i0$.

To complete the proof of the lemma, we condition upon the event that the seeds s_h, for $h = j \bmod p$, are set. The number of completions of these seeds into string S is at least $2^{(k-1)n(p-1)/p}$. Each of such inputs (S,u) is equally probable and they together have probability $\frac{1}{2}$. So, the expected per processor communication volume is $\Omega(\log(2^{(k-1)n(p-1)/p})) = \Omega(kn)$. □

MST versus VP problem. In some cases, solving MST problem requires solving the VM problem. Consider the following example. There are p processors and $2p$ nodes of the graph partitioned into sets A and B of size p each. Every pair of nodes in A are connected by an edge of weight 0, pairs of nodes in B are not connected, and each node of A is connected with all nodes of B with edges of arbitrary weights. The out edges of the ith node of A are given to the ith processor. In order to compute MST of this graph, it is necessary to determine, for each element of B, the minimum weight edge connecting it with the nodes of A. Thus, we have to solve a VM problem. Adjusting this example and using the bound on VM from Theorem 1, we get the following corollaries:

Corollary 1 *Let $p \leq m/n$. Consider the MST problem for graphs with n vertices and m edges with weights encoded by k-bit numbers on a p processor system. Then there is an input that requires total communication volume $\Omega(npk)$.*

Corollary 2 *Let $p \geq m/n$, $m \geq 2n$. Consider the MST problem for graphs with n vertices and m edges with weights encoded by k-bit numbers on a p processor system. Then there is an input that requires total communication volume $\Omega(mk)$.*

3 Basic BSP Algorithms

In this section we present the BSP complexity of some basic algorithms which are used as subroutines in our algorithms.

Lemma 3
(a) [8] *Ranking a list of size n and computing the Euler-Tour of a tree of size n can be performed in time $O(\frac{n}{p} \cdot \log p + g \cdot \frac{n}{p} \cdot \log p + L \cdot \log p)$ on the BSP model.*
(b) Let $s \geq p$ and let s keys be distributed evenly among the processors. Selecting the k-smallest key takes $O(s/p)$ computation time and $g \cdot s/p + O(g \cdot \sqrt{(n+m)/p} \cdot \sqrt{s}/p + L \cdot \log p/\log((n+m)/p))$ communication time in the BSP model with probability at least $1 - n^{-c}$ for any constant $c > 0$.

For result (b) we employ the selection algorithm given in [6], but slightly increase the degree of the broadcast tree. This increases the communication time, but reduces the number of supersteps. Since other parts of algorithm MST-SPARSE require $g \cdot (n+m)/p$ communication time, this technique affects the total complexity only by a constant factor. This technique can also be used to perform a single-item prefix operation in time $T_{\text{prefix}} = o(g \cdot (n+m)/p) + O(L \cdot \lceil \log p / \log((m+n)/p) \rceil)$.

4 Simple Algorithm for Dense Graphs

In this section we describe a simple algorithm for finding an MST in relatively dense graphs (i.e., graphs for which $m \geq pn$, but not necessarily $m = \Theta(n^2)$).

Algorithm 1 MST-DENSE

(1) Every processor locally computes the minimum spanning forest of the graph induced by the edges stored in its local memory. Edges not in any MSF cannot belong to the MST and can therefore be discarded. (W.l.o.g. we assume that the weights are different.)

(2) Perform $2 \cdot \log \log p$ so-called Borůvka-steps [25]. In every step the cheapest edge incident to each node is selected, and every connected component defined by the selected edges is contracted into a single "supervertex". This is done as follows:

(2.1) Every processor computes the locally cheapest edge incident to each supervertex. After that, the processors collectively determine the cheapest edge incident to each supervertex by performing a vector minimum operation.

(2.2) Processor $p-1$ sequentially computes the connected components of the graph defined by the edges selected in Step (2.1), determines the number of supervertices n' remaining, and labels the vertices by numbers 0 through $n'-1$ such that the vertices belonging to the same component receive the same number. After that, it broadcasts the vector of length n' containing the new node numbers to all other processors.

(2.3) Every processor relabels the edges it contains with the new labels of the endpoints and eliminates any self-loops.

(3) Since the size of each component at least doubles in every Borůvka-step, at most $n/\log^2 p$ supervertices remain after the previous step. Now, every processor locally computes the MSF of the graph G' induced by the set of remaining supervertices and the edges stored in its local memory.

(4) The p MSFs are merged to form the MST of G'. This is done as follows. First, each processor P_i, $p/2 \leq i < p$, sends its MSF to processor $P_{i-p/2}$. Thereupon, every processor P_i, $0 \leq i < p/2$, merges its MSF with the MSF it received from processor $P_{i+p/2}$. This is repeated recursively on the processors $P_0, \ldots, P_{p/2-1}$. After $O(\log p)$ supersteps, processor P_0 computes the MSF of the graph G'. The MST of the original graph consists of the edges identified in Step (2) and the MSF of G'.

- End of Algorithm -

Theorem 2 *If $p \leq m/n$, then the BSP cost of* MST-DENSE *is given by $W = O(m/p + n \cdot \log \log p)$, $H = O(n)$, and $S = O(\log p)$. If $p \cdot \log \log p = o(m/n)$, then* MST-DENSE *is 1-optimal.*

Proof: Omitted. In Step (2) we employ the two-phase broadcast given in [20]. □

When the density of the input graph is slightly larger, a constant number of supersteps can be achieved, albeit at the cost of a slight increase in the communication time. In this case, Step (2) and (3) are omitted.

Theorem 3 *If $m/n \geq p^{1+\epsilon}$ (constant $\epsilon > 0$), then the MST can be computed in $W = O(m/p + n)$, $H = O(m/p^{1+\epsilon/2})$, and $S = O(1)$ time on the BSP model. Algorithm* MST-DENSE *is 1-optimal.*

Proof: Step (1) takes $O(m/p + n)$ time. After Step (1), every processor contains a subgraph consisting of at most $n-1$ edges. In Step (4), instead of a binary tree, we employ a t-ary tree, where $t = p^{\epsilon/2}$. □

Algorithm MST-DENSE covers the dense case, i.e., graphs for which $m \geq pn$. Moreover, it has a very simple structure, and we therefore believe that it can be implemented very efficiently.

5 Algorithm for All But Very Sparse Graphs

The main drawback of algorithm MST-DENSE is that it can employ at most m/n processors efficiently. In this section, we give a top-level description of an algorithm for the case that more than m/n processors are available. It uses three subroutines: BSP-AS, BSP-AS-SELECT, and BSP-VERIFY. They are described in Section 6, 7 and 8, respectively. Subroutine BSP-VERIFY is only sketched here, a detailed description can be found in the full paper [2].

Let $k = m/n$ be the density of the input graph.

Algorithm 2 MST-SPARSE(G, k)

(1) Reduce the number of nodes of G by a factor k by performing $\log k$ rounds of algorithm BSP-AS-SELECT. Form the graph $G' = (V', E')$ by contracting every tree that was constructed by the algorithm into a single vertex. G' consists of $n' \leq n/k$ nodes and $m' \leq m$ edges.

(2) Construct a sample graph G'_s of G' by including every edge with probability k^{-1}. G'_s consists of at most $O(m'/k) = O(n)$ edges, with high probability.

(3) Find the minimum spanning forest F'_s of the graph G'_s using algorithm BSP-AS.

(4) Using algorithm BSP-VERIFY, determine and delete all edges of G' which cannot belong to the MST. By the sampling lemma of [21], the remaining graph G'' has $O(k \cdot n') = O(n)$ edges.

(5) Compute the MST T'' of the graph G'', again by using algorithm BSP-AS. The MST of the original graph consists of the edges of T'' plus the edges found in Step (1).

- End of Algorithm -

The total BSP complexity of algorithm MST-SPARSE is given in the following theorem.

Theorem 4 *Let $p \leq m + n$, $\log p \leq \frac{m}{n} \leq p$ and $\epsilon > 0$ constant. The BSP cost of algorithm* MST-SPARSE *is given by $W = O(\frac{m}{p} + \frac{n}{p} \cdot \log p)$, $H = (12+\epsilon) \cdot \frac{m}{p} + o(\frac{m}{p}) + O(\frac{n}{p} \cdot \log p)$, and $S = O(\log p \cdot \lceil \frac{\log p}{\log((m+n)/p)} \rceil)$ with probability at least $1 - p^{-c}$ for any constant $c > 0$. If $m \geq n \log n$ the runtime bound holds with probability at least $1 - n^{-c}$.*

Proof: The total BSP cost of algorithm MST-SPARSE is obtained by simply adding the cost of every step. The probability that MST-SPARSE requires more than the claimed running time is determined as follows. In Step (1), the runtime bound of algorithm BSP-AS-SELECT fails with probability at most $k \cdot p^{-c}$, for some constant $c > 0$. Since $k \leq \log p$, this probability is small enough. In Step (2), a sample graph is constructed by including every edge independently with probability k^{-1}. By applying Chernoff bounds, the sample graph consists of $\Theta(m/k)$ edges with probability at least $1 - \exp(-cn)$. Given a graph G with n' nodes, let H be a subgraph of G obtained by including each edge with probability p, and let F be the minimum spanning forest of H. The sampling lemma of Karger, Klein and Tarjan [21] shows that the number of F-light edges of G (edges which can not be discarded) is at most n'/p with probability at least $1 - \exp(-cn')$. In algorithm MST-SPARSE we have $n' = n/k$ and $p = k^{-1}$ where $k = m/n$, which implies that at most n edges remain with probability $\geq 1 - \exp(-c \cdot n/k)$. This probability is high enough when $n/k \geq \log n$. Since $k = m/n$, this is the case if $m \leq n^2/\log n$. The remaining case ($m > n^2/\log n$) can be dealt with by choosing $k = m/(n \cdot \log n)$. It is easy to verify that this choice affects the running time of the algorithm by at most a constant factor. □

6 BSP Implementation of the Awerbuch-Shiloach Algorithm

In this section, we describe a BSP implementation of the CRCW PRAM algorithm due to Awerbuch and Shiloach [3], which will be denoted by BSP-AS. Briefly, the algorithm iteratively executes three steps. In the first step, all processors assigned to edges emanating from a *star* (tree of depth 1) try to *hook* the star onto another tree. In Step (2), cycles are eliminated by deleting the edge pointing from the smaller-numbered vertex to the larger-numbered vertex. Finally, in Step (3) the height of each tree is reduced by an operation called *shortcutting* in which every vertex adopts its grandparent as its new parent.

The crux in our algorithm is to show how concurrent reads and writes can be implemented efficiently on the BSP model. This is usually done by sorting but sorting requires too many supersteps if the slackness is less than polynomial [17]. Our approach uses a data structure in which all edges incident to a node are stored consecutively. We show that this technique, which is similar to a technique developed by Blelloch [7] for the SCAN model of parallel computation, can also be applied for the BSP model. One novelty of our technique is that two edge arrays are needed; one to represent the graph G and the other to represent the forest which is created in the course of the algorithm. Furthermore, to maintain each data structure the other one is needed.

This section is organized as follows. First, the data structure is described. After that, it is shown how this data structure can be used to resolve concurrent reads and writes, and how the data structure is maintained if a hooking or a shortcutting step is applied. Finally, we describe algorithm BSP-AS and analyze its running time.

Data Structure. We use the following data structures in order to represent the graph G. Each undirected edge $\{v, w\}$ is represented by two directed edges (v, w) and (w, v), which are stored in an array $\mathsf{E}[1..2m]$. All edges incident to a node are stored consecutively in this array (although not necessarily sorted). Hence, the array E can be viewed as an adjacency list representation of the graph G. Every entry $e = \mathsf{E}[i]$ in the array E consists of the following fields: $e.\mathit{Src}$ is the source node of the edge, $e.\mathit{Dest}$ is its destination node, $e.\mathit{Weight}$ is its weight, and $e.\mathit{Forest}$ is a boolean value that indicates if the edge belongs to the minimum spanning tree. Furthermore, we also maintain a field $e.\mathit{Opp}$ for every edge $e = (v, w)$, which is a pointer to the opposite edge $e' = (w, v)$.

In the course of the algorithm, connected components are formed which are represented by a collection of rooted trees. The edges of these trees are stored in an array $\mathsf{F}[1..2n]$. As in the edge array, every forest edge f is represented by two directed edges, and all edges incident to the same node are stored consecutively. Every entry $f = \mathsf{F}[i]$ also consists of the fields $f.\mathit{Src}$, $f.\mathit{Dest}$ and $f.\mathit{Opp}$. In addition, a boolean field $f.\mathit{Parent}$ is used to indicate whether the edge points from a child to its parent or vice versa. Initially, for each node v, there exist two edges (v, v) in F.

A third array $\mathsf{V}[1..n]$ is used to store node information. Every entry $v = \mathsf{V}[i]$ consists of the fields $v.P$ and $v.\mathit{Hook}$. The parent of node u in the collection of rooted trees is given by $\mathsf{V}[u].P$, and $\mathsf{V}[u].\mathit{Hook}$ is an extra field that can be used to exchange information between the arrays E and F (cf. Algorithm 3). All three arrays are distributed evenly among the processors.

Simulating Concurrent Accesses and Maintaining the Data Structures. In algorithm BSP-AS, all concurrent accesses are one of two forms: (a) for each edge (u, v), read or write information about node u, or (b) for each edge (u, v), read or write information about node v. One typical example is when every edge (u, v) wants to know the "parent" of its source node u in the forest, as is needed in the shortcutting step.

Lemma 4 *Every concurrent access of type (a) or type (b) can be simulated in $O(g \cdot (m + n)/p + L + T_{\mathit{prefix}})$ time on the BSP model.*

Proof: Divide the array into n segments consisting of edges with the same source node. The first edge in each segment is called the *segment leader*. Then a concurrent read of type (a) can be realized by the following two steps: (1) each segment leader fetches the requested information from its

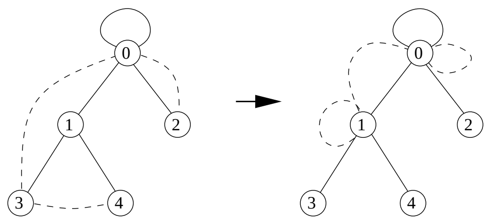

Figure 1: Example graph: Dotted lines are nontree edges and straight lines are tree edges.

source node, (2) using a segmented broadcast, this information is distributed across the segment. A concurrent write of access type (a) as well as access type (b) can be implemented similarly. □

We now show how the arrays F and E are restored in every round of algorithm BSP-AS, so that prior to each round all edges incident to the same node are stored consecutively. In this extended abstract it is only shown how the ordering of the edges in E is maintained when every edge (v, w) is replaced by the edge $(parent(v), parent(w))$. Below, an edge f in the forest array F is called a *child edge* if it points from a node to its child, and a *parent edge* otherwise. Note that both array F and array V are needed in order to maintain array E.

Algorithm 3 Maintaining the edge array E.

(1) In array E, count the number $degree(v)$ of edges incident to each node v. After that, the segment leader associated with node v writes $degree(v)$ into position v of array V.

(2) Every parent edge (v, w) in the forest edge array F reads $degree(v)$ from position v in V, and sends this number to the position of its twin pointer. Now every child edge (v, w) is labeled with $degree(w)$. The label of each parent edge is set to 0.

(3) Perform a prefix sum operation on the values $degree(v)$ in F.

(4) By reversing the actions of Step (2) and (1), every segment leader in E is correctly labeled with its new position. The new indices of the remaining edges can be computed by a segmented parallel prefix with the increment operator (+1).

(5) All edges in E are permuted to their new positions.

- End of Algorithm -

An example is shown in Figure 1 and 2. The runtime of the algorithm described above can be seen to be $O(g \cdot (m + n)/p + L + T_{\text{prefix}})$. Moreover, the time required for one round of algorithm BSP-AS is dominated by the time taken to implement the concurrent accesses and the time needed to maintain the data structures.

Algorithm and Analysis. Now we are ready to describe algorithm BSP-AS.

Algorithm 4 BSP-AS

(1) Simulate the Awerbuch-Shiloach algorithm for $2 \cdot (\log p + \log\log p)$ rounds, as follows.

(1.1) *Star recognition.* For each node v, determine if it belongs to a star. Concurrent reads and writes in this step are avoided by using the F array.

(1.2) *Hooking and Tie breaking.* For each active node u which is a star, determine its minimum edge $\mathsf{E}[i_u]$ and set $\mathsf{E}[i_u].Forest := true$. Tie breaking can be done by sending a message from each minimum edge to its twin pointer. Afterwards, each node u sets $\mathsf{V}[\mathsf{V}[\mathsf{E}[i_u].Src].P].P := \mathsf{V}[\mathsf{E}[i_u].Dest].P$.

(1.3) *Shortcutting.* For each node u which does not belong to a star, set $\mathsf{V}[u].P := \mathsf{V}[\mathsf{V}[u].P].P$. This can be done by first applying shortcutting in the array F, after which every parent edge (v, w) in F updates the parent field of $\mathsf{V}[v]$.

(1.4) *Maintain the edge array* E. For each edge $\mathsf{E}[i]$, get the new component names of its endpoints by setting $\mathsf{E}[i].Src := \mathsf{V}[\mathsf{E}[i].Src].P$ and $\mathsf{E}[i].Dest := \mathsf{V}[\mathsf{E}[i].Dest].P$. This is done using the technique described in Lemma 4. Erase internal edges (edges whose endpoints are the same) and restore the edge ordering by $\mathsf{E}[i].Src$, as explained in Algorithm 3.

(2) Contract every connected component as defined by the forest edges into a single vertex by first applying the Euler tour algorithm of [8], after which every segment leader fetches the new node number.

(3) Every processor locally computes the MSF of the graph defined by the remaining supervertices and the edges stored in its local memory.

(4) Merge the MSFs as in Step (4) of algorithm MST-DENSE.

- End of Algorithm -

The time complexity of the Awerbuch-Shiloach algorithm is calculated by proving that after s rounds the total height of all trees is less than or equal to $(2/3)^s \cdot n$, but we need that the size of each component increases by a constant factor $\alpha > 1$ in every round.

Lemma 5 *After s rounds of* BSP-AS*, the size of each component remaining is at least $2^{s/2}$.*

Proof: By induction on s: The base case $s = 0$ trivially holds. Now suppose it holds for $s - 1$ and consider a component C during iteration s. If C is a star, then it gets hooked and the resulting component has size at least $2 \cdot 2^{(s-1)/2} > 2^{s/2}$. Otherwise, let s' be the previous iteration in which C took part in a hooking step. Note that s' is well-defined because every component gets hooked in the first iteration. Because C did not become a star in

Array E

Idx	0	1	2	3	4	5
Src	3	3	0	0	2	4
Dest	4	0	3	2	0	3
Opp	5	2	1	4	3	0
Degree	2		2		1	1
New idx	0		4		3	2
Pref	0	1	4	5	3	2
New-Src	1	1	1	0	0	0
New-Dst	1	0	1	0	1	0

Array V

Node	0	1	2	3	4
Parent	0	0	0	1	1
Degree	2	0	1	2	1
New idx	4	3	3	0	2

Array F

Idx	0	1	2	3	4	5	6	7	8	9
Src	4	3	1	1	1	0	0	0	0	2
Dest	1	1	3	4	0	0	1	2	0	0
Opp	3	2	1	0	6	8	4	9	5	7
Parent edge	x	x			x	x				x
Degree	1	2			0	2				1
to Opp			2	1			0	1	2	
Scan			0	2			3	3	4	
to Opp	2	0			3	4				3

Figure 2: Example of maintaining the edge array E for the graph depicted in Figure 1.

$s - s'$ iterations, it must have contained a chain of length at least $2^{s-s'}$ (actually, $2^{s-s'}+1$ since it takes $\lceil \log(h-1) \rceil$ shortcutting steps to reduce a tree of h levels into a star). By induction, every component on that chain had size at least $2^{(s'-1)/2}$. It follows that the size of C is at least $2^{s-s'} \cdot 2^{(s'-1)/2} = 2^{s-(s'+1)/2} \geq 2^{s/2}$ since $s \geq s'+1$. □

Thus, after $2 \cdot (\log p + \log\log p)$ rounds, the size of each component is at least $p \cdot \log p$, which implies that at most $n/(p \cdot \log p)$ components remain. At this point, we switch to the last two steps of algorithm MST-DENSE (see Section 4). First, every processor locally computes the MSF of the graph defined by the remaining supervertices and the edges stored in its local memory. Thereupon, the MSFs are merged together as in Step (4) of algorithm MST-DENSE. The total BSP complexity of algorithm BSP-AS is given in the following theorem:

Theorem 5 *Let $m+n \geq p$. The BSP cost of algorithm* BSP-AS *is given by $W = O(\frac{m+n}{p} \cdot \log p)$, $H = O(\frac{m+n}{p} \cdot \log p)$, and $S = O(\log p \cdot \lceil \frac{\log p}{\log((m+n)/p)} \rceil)$.*

7 Towards a Work Optimal Algorithm

The amount of local work BSP-AS performs as well as its communication time is not linear in the problem size. To achieve this, we employ the following observation. Since the purpose of the first phase of MST-SPARSE is to reduce the number of nodes by a factor of k (by finding MST components of size at least k and by contracting each one into a single node), it is sufficient to consider only the k cheapest edges incident to each node in the first round. Similarly, in the second round, each component (of size at least two) needs only $k/2$ other components to hook with. In general, in round i, it is sufficient to consider only the $k/2^i$ cheapest edges incident to each component. Since the number of edges decreases geometrically, the communication time of the sketched algorithm is given by $O(g \cdot (m/p + k \cdot n/p + \log k \cdot n/p))$. By choosing $k = m/n$ runtime $O(g \cdot (m/p))$ is achieved. This idea of growing components only by a bounded factor and therefore not having to consider all edges was originally used by Johnson and Metaxas [19] in their EREW PRAM algorithm. In our algorithm, however, edges are deleted much more aggressively in order to keep the communication requirements low.

There are two problems that need to be resolved. First, edges that remain after each round should be non-internal (i.e., their endpoints should not belong to the same component). However, nodes are unable to identify which of their incident edges are internal, because they do not know a unique component number (this number is not available until a component becomes a star). Therefore, after deleting all but the best $k/2^i$ edges of a node, it could happen that one tree node has only internal incident edges left while the cheapest edge incident to the entire component has been deleted. Thus, this component has to stop hooking (become *inactive*) or otherwise an edge may be misidentified as belonging to the MST. It may seem that a component that becomes inactive soon will not have the required size, but Lemma 6 shows that this is not the case.

A second problem is caused by multiple edges which connect the same components. In algorithm BSP-AS it is impossible to identify these edges but this causes no problems because the cheapest edge is always selected and edges are never erased. However, in the modified algorithm problems will occur if all but the $k/2^i$ cheapest edges are erased and all remaining edges happen to point to the same component. To circumvent this problem we do not erase edges incident to components but edges incident to original nodes. This means that in round i, the best $k/2^i$ edges incident to each original node remain.

Algorithm 5 BSP-AS-SELECT(k)

(1) For each node v, select its k-cheapest edge e_v. All edges incident to node v with weight less than or equal to e_v (as well as their opposite edges) have to stay "alive". All other edges are (temporarily) deleted. The remaining edges are distributed evenly among the processors.

```
(2) Repeat 3 times:
    (2.1) For $i = 1, \ldots, \log k$ do
        (2.1.1) For every original node $v$, select the $k/2^i$-cheapest edge $e_v$. All edges incident to node $v$ with weight less than or equal to $e_v$ (as well as their opposite edges) have to stay alive. All other edges are (temporarily) deleted.
        (2.1.2) Determine which stars become inactive.
        (2.1.3) Execute one round of BSP-AS. Only active stars take part in hooking.
    (2.2) Using list ranking and the Euler tour technique, each connected component is contracted into a single node. All edges deleted in Step (2.1.1) are reactivated.
(3) All edges deleted in Step (1) are reactivated.
```

- End of Algorithm -

Lemma 6
(a) After loop (2.1), each component consists of at least $\min_{1 \le i \le \log k}(\max\{2^{i/2}, k/2^i\})$ *nodes.*
(b) After loop (2.1), each component consists of at least $k^{1/3}$ *nodes.*

Proof: Part (a): Consider an arbitrary component C and assume that it is active for j rounds. In the first j rounds the behavior of C is the same as in algorithm BSP-AS and, thus, Lemma 5 shows that the size of the component is at least $2^{j/2}$. Furthermore, because C becomes inactive in round j, it contains at least one node which has deleted edges by selection (and therefore had a lot of incident edges), and has no incident edges left after the internal edges are deleted in Step (4) of algorithm BSP-AS. Let v be this node. Since v had $k/2^j$ incident edges left prior to round j and these edges originally pointed to different nodes, the component that contains v consists of at least $k/2^j$ nodes.
Part (b) follows from the observation that the minimum of the function given in (a) is assumed for $j = \frac{2}{3}\log k$. □

Thus, by applying loop (2.1) three times and by contracting every connected component induced by the forest edges into a single vertex, each component consists of at least k nodes.

Lemma 7 *Let $k \le p \le n$ and $k = \alpha \cdot m/n \ge \log p$ for some constant $\alpha > 0$. For any constant $\epsilon > 0$, algorithm* BSP-AS-SELECT *determines minimum spanning tree components of size at least k in time $W = O(\frac{m}{p} + \frac{n}{p} \cdot \log p)$, $H = (6+\epsilon) \cdot \frac{m}{p} + o(\frac{m}{p}) + O(\frac{n}{p} \cdot \log p)$, and $S = O(\log p \cdot \lceil \frac{\log p}{\log((m+n)/p)} \rceil)$ with probability at least $1 - p^{-c}$ for any constant $c > 0$. If $m \ge n \log n$ the runtime bound holds with probability at least $1 - n^{-c}$.*

Proof: By Lemma 3(b), selection takes $g \cdot 2 \cdot m/p + O(m/p + T_{\text{prefix}})$ time. In this case, however, multiple independent selections have to be performed on groups of consecutively numbered processors. Since $k \ge \log p$ is assumed, the runtime bound still holds with high probability. It also takes $g \cdot 2 \cdot m/p + O(m/p + T_{\text{prefix}})$ time to notify every twin edge that it has to stay alive, as well as to redistribute the surviving edges evenly among the processors. The total BSP cost of Step (1) is therefore given by $g \cdot 6 \cdot m/p + O(m/p + T_{\text{prefix}})$.

In order to reduce the number of times every edge has to be communicated, the following technique is applied. Let c be a constant such that the communication time in Step (2) of BSP-AS-SELECT is at most $c \cdot m/p + O((n/p) \cdot \log p)$. We set $k' = \epsilon \cdot m/(c \cdot n)$ and call BSP-AS-SELECT(k'). The communication time incurred in Step (2) is then given by $\epsilon \cdot m/p + O((n/p) \cdot \log p)$. □

8 Parallel Verification

We are given a forest F with at most $n' \le n/k \le n/\log p$ nodes and a set of $m' \le 2 \cdot m$ non-forest edges. For every non-forest edge $e = (v, w)$, it needs to be verified whether $w(e)$ is greater than the weight of the heaviest edge on the path between v and w in F (∞ if v and w are not connected).

Our verification algorithm, called BSP-VERIFY, is derived from the EREW PRAM algorithm presented in [23]. It improves upon the algorithm given in [23] by using fewer supersteps. Furthermore, BSP-VERIFY employs a simple load balancing technique in order to minimize the number of times the edges have to be communicated. For a detailed description of the algorithm, the reader is referred to [2].

Lemma 8 *Let $n \ge p$ and $m \ge n \cdot \log p$. The BSP cost of* BSP-VERIFY *is given by $W = O(\frac{m}{p} + \frac{n}{p} \cdot \log p)$, $H = 6 \cdot \frac{m}{p} + o(\frac{m}{p}) + O(\frac{n}{p} \cdot \log p)$, and $S = O(\log p)$.*

References

[1] M. Adler, J. W. Byers, and R. M. Karp. Parallel sorting with limited bandwidth. In *Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 129–136, 1995.

[2] M. Adler, W. Dittrich, B. Juurlink, M. Kutyłowski, and I. Rieping. Towards an Efficient Minimum Spanning Tree Algorithm. Technical Report TR-RSFB-97-052, University of Paderborn, Computer Science Dept., 1997.

[3] B. Awerbuch and Y. Shiloach. New connectivity and MSF algorithms for Shuffle-Exchange network and PRAM. *IEEE Transactions on Computers*, C-36(10):1258–1263, 1987.

[4] A. Bäumker and W. Dittrich. Fully Dynamic Search Trees for an Extension of the BSP Model. In *Annual ACM Symposium on Parallel Algorithms and Architecture*, 1996.

[5] A. Bäumker, W. Dittrich, and F. Meyer auf der Heide. Truly efficient parallel algorithms: c-optimal multisearch for an extension of the BSP model. In *Proc. of the Annual European Symposium on Algorithms*, 1995.

[6] A. Bäumker, W. Dittrich, F. Meyer auf der Heide, and I. Rieping. Realistic parallel algorithms: Priority queue operations and selection for the BSP* model. In *Proc. of the Annual International EURO-PAR Conference, LNCS*, 1996.

[7] G. E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, pages 1526–1538, 1989.

[8] E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *Annual International Colloquium on Automata, Languages and Programming (ICALP)*, 1997.

[9] K. Chong. Finding Minimum Spanning Trees on the EREW PRAM. Manuscript, 1997.

[10] R. Cole, P. N. Klein, and R. E. Tarjan. Finding minimum spanning forests in logarithmic time and linear work using random sampling. In *Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 243–250. ACM, 1996.

[11] D. Culler, R. Karp, D. Patterson, A. Sahay, K. S. an d E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proc. 4th Symp. on Principles and Practice of Parallel Programming*, pages 1–12. ACM SIGPLAN, May 1993.

[12] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Annual ACM Symposium on Computational Geometry*, 1993.

[13] F. Dehne and S. Götz. Efficient Parallel Minimum Spanning Tree Algorithms for Coarse Grained Multicomputers and BSP. Manuscript, 1997.

[14] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP model. In *Annual ACM Symposium on Parallel Algorithms and Architecture*, 1996.

[15] A. V. Gerbessiotis and L. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22:251–267, 1994.

[16] P. B. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation? In *Annual ACM Symposium on Parallel Algorithms and Architecture*, 1997.

[17] M. T. Goodrich. Communication-efficient parallel sorting. In *Proc. of the ACM Symposium on Theory of Computing*, 1996.

[18] M. T. Goodrich. Randomized fully-scalable BSP techniques for multi-searching and convex hull construction. In *ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[19] D. B. Johnson and P. Metaxas. A parallel algorithm for computing minimum spanning trees. *Journal of Algorithms*, 19:383–410, 1995.

[20] B. H. H. Juurlink and H. A. G. Wijshoff. Communication Primitives for BSP computers. *Information Processing Letters*, 58(6):303–310, 1996.

[21] D. Karger, P. Klein, and R. Tarjan. A Randomized Linear-Time Algorithm to find Minimum Spanning Trees. *Journal of the ACM*, 42:321–328, 1995.

[22] D. R. Karger. Random sampling for minimum spanning trees and other optimization problems. In *Annual ACM Symposium on Foundations of Computer Science*, 1993.

[23] V. King, C. Poon, V. Ramachandran, and S. Sinha. An optimal EREW PRAM algorithm for minimum spanning tree verification. *Information Processing Letters*, 62(3):153–159, 1997.

[24] C. Poon and V. Ramachandran. A randomized linear work EREW PRAM algorithm to find a minimum spanning forest. In *International Symposium on Algorithms and Computation (ISAAC)*, LNCS 1350, pages 212–222, 1997.

[25] J. Reif, editor. *Synthesis of parallel algorithms*. Morgan Kaufmann Publishers, 1993.

[26] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8), Aug. 1990.

[27] A. Yao. Lower bounds by probabilistic arguments. In 24th *IEEE Symposium on Foundations of Computer Science*, pages 420–428, 1983.

[28] A. C. Yao. Some complexity questions related to distributive computing. In *Proc. of the ACM Symposium on Theory of Computing*, pages 209–213, 1979.