An Extended Abstract appears in Proc. ISAAC'97, LNCS 1350, pp. 212-222.

A Randomized Linear Work EREW PRAM Algorithm to Find a Minimum Spanning Forest*

Chung Keung Poon[1] and Vijaya Ramachandran[2]

[1] Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong. Email: ckpoon@cs.cityu.edu.hk.
[2] Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, USA. Email: vlr@cs.utexas.edu.

Abstract. We present a randomized EREW PRAM algorithm to find a minimum spanning forest in a weighted undirected graph. On an n-vertex graph the algorithm runs in $o((\log n)^{1+\epsilon})$ expected time for any $\epsilon > 0$ and performs linear expected work. This is the first linear work, polylog time algorithm on the EREW PRAM for this problem. This also gives parallel algorithms that perform expected linear work on two general-purpose models of parallel computation – the QSM and the BSP.

1 Introduction

The design of efficient algorithms to find a minimum spanning forest (MSF) in a weighted undirected graph is a fundamental problem that has received much attention. There have been many algorithms designed for the MSF problem that run in close to linear time (see, e.g., [CLR91]). Recently a randomized linear-time algorithm for this problem was presented in [KKT95]. Based on this work [CKT94] presented a randomized parallel algorithm on the CRCW PRAM which runs in $O(2^{\log^* n} \log n)$ expected time while performing linear work. The expected time was later improved to logarithmic by [CKT96].

In this paper we consider the design of a linear-work parallel algorithm on a more restricted model of parallel computation – the EREW PRAM. A major motivation for considering the EREW PRAM is the adaptability of an algorithm developed on this model to more realistic parallel computation models. In particular, the EREW PRAM (and the more powerful QRQW PRAM [GMR94]) are special cases of the Queuing Shared Memory (QSM) model [GMR97], which is a general-purpose shared-memory model of parallel computation. It is shown in [GMR97] that the QSM (and hence the EREW PRAM) has a randomized work-preserving emulation with small slow-down on the Bulk Synchronous Parallel (BSP) computer [Val90], which is a general-purpose distributed-memory model of parallel computation. However, these results are not known to hold for the CRCW PRAM. Thus algorithms designed on an EREW PRAM (or the QRQW PRAM) have more applicability than CRCW PRAM algorithms.

* This research was supported in part by NSF grant CCR/GER-90-23059 and Texas Advanced Research Program Grant 003658386.

We present a randomized algorithm to find a minimum spanning forest on an EREW PRAM that performs linear expected work and runs in expected time $O(\log n \cdot \log\log n \cdot 2^{\log^* n})$. Here n is the number of vertices in the input graph. This is the first parallel algorithm for this problem on the EREW PRAM that performs linear work. The running time of our algorithm is only slightly super-logarithmic – by a factor that is $o((\log\log n)^{1+\epsilon})$, for any $\epsilon > 0$. We also present a refinement of this result that gives a time, edge-density trade-off while maintaining linear work, and we describe the results obtained by mapping our algorithm on to the QSM and the BSP.

Prior to our work, the best results known for the EREW PRAM were a deterministic algorithm by Chong [Cho96] which runs in $O(\log n \log\log n)$ time and performs $O(m \log n \log\log n)$ work, and a randomized algorithm reported in [Kar95] which, with high probability, runs in $O(\log n)$ time and performs $O(m + n^{1+\epsilon})$ work for any constant $\epsilon > 0$.

2 Outline of the algorithm

Let G be a graph with n vertices and m edges. We assume that edges in G have distinct weights, so the graph has a unique minimum spanning forest (MSF). We also assume that G does not contain isolated vertices, so we have $m \geq \lceil n/2 \rceil$.

Our algorithm can be viewed as a parallelization of [KKT95] and has a recursive structure similar to that of [CKT94]. As in these two algorithms, our algorithm makes use of the following well known properties (see [Tar83]).

Cycle Property: For any cycle C in a graph, the heaviest edge in C does not appear in the MSF.

Cut Property: For any proper nonempty subset X of the vertices, the lightest edge with exactly one endpoint in X belongs to the MSF.

To find the MSF of G, the algorithm first identifies a set, E, of edges that belong to the MSF of G. It then *contracts* each component induced by E into a single vertex. This reduces the number of (non-isolated) vertices by a suitable factor k. The algorithm then computes the MSF in the contracted graph G_c. By the cut property, the set E together with the MSF of G_c will form the MSF of G.

Although the number of vertices in the contracted graph G_c is reduced by a factor k, the number of edges may not be reduced significantly, even if internal and parallel edges are removed. We can reduce the number of edges in G_c by a factor $\lambda = \sqrt{k}$ by using a random sampling technique first employed for the MSF problem by [Kar93, KKT95]. This method samples the edges in G_c independently with probability $p = 1/\lambda$ and then recursively computes the MSF, F, of the sampled graph, which has on average $O(mp) = O(m/\lambda)$ edges. This is the first recursive call. Although F is probably not the MSF of G_c, it can be used to identify some edges in G_c that are not in the MSF of G_c. Following [KKT95], an edge in G_c is said to be *F-heavy* if it forms a cycle when added to F and is heavier than every edge in that cycle. Edges in G_c which are not F-heavy are said to be *F-light*. By the cycle property, F-heavy edges cannot be in the MSF of G_c. Thus, to compute the MSF we need to consider only the F-light edges. By the sampling lemma (Lemma 2.1) of [KKT95], the expected number of F-light edges in G_c is at most $O((n/k)/p) = O(n/\lambda)$.

By *filtering* G_c with F, i.e., removing the F-heavy edges in G_c, we obtain a graph G' that has $O(n/k)$ vertices and $O(n/\sqrt{k})$ edges. The algorithm then recursively computes the MSF of G'. This is the second recursive call. Since G' depends on the output, F, of the first recursive call, it does not seem possible to execute the two calls in parallel. Consequently, the above algorithm runs in $\Omega(2^l)$ time where l is the number of recursion levels – even if the maximum time spent in the local computation of each recursive call is constant.

By setting k to a sufficiently large constant and recursing until the size of the graph is reduced to a constant, one can obtain a linear work algorithm [KKT95]. However, there will be $l = \Theta(\log_k n) = \Theta(\log n)$ levels of recursion and hence the parallel running time is $\Omega(n^\epsilon)$ for some $\epsilon > 0$. In [CKT94] a linear-work CRCW PRAM algorithm was presented by setting the reduction factor of a recursive call to the exponential of that of its parent call. With this reduction factor, the number of recursion levels was reduced to $O(\log^* m)$. The algorithm in [CKT94] requires a randomized CRCW PRAM in order to achieve logarithmic time in the local computation of each recursive call, and has an overall running time of $O(\log n \cdot 2^{\log^* m})$.

In this paper we present a randomized linear-work parallel algorithm on the more restricted EREW PRAM model. As in [CKT94] our algorithm has $O(\log^* m)$ levels of recursion, but we perform each recursive call on an EREW PRAM in $O(\log n \log\log n)$ time, by designing an appropriate contraction procedure, and by using the $O(\log n)$ time, linear work EREW algorithm of [KPRS97] for the filtering step that detects F-heavy edges. Moreover, we stop the recursion when the size of the graph is reduced to a polylog factor of the original one. At this point, we switch to the deterministic algorithm of Chong [Cho96] which runs in $O(\log n' \log\log n')$ time and $O(m' \log n' \log\log n')$ work on a graph with n' vertices and m' edges. Since we apply Chong's algorithm on a sufficiently small input, we show that the work on all recursive calls to Chong's algorithm remains linear with respect to m, the size of the input graph G. Consequently, our overall algorithm has linear work. The expected running time of our algorithm is $O(\log n \cdot \log\log n \cdot 2^{\log^* m})$. We also present a refinement of our algorithm that runs in expected time $O(\log n \cdot \log\log n \cdot 2^{\log^* f})$, where $f = 1 + (\log n \log\log n \cdot \sqrt{n/m})$, while performing linear expected work. Finally, we describe the adaptation of our algorithm to the QSM and the BSP.

3 Detailed Algorithm and Analysis

We first introduce some notations. Denote by $|H|$ the number of edges in a graph H. Define $\log^* x$ as the minimum i such that $\log^{(i)} x \leq 2$ where $\log^{(0)} x = x$ and $\log^{(i)} x = \log(\log^{(i-1)} x)$ for integer $i \geq 1$. Given a graph G and an edge $e = (u, v)$ in G, the *contraction of* e *in* G results in the graph H that is obtained from G by deleting edge e, combining u and v into a single vertex, and removing isolated vertices and internal edges, while allowing multiple edges to remain. Given a subset of edges S in G, *the contraction of* G *with respect to* S is the graph obtained through the contraction of each edge in S.

Here are several parameters used by our algorithm. Recall that our input graph G has n vertices and m edges. We define the integer parameters l and $k_1, k_2, \ldots, k_l$ as follows. Set $k_1 = \lfloor \log m \rfloor$, $k_i = \lfloor \log k_{i-1} \rfloor$ for $1 < i \leq l$ and l as the smallest integer such that $k_l \leq 2$. The proof of the following claim is straightforward.

Claim 1. *Let l and $k_i, 1 \leq i \leq l$ be as defined above. Then, for m sufficiently large,*

1. $l \leq \log^* m$;
2. $\log k_{i-1} \leq k_i + 1$ *for* $2 \leq i \leq l$;
3. $k_i \geq 2^{l-i}$ *for* $1 \leq i \leq l$.

The pseudo-code for our algorithm, *FindMSF*(), is shown below. To find the MSF of G, we will call $FindMSF(G, l, F)$ where l is set as above and F is the output (i.e., the edges in the MSF of G).

```
Algorithm FindMSF($H, i, F$)
var  $H_c$, $H_s$, $F_s$, $H'$, $F'$
1   if $i = 1$ then
        apply Chong's algorithm on $H$, and
        set $F$ to be the set of edges found
    else
2       call Contract($H, k_{i-1}^6, H_c, F$)
3       sample the edges in $H_c$ with probability $1/k_{i-1}^3$ to form the graph $H_s$
4       call FindMSF($H_s, i-1, F_s$)
5       call Filter($H_c, F_s, H'$)
6       call FindMSF($H', i-1, F'$)
7       $F := F \cup F'$
```

Data structure for graph representation :
We will represent the input graph G and the output F using the adjacency lists data structure. More precisely, each vertex u in G has a doubly-linked list containing one record for each of its incident edges. The record for edge $e = (u, v)$ in the adjacency list of vertex u contains the following 6 fields:

1. $weight(e)$: edge weight
2. $endpt1(e)$: the vertex name of itself (i.e., u)
3. $endpt2(e)$: the vertex name of the other endpoint (i.e., v)
4. $endpt2\_ptr(e)$: a pointer to the edge record of $e = (u, v)$ on the adjacency list of v
5. $mark(e)$
6. $oldgraph\_ptr(e)$

The first 4 fields are for storing the input graph G. The fifth field is used to store which edges are in F when the algorithm finished. By standard techniques (using prefix sums), one can convert F in such a representation into one in the adjacency lists representation in $O(\log n)$ time and $O(|G|)$ work.

We now explain the use of the last field. In addition to the input graph, there are other graphs generated during subsequent subroutine calls or recursive calls. They will also be represented using the above adjacency lists data structure. The last field in each edge record of such a graph contains a pointer to the corresponding edge record in the graph from which it is generated. This facilitates the passing of results from recursive calls.

The algorithm makes calls to two procedures:

- The procedure *Contract*(). The procedure call *Contract*(H, k, H_c, F) produces a set of edges F and a multi-graph H_c which is the contraction of H with respect to F such that (1) F is a subset of the MSF of H, and (2) H_c has at most $(1/k)$ times the number of vertices in H. The set F is represented by marking the *mark* field on the adjacency lists of H. Note that H_c is a multi-graph without any isolated vertices. The algorithm for *Contract* is given in the next section, together with an analysis that shows that it runs in $O(\log n \log k)$ time and $O(|H| \log k)$ work on a deterministic EREW PRAM.
- The procedure *Filter*(). The procedure call *Filter*(H, F, H'), in which F is a forest of multi-graph H, returns a subgraph H' of H which contains the vertex set of H and the F-light edges in H. It runs in $O(\log n)$ time and $O(|H|)$ work on a deterministic EREW PRAM. The filtering step is described in section 5.

The algorithm also makes recursive calls to itself. Note that the input graphs for the recursive calls may be multigraphs, i.e., may contain multiple edges. The following lemma gives bounds on the expected number of vertices and edges in these graphs in terms of n and m, the number of vertices and edges in the original input G.

Claim 2. *For any integer i where $1 \leq i \leq l$, any call to FindMSF(H, i, F) resulting from the initial call to FindMSF(G, l, F) satisfies the following: (1) the expected number of vertices in H is at most $O(n/k_i^6)$ and (2) the expected number of edges in H is at most $O(m/k_i^3)$.*

Proof. We will prove the claim by induction on i.

(Base Case) The claim is true for $i = l$ since $1 \leq k_l \leq 2$ by the definition of l and by Claim 1.

(Induction Step) Assume that the claim is true for $i = j$ and consider the call *FindMSF*(H, j, F). After contraction in step 2, the number of vertices in H_c is $O(n/k_{j-1}^6)$ by Claim 5. Note that H_c may contain multiple edges. However, the total number of edges in H_c is no more than that in H. Hence the sampling in step 3 produces a graph H_s with $O(m/k_{j-1}^3)$ expected edges. Consequently, the recursive call to *FindMSF*$(H_s, j-1, F)$ in step 4 will satisfy the claim.

By Lemma 2.1 in [KKT95] and Claim 7, expected number of edges in H' is $O((n/k_{j-1}^6) \times k_{j-1}^3) = O(m/k_{j-1}^3)$. Again, this bound is true even though H_c may contain multiple edges. Hence the recursive call to *FindMSF*$(H', j-1, F)$ in step 6 also satisfies the claim. This completes the induction step and also the proof of the claim. □

Claim 3. *The call FindMSF*(G, l, F) *runs in* $O(2^l \cdot \log n \cdot \log\log n)$ *expected time and* $O(m)$ *expected work on a randomized EREW PRAM.*

Proof. We assume there are $m/(2^l \log n \log\log n)$ processors available. We first analyze the total expected time required. The initial call to *FindMSF*(G, l, F) will generate 2 recursive calls to *FindMSF*$(H, l-1, F)$, 2^2 calls to *FindMSF*$(H, l-2, F)$, 2^3 calls to *FindMSF*$(H, l-3, F)$, $\ldots$, and 2^{l-1} calls to *FindMSF*$(H, 1, F)$. Since the local computations in these recursive calls are performed in sequential order, the running time of *FindMSF*(G, l, F) is the sum of the time for the local computation in each recursive call.

Consider each call to *FindMSF*$(H, 1, F)$. By Claim 2, the expected number of edges in H is $|H| = O(m/k_1^3)$ $=O(m/(\log m)^3)$. Applying Chong's algorithm with $|H| \leq m/(2^l \log n \log\log n)$ processors takes $O(\log n \log\log n)$ expected time. Hence $O(2^l \log n \log\log n)$ expected time suffices for all recursive calls to *FindMSF*$(H, 1, F)$ in total.

Consider the local computation in a call to *FindMSF*(H, i, F) for which $i > 1$. By Claim 2, $|H| = O(m/k_i^3)$. Suppose there are $O(|H|/\log n) = O(m/(k_i^3 \log n))$ processors. By Claim 6 and 7, the expected time required by *Contract* and *Filter* (in step 2 and 5 respectively) is $O(\log n \log k_{i-1} + \log n) = O(\log n \log k_{i-1})$. In step 3, sampling the edges in H_c and compacting the sampled edges to form H_s requires $O(\log n)$ time. Since we have $m/(2^l \log n \log\log n)$ processors, the expected time is $O(\log n \log k_{i-1} \cdot \lceil \frac{m/(k_i^3 \log n)}{m/(2^l \log n \log\log n)} \rceil) = O(\log n \log k_{i-1} \cdot \lceil \frac{2^l \log\log n}{k_i^3} \rceil)$. By Claim 1 $\log k_{i-1} \leq k_i + 1$ and $k_i \geq 2^{l-i}$. Hence for $i \geq 3$, the expected time for *Contract* and *Filter* is $O((2^l/k_i^2) \log n \log\log n) = O(2^{2i-l} \log n \log\log n)$. For $i = 2$, the number of processors available, $m/(2^l \log n \log\log n)$, is $\Omega(m/({k_2}^3 \log n))$, and the expected time thus is clearly $O(\log n \log\log n)$. Hence the total expected time required locally in all the calls to *FindMSF*(H, i, F) for which $i > 1$ is $O(\sum_{i=3}^{l} 2^{l-i} \cdot 2^{2i-l} \log n \log\log n + 2^{l-2} \log n \log\log n) = O(2^l \log n \log\log n)$. Consequently *FindMSF*$(G, l, F)$ takes $O(2^l \cdot \log n \log\log n)$ expected time and hence $O(m)$ expected work. □

Claim 4. *The algorithm FindMSF*(H, l, F) *correctly computes the MSF of* H.

Proof. We will prove the correctness by induction on l.

(Base Case) When $l = 1$, Chong's algorithm computes the MSF of H.

(Induction Step) When $l > 1$, the procedure *Contract* identifies a subset of edges in the MSF of H by Claim 5. By induction hypothesis, the recursive call *FindMSF*$(H_s, l-1, F)$ returns the MSF F of the sampled graph, H_s, of H_c. By Claim 7 and the cycle property, *Filter* only removes edges of H_c which are not in the MSF of H_c. By the induction hypothesis again, the recursive call

FindMSF$(H', l-1, F)$ returns the MSF of H' which is the same as the MSF of H_c. Finally, by the cut property the combined set of edges obtained in steps 2 and 6 forms the MSF of H. □

4 The Contraction Procedure

The purpose of the procedure call *Contract*(H, k, H_c, F) is to identify a set of edges F and to produce H_c, the contracted graph of H with respect to F such that (1) F is a subset of the edges in the MSF of H, and (2) H_c has at most $(1/k)$ times the number of vertices in H. We assume H to have distinct edge weights but allow multiple edges.

The pseudo-code of *Contract*() is shown below. The procedure allocates two local variables, $status[u]$ and $parent[u]$, for each vertex u in H. For each edge e in F, $mark(e)$ is set to 1 in the algorithm below.

```
Procedure $Contract(H, k, H_c, F)$
var  $status[], parent[]$
0  set $H_c := H$
   set $oldgraph\_ptr$ of each edge in $H_c$ to the corresponding edge in $H$.
   (Remark: All the operations below are applied on $H_c$ unless otherwise stated.)
1  pfor each vertex $u$ do
       if $u$ is isolated then
           $status[u] := done$
       else
           $status[u] := active$
       $parent[u] := u$
2  repeat $\log k$ times
   a)  pfor each $active$ vertex $u$ do
           find its minimum weight incident edge $e = (u, v)$
           set $parent[u] := v$
           broadcast $parent[u]$ to all incident edges of $u$
           if $parent[v] = u$ and $u < v$ then
               $status[u] := root$
           else
               set $mark(e) := 1$ for the edge $e = (u, v)$
                 on $u$'s and $v$'s adjacency lists in $H$
   b)  pfor each $active$ vertex $u$ do
           plug its adjacency list into $parent[u]$'s adjacency list
           $status[u] := done$
   c)  pfor each $root$ vertex $u$ do
           set $endpt1(e) = u$ for each incident edge $e$ on $u$'s adjacency list
           set $endpt2(e) = u$ for each edge $e = (u, v)$ on
               $v$'s adjacency list
           remove internal edges
           if $u$ is isolated then
```

```
            $status[u) := done$
        else
            $status[u) := active$
3   Remove all $done$ vertices in $H_c$
4   Place all edges $e$ with $mark(e) = 1$ in $F$
```

Claim 5. *Let k be a positive integer, and H be a multi-graph with distinct edge weights. The procedure call Contract(H, k, H_c, F) produces a set of edges F and a multi-graph H_c which is the contraction of H with respect to F such that*

1. *F is a subset of the MSF of H, and*
2. *H_c has at most n/k vertices, where n is the number of vertices in H.*

Proof. In each iteration of step 2a, the graph formed by the parent pointers is a collection of rooted directed trees in which the edges in each tree point from children to their parent, and with an outgoing edge from the root to one of its children. Thus, after each iteration of step 2, each root contains the concatenation of the adjacency lists of all vertices in its tree, with edges internal to the tree removed, and all remaining edges re-labeled to reflect their new endpoints (the roots of the two trees containing their original endpoints).

In each iteration of step 2, the parent pointers are set using a *Borůvka* step [Bor26], and by the cut property, the corresponding edges are in the MSF for H [KKT95]. Thus F is a subset of the MSF for H. In each iteration of step 2a, each active vertex will hook to another active vertex. Done vertices are either isolated vertices or have given up all their edges to their roots in the previous iteration. Hence each rooted directed tree defined by the parent pointers in the current graph contains at least two active vertices and at most one of them remains active at the end of step 2c. This means the number of active vertices reduces by a factor of at least two in each iteration of step 2. After $\log k$ iterations the number of active vertices is reduced by a factor of at least k. Since the vertex set of H_c is the set of active vertices at the end of step 2, the number of vertices in H_c is at most n/k. □

Claim 6. *The procedure call Contract(H, k, H_C, F) runs in time $O(\log n \log k)$ and performs $O(|H| \log k)$ work on a deterministic EREW PRAM.*

Proof. Step 1 requires $O(1)$ time and $O(|H|)$ work. In each iteration of step 2, step 2a takes $O(\log n)$ time and $O(|H|)$ work, step 2b takes $O(1)$ time and $O(|H|)$ work using the *edge plugging* technique of Johnson and Metaxes [JM97], and step 2c takes $O(\log |H|) = O(\log n)$ time and $O(|H|)$ work (this step is performed by broadcasting the name of the root to all elements in the newly-formed adjacency list, relabeling each edge by its new endpoints and then removing those edges whose two endpoints are the same). Thus over all iterations, step 2 requires $O(\log n \log k)$ time and $O(|H| \log k)$ work. Step 3 takes constant time and $O(|H|)$ work. Hence the whole procedure requires $O(\log n \log k)$ time and $O(|H| \log k)$ work. □

5 The Filtering Procedure

Given a multi-graph H with distinct edge weights and a forest F for H, the procedure *Filter*(H, F, H') removes all F-heavy edges in H, i.e., it removes each edge $e = (u, v)$ in H whose weight is greater than the weight of any edge on the path between u and v in F, and returns the 'filtered' graph in H'.

We adapt the MSF verification algorithm in [KPRS97] to identify and remove the F-heavy edges. One method used to avoid concurrent reads in the algorithm in [KPRS97] is to convert the graph so that all endpoints of nontree edges are distinct. This is done using a scheme of [Ram96] that transforms each rooted tree in F by appending a chain to each vertex, with one copy of the vertex for each nontree edge incident on it. It is not difficult to see that the least common ancestor of an edge is unaltered by this transformation. The same scheme when applied to the multigraph H will convert it into a simple graph in which all nontree edges have distinct endpoints.

The algorithm in [KPRS97] can now be adapted in a straightforward way to identify F-heavy edges. Although that algorithm only determines whether or not there is an F-heavy edge in the graph, it is straightforward to modify it to identify all F-heavy edges within the same time and work bounds. This leads to the following claim:

Claim 7. *Let F be a forest of a multi-graph H with distinct edge weights. The procedure call Filter(H, F, H') returns a graph H' in which the F-heavy edges of H are deleted and it runs in $O(\log n)$ time and $O(|H|)$ work on a deterministic EREW PRAM.*

6 Some Extensions

In this section, we describe another algorithm which computes the MSF in $O(2^{\log^* f} \log n \log\log n)$ expected time and $O(m)$ expected work on an EREW PRAM, where $f = 1 + (\log n \cdot \log\log n \cdot \sqrt{n/m})$. Note that if $m = \Omega(n(\log n \log\log n)^2)$, then $f = O(1)$. Otherwise, $f = O(\log n \log\log n)$. When $m = \Omega(n(\log n \log\log n)^2)$, this algorithm has the same performance as an algorithm in [Kar95] for the CREW PRAM, and it matches the time of (and performs less work than) Chong's algorithm. For m not much smaller than $n(\log n \log\log n)^2$, this algorithm runs faster than *FindMSF*, and it is no worse than it in any case. For all edge densities this algorithm performs linear work.

We first describe a randomized EREW PRAM algorithm which computes the MSF in expected time $O(\log n \log\log n)$ and expected work $O(mf)$. The algorithm can be viewed as a combination (and generalization) of a CREW PRAM MSF algorithm in [Kar95] and the EREW PRAM MSF verification algorithm in [KPRS97]. Hence we will call it the K-KPRS algorithm. The generalization of the CREW PRAM algorithm of [Kar95] lies in the sampling probability used in K-KPRS – $f/(\log n \log\log n)$ – in place of $1/(\log n \log\log n)$ used in [Kar95] (which is independent of the edge density). Algorithm K-KPRS requires $mf/(\log n \log\log n)$ processors and has the following steps.

1. Sample the edges in G independently with probability $f/(\log n \log\log n)$. We expect the sampled graph G_s to have $mf/(\log n \log\log n)$ edges.
2. Compute the MSF, F, of G_s using Chong's algorithm. This takes $O(\log n \log\log n)$ time.
3. Apply *filter* on G and F. This takes $O(\log n \lceil \log\log n/f \rceil) = O(\log n \log\log n)$ time. Let G' be the filtered graph.
4. Compute the MSF of G' using Chong's algorithm. The expected number of edges in G' is $n \log n \log\log n/f$. One can easily check that this is $O(mf/(\log n \log\log n))$ by considering the two cases: $m = \Omega(n(\log n \log\log n)^2)$ and $m = o(n(\log n \log\log n)^2)$.

Hence the K-KPRS algorithm spends $O(\log n \log\log n)$ expected time which in turn implies that it performs at most $O(mf)$ expected work.

Now we use algorithm K-KPRS in place of Chong's algorithm in the base case of our *FindMSF* algorithm and call with parameter $\log^* f$ instead of $\log^* m$. Hence the new algorithm has the stated time and work bounds as given in the claim below.

Claim 8. *The MSF of an n-vertex, m-edge, weighted graph can be computed in $O(2^{\log^* f} \log n \log\log n)$ expected time and $O(m)$ expected work on an EREW PRAM, where $f = 1 + (\log n \log\log n \cdot \sqrt{n/m})$. If $m = \Omega(n(\log n \log\log n)^2)$ the MSF can be computed in $O(\log n \log\log n)$ expected time and $O(m)$ expected work on an EREW PRAM.*

7 Adaptation to QSM and BSP Models

The QSM model [GMR97] and the BSP model [Val90] are general-purpose models of parallel computation that take into account some of the important features of real parallel machines that are not reflected in the PRAM model. The QSM is a shared-memory model with a *gap* parameter g for access to global memory. The BSP is a distributed memory model that consists of processor-memory units interconnected by a general-purpose interconnection network whose performance is parameterized by a gap parameter g as well as a *periodicity* parameter L. For a precise definition of the two models, see [Val90, GMR97].

It is straightforward to see ([GMR97]) that any EREW PRAM algorithm that runs in time t and work w is a QSM algorithm that runs in time $g \cdot t$ and work $g \cdot w$. Also, it is shown in [GMR97] that any QSM algorithm that needs to access r distinct memory locations must perform work $\Omega(g \cdot r)$.

Let $T(n,m) = O(\log n \log\log n 2^{\log^* f})$, where $f = 1 + (\log n \log\log n \cdot \sqrt{n/m})$. Thus $T(n,m)$ is the expected running time of $FindMSF(G,l,F)$ on an EREW PRAM, when G is a graph with n nodes and m edges. Based on the results stated above on mapping an EREW PRAM algorithm on to the QSM we have the following Claim.

Claim 9. *Let $G = (V,E)$ be a graph on n vertices and m nodes with distinct weights on edges. A minimum spanning forest for G can be computed on the*

QSM in expected time $O(g \cdot T(n,m))$ and expected work $O(g \cdot (n+m))$. The work bound is optimal.

A randomized work-preserving emulation of the QSM on the BSP is presented in [GMR97] (see also [Ram97]) with the following performance.

Claim 10. *([GMR97]) An algorithm that runs in time t' on a p'-processor QSM with gap parameter g can be emulated on a p-processor BSP with gap parameter g and periodicity parameter L in time $t = O(t' \cdot (p'/p))$ w.h.p. provided $p \leq \frac{p'}{(L/g)+g\log p}$, and t' is bounded by a polynomial in p.*

Using the above two claims, we obtain the following result.

Claim 11. *Let $G = (V,E)$ be a graph on n vertices and m nodes with distinct weights on edges. A minimum spanning forest for G can be computed on a BSP with expected work $O(g \cdot (n+m))$ and expected time $O(g \cdot T(n,m) \cdot ((L/g)+g\log p))$.*

Proof. Let $E[T_{BSP}]$ be the expected running time of the MSF algorithm on the BSP with $p = \frac{n+m}{T(n,m)\cdot((L/g)+g\log p)}$ processors. By Claim 9 and 10 $E[T_{BSP}]$ is bounded by $E[T_{BSP}] \leq (1-1/n^c) \cdot O(g \cdot T(n,m) \cdot ((L/g)+g\log p)) \; + \; (1/n^c) \cdot (g+L) \cdot O(m \log n)$ where c is a constant under our control. In deriving the above expression it was assumed that if the algorithm exceeds the time bound in Claim 11, the MSF is computed sequentially on one BSP processor using a standard sequential algorithm such as Kruskal's algorithm (see, e.g., [CLR91]). By choosing c sufficiently large ($c > 2 + \frac{\log(g+L)}{\log n}$ should suffice) the second term can be made smaller than the first term, resulting in the desired result. $\square$

Soon after hearing of our result, [DJR97] announced some results for finding an MSF on the BSP. We have also been informed of recent independent work by [DG97] on BSP algorithms for the MSF problem. Both of these results are for minimizing the number of 'supersteps' in the BSP computation, and they perform super-linear work on a general input graph. In contrast, our parallel algorithms for finding an MSF on the QSM and the BSP perform expected linear work (i.e., $O(g \cdot (n+m))$ work) on any input graph.

Acknowledgment

The authors would like to thank Santanu Sinha for discussions on this problem.

References

[Bor26] O. Boruvka. O jistem problemu minimalnim. *Praca Moravske Prirodovedecke Spolecnosti*, 3:37–58, 1926. In Czech.

[Cho96] K. W. Chong. Finding minimum spanning trees on the EREW PRAM. In *Proceedings of the 1996 International Conference on Algorithms*, pages 7–14, Taiwan, 1996.

[CKT94] R. Cole, P.N. Klein, and R.E. Tarjan. A linear-work parallel algorithm for finding minimum spanning trees. In *Proceedings of the 1994 ACM Symposium on Parallel Algorithms and Architectures*, pages 11–15, 1994.

[CKT96] R. Cole, P.N. Klein, and R.E. Tarjan. Finding minimum spanning trees in logarithmic time and linear work using random sampling. In *Proceedings of the 1996 ACM Symposium on Parallel Algorithms and Architectures*, pages 213–219, 1996.

[CLR91] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, 1991.

[DG97] F. Dehne and S. Gotz. Efficient parallel minimum spanning algorithms for coarse grained multicomputers and BSP, June 1997. Manuscript, Carleton University, Ottawa, Canada.

[DJR97] W. Dittrich, B. Juurlink, and I. Rieping, June 1997. Private communication by Ingo Rieping.

[GMR94] P.B. Gibbons, Y. Matias, and V. Ramachandran. The QRQW PRAM: Accounting for contention in parallel algorithms. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 638–648, 1994.

[GMR97] P. B. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation? In *Proceedings of the 1997 ACM Symposium on Parallel Algorithms and Architectures*, pages 72–83, 1997.

[HZ96] S. Halperin and U. Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests and for other basic graph connectivity problems. In *Proceedings of the Seventh ACM-SIAM Symposium on Discrete Algorithms*, pages 438–447, 1996.

[JM97] D. B. Johnson and P. Metaxas. Connected components in $O(\log^{3/2} n)$ parallel time for CREW PRAM. *Journal of Computer and System Sciences*, 54:227–242, 1997.

[Kar93] D. R. Karger. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In *34th Annual Symposium on Foundations of Computer Science*, pages 84–93, 1993.

[Kar95] D. R. Karger. *Random Sampling in Graph Optimization Problems.* PhD thesis, Department of Computer Science, Stanford University, 1995.

[KKT95] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42:321–328, 1995.

[KPRS97] V. King, C. K. Poon, V. Ramachandran, and S. Sinha. An optimal EREW PRAM algorithm for minimum spanning tree verification. *Information Processing Letters*, 62(3):153–159, 1997.

[Ram96] V. Ramachandran. Private communication to Uri Zwick, January, 1996. To be included in journal version of [HZ96].

[Ram97] V. Ramachandran. A general purpose shared-memory model for parallel computation. Technical Report TR97-16, Univ. of Texas at Austin, 1997.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

This article was processed using the LaTeX macro package with LLNCS style