A Randomized Time-Work Optimal Parallel Algorithm for Finding a Minimum Spanning Forest

Seth Pettie and Vijaya Ramachandran

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{seth,vlr}@cs.utexas.edu

Abstract. We present a randomized parallel algorithm to find a minimum spanning forest (MSF) in a weighted, undirected graph. On an EREW PRAM our algorithm runs in logarithmic time and linear work w.h.p. This is both time and work optimal and is the first provably optimal parallel algorithm under both measures.

1 Introduction

In this paper we present a randomized parallel minimum spanning forest (MSF) algorithm that is optimal with respect to both time and work. Finding an MSF is an important problem and there has been considerable prior work on parallel algorithms for the MSF problem. Following the linear-time sequential MSF algorithm of Karger, Klein and Tarjan [KKT95] came linear-work parallel MSF algorithms for the CRCW PRAM [CKT94,CKT96] and the EREW PRAM [PR97]. The best CRCW PRAM algorithm known to date [CKT96] runs in logarithmic time and linear work, but the time bound is not known to be optimal. The best EREW PRAM algorithm known prior to our work is the result of Poon and Ramachandran which runs in $O((m+n) \log n)$ time and linear work. All of these algorithms are randomized. Recently a deterministic EREW PRAM algorithm for MSF was given in [CHL99], which runs in logarithmic time with a linear number of processors, and hence with work $O((m+n) \log n)$, where $n$ and $m$ are the number of vertices and edges in the input graph. It was observed by Poon and Ramachandran [PR98] that the algorithm in [PR97] could be speeded up to run in $O((m+n) \cdot 2^{\log^* n})$ time and linear work by using the algorithm in [CHL99] as a subroutine (and by modifying the 'Contract' subroutine in [PR97]).

In this paper we improve on the running time of the algorithm in [PR97,PR98] to $O(\log n)$, which is asymptotically optimal and we improve on the algorithm in [CKT96] by matching the logarithmic time bound on the weaker EREW PRAM.

The structure of our algorithm is fairly simple. The most complex portion turns out to be the subroutine calls made to the 'CHL algorithm' [CHL99] (which we use as a black-box). As a result our algorithm can be used as a simpler alternative to several other parallel algorithms.
1. For the CRCW PRAM we can replace the calls to the CHL algorithm by calls to the simple logarithmic time, linear-processor CRCW algorithm in [AS87]. The resulting algorithm runs in logarithmic time and linear work and is considerably simpler than the MSF algorithm in [CKT96].

2. As modified for the CRCW PRAM, our algorithm is simpler than the linear-work logarithmic-time CRCW algorithm for connected components given in [Gaz91].

3. Our algorithm improves on the EREW connectivity and spanning tree algorithms in [HZ94,HZ96] since we compute a minimum spanning tree within the same time and work bounds. Our algorithm is arguably simpler than the algorithms in [HZ94,HZ96].

In the following we say that a result holds with high probability (or w.h.p.) in $n$ if the probability that it fails to hold is less than $1/n^c$, for any constant $c > 0$.

2 The High-Level Algorithm

Our algorithm is divided into two phases along the lines of the CRCW PRAM algorithm of [CKT96]. In Phase 1, the algorithm reduces the number of vertices in the graph from $n$ to $n/k$ vertices, where $n$ is the number of vertices in the input graph, and $k = (\log^2(n))^2$. To perform this reduction the algorithm uses the familiar recursion tree of depth $\log^* n$ [CKT94,CKT96,PR97], which gives rise to $O(2\log^* n)$ recursive calls, but the time needed per invocation in our algorithm is well below $O(\log n/2\log^* n)$. Thus the total time for Phase 1 is $O(\log n)$. We accomplish this by requiring Phase 1 to find only a subset of the MSF. By contracting this subset of the MSF we obtain a graph with $O(n/k)$ vertices. Phase 2 then uses an algorithm similar to the one in [PR97], but needs no recursion due to the reduced number of vertices in the graph. Thus Phase 2 is able to find the MSF of the contracted graph in $O(\log n)$ time and linear work.

**High-Level($G$)**

*(Phase 1)*

$G_\ell := \text{For all } v \in G, \text{ keep lightest } k \text{ edges of edge-list}(v)$

$M := \text{Find-k-Min}(G_\ell, \log^* n)$

$G' := \text{Contract all edges in } G \text{ appearing in } M$

*(Phase 2)*

$G_s := \text{Sample edges of } G' \text{ with prob. } 1/\sqrt{k} = 1/\log^2(n)$

$F_\ast := \text{Find-MSF}(G_s)$

$G_f := \text{Filter}(G', F_\ast)$

$F := \text{Find-MSF}(G_f)$

Return($M \cup F$)

**Theorem 1.** With high probability, High-Level($G$) returns the MSF of $G$ in $O(\log n)$ time using $(m + n)/\log n$ processors.

† We use $\log^r n$ to denote the log function iterated $r$ times, and $\log^* n$ to denote the minimum $r \text{ s.t. } \log^r n \leq 1$. 