A Coarse Grained Parallel Algorithm for Maximum Weight Matching in Trees[*]

Albert Chan and Frank Dehne
School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6
{achan,dehne}@scs.carleton.ca

Abstract

We present an efficient coarse grained parallel algorithm for computing a maximum weight matching in trees. A divide and conquer approach based on centroid decomposition of trees is used. Our algorithm requires $O(\log^2 p)$ communication rounds with $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ memory per processor (where p is the number of processors used and n is the size of the tree).

Keywords: parallel algorithms, coarse grained parallelism, BSP, trees, maximum weight matching,

1 Introduction

In this paper we consider the problem of finding a maximum weight matching in a tree. That is, given a tree $T=(V,E)$ and a weight $w_i \geq 0$ for each edge $e_i \in E$, we want to find a matching $M \subset E$ of T such that the total weight $w = \sum_{e_i \in M} w_i$ is maximizied.

Pawagi [4, 5] presented a PRAM algorithm for maximum weight tree matching. His algorithm requires $O(\log^2 n)$ computation time and $O(n)$ processors.

Unfortunately, theoretical results from PRAM algorithms do not necessarily match the speedups observed on *real* parallel machines. In this paper, we present a parallel algorithm that is more practical in that the assumption and cost model used reflect better the reality of commercially available multiprocessors. More precisely, we will use a version of the BSP model [6], referred to as the *coarse grained multicomputer* (CGM) model. In comparison to the BSP model, the CGM [2] allows only bulk messages in order to minimize message overhead costs. A CGM is comprised of a set of p processors $P_1, \ldots, P_p$ with $O\left(\frac{N}{p}\right)$ local memory per processor and an arbitrary communication network (or shared memory). All algorithms consist of alternating local computation and global communication rounds. Each communication round consists of routing a single h-relation with $h = O\left(\frac{N}{p}\right)$, i.e. each processor sends $O\left(\frac{N}{p}\right)$ data and receives $O\left(\frac{N}{p}\right)$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead. A CGM computation/communication round corresponds to a BSP superstep with communication cost $g\frac{N}{p}$ (plus the above "packing requirement"). Finding an optimal parallel algorithm in the CGM model is equivalent to minimizing the number of communication rounds as well as the total local computation time. The CGM model has the advantage of producing results which correspond much better to the actual performance on commercially available parallel machines. In addition to minimizing communication and computation volume, it also minimizes other important costs like message overheads and processor synchronization.

In this paper, we present a parallel CGM algorithm for maximum weight tree matching. The algorithm requires $O(\log^2 p)$ communication rounds and with $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ memory per processor. It assumes that the local memory per processor, $\frac{N}{p}$, is larger than p^{ε} for some fixed $\varepsilon > 0$. This assumption is true for all commercially available multiprocessors. Our results imply a BSP algorithm with $O(\log^2 p)$ supersteps, $O\left(g\log(p)\frac{n}{p}\right)$ communication time, and $O\left(\log(p)\frac{n}{p}\right)$ local computation time.

While our algorithm follows the general structure of Pawagi's PRAM algorithm [4, 5] it is, in the end, very different from the PRAM method. For example, Pawagi's algorithm includes steps such as finding the centroid of a tree, re-rooting a tree to a given node, and finding a maximum gain alternating path of a tree. These steps are straight forward for the PRAM but non-trivial for the CGM. The main contribution of this paper lies in the study of these individual steps and how they can be implemented on a CGM. Our main result is a parallel maximum weight tree matching algorithm which is much more practical and efficient on commercially available multiprocessors. Our CGM algorithms for finding the centroid of a tree, re-rooting a tree and finding a maximum gain

[*] Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

alternating path of a tree are also interesting in their own right and probably also useful for other coarse grained parallel graph algorithms.

A CGM heuristic algorithm for graph coloring has recently been presented in [3]. Graph coloring is closely related to maximum matching without weights but not to the weighted case studied in this paper. Furthermore, [3] presents a heuristic while our algorithm computes a guaranteed maximum weight matching.

The remainder of the paper is organized as follows. Section 2 presents an algorithm for finding the centroid of a tree. An algorithm for re-rooting a tree is presented in Section 3. Section 4 describes an algorithm to find a maximum gain alternating path of a tree. These algorithms are combined in Section 5 to obtain a CGM method for computing a maximum weight matching for a tree. Section 6 concludes the paper.

2 Partitioning A Tree Via It's Centroid

We are using divide and conquer to solve the maximum weight matching problem. In order for divide and conquer to work efficiently, we partition the tree such that the subproblems are guaranteed to shrink by at least a constant factor.

In this section, we present an algorithm to do such partitioning efficiently.

Definition 1 The centroid of a tree *T* is a vertex *c* that minimizes the size of the largest subtree in the forest generated by deletion of *c* from *T*.

Note that, for an *n*-vertex tree, the size of the largest subtree in the forest generated by the deletion of the centroid is less than or equal to $\frac{n}{2}$ [4,5].

Algorithm 1 Centroid of a Tree
Input: A tree *T* with *n* vertices and *n*-1 edges, rooted at *r*. All nodes and edges are evenly distributed over a *p*-processor coarse grained multi-computer.
Output: A vertex *c* which is the centroid of *T*.

(1) Compute the Euler tour of *T*. [1]
(2) Starting with the root *r*, using list ranking [1], calculate the distance of each vertex along the Euler tour. From these distances, each vertex can easily calculate the size of each subtree. Among these subtrees, the subtree with maximum size for the vertex is selected.
(3) Each processor determines which vertex has the smallest maximum subtree. The selected vertex is sent to Processor P_1.
(4) Processor P_1 selects and outputs the vertex that has the smallest maximum subtree.

--- End of Algorithm ---

Theorem 1 Algorithm 1 calculates the centroid of a tree using $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\log p\right)$ local computation, and $O\left(\frac{n}{p}\right)$ storage per processor.

Proof. The correctness of the algorithm follows immediately from the definition of the centroid. All steps are within the stated bounds. □

3 Re-rooting A Tree

In this section, we describe a CGM algorithm to re-root a tree.

Algorithm 2 Re-rooting a Tree
Input: A tree *T* with *n* vertices and *n*-1 edges, rooted at *r*. Each vertex *v* in *T* has a pointer *p*(*v*) to its parent. $p(r)=\emptyset$. A vertex $u \neq r$ in *T* (the new root).
Output: A tree *T*’ that is topologically equivalent to *T* but rooted at *u*.

(1) Find the Euler tour *E* of *T*. [1]
(2) Perform list ranking along *E* [1]. Each edge *e* in *T* will receive two ranks, r_{e_1} and r_{e_2}. Assume $r_{e_1} < r_{e_2}$.
(3) Let the ranks for the edge (*u*, *p*(*u*)) be r_1 and r_2. Broadcast these two ranks to all processors.
(4) Each processor for each edge *e*, compares r_{e_1}, r_{e_2} with r_1 and r_2. Mark the edge *e* if $r_{e_1} \leq r_1$ and $r_{e_2} \geq r_2$. Note that all marked edges form a path from u to r.
(5) Each processor for each vertex *v* checks if the edge (*v*, *p*(*v*)) is marked. If so, send *v* to *p*(*v*).
(6) Every vertex *v*’ that received a new vertex *v*” sets its parent *p*(*v*) = *v*”.
(7) The vertex *u* sets its parent to $p(u)=\emptyset$.

---End of Algorithm ---

Theorem 2 Using a *p* processor CGM, Algorithm 2 re-roots an *n*-node tree in $O(\log p)$ communication rounds with $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ memory per processor.

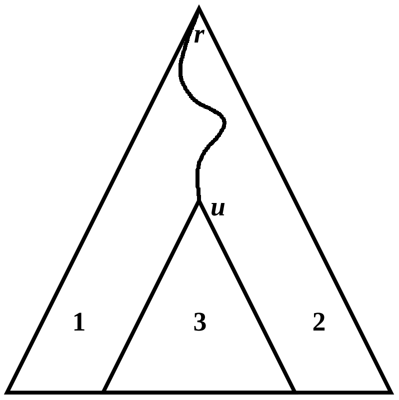

Figure 1: Re-rooting a Tree

Proof. The correctness of the algorithm lies on the properties of the Euler tour. As shown in Figure 1, the edges can be divided into three zones. The edges on the "left" of the path (u, …, r) (zone 1 in Figure 1) will have $r_{e_1} \le r_{e_2} \le r_1$. The edges on the "right" of the path (u, …, r) (zone 2 in Figure 1) will have $r_2 \le r_{e_1} \le r_{e_2}$. The edges "below" the path (u, …, r) (zone 3 in Figure 1) will have $r_1 \le r_{e_1} \le r_{e_2} \le r_2$. The edge on the path ($u$, …, r) (and only the edges on the path) will have $r_{e_1} \le r_1$ and $r_{e_2} \ge r_2$. Thus, Step 4 will mark all the edges on the path (u, …, r), and Step 6 will reverse all the parent pointers along the path. Therefore, Algorithm 2 can re-root a tree.

Step 1 requires $O(1)$ communication rounds, $O\left(\frac{n}{p}\right)$ local computation and local storage. Step 2 requires $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ local storage. Step 3 requires $O(1)$ communication rounds. Step 4 and 6 require no communication, $O\left(\frac{n}{p}\right)$ local computation and local storage. Step 5 requires $O(1)$ communication rounds. Step 7 requires no communication and $O(1)$ local computation. □

4 Maximum Gain Alternating Path

We start this section with some definitions and an observation.

Definition 2 Giving a graph $G=(V, E)$ and a matching $M \subseteq E$, a node $v \in V$ is free if and only if $\forall u \in V$, $(u,v) \notin E$ or $(u,v) \notin M$.

Definition 3 A path $P=(v_{i_1}, v_{i_2}, \ldots, v_{i_n} = v)$ in T is said to be an alternating path with respect to a matching M if every odd numbered edge on this path is in M and every even numbered edge is not in M or vice versa.

Definition 4 The gain of an alternating path P, denoted by $G(P)$, is the difference between the sum of the weights of the edges of P that are not in M and the sum of the weights of the edges of P that are in M.

An alternating path with positive gain can be used to obtain a new matching by removing from M all edges that are in P and adding the unmatched edges in P to M. Observe that the resulting matching is at least as large as the original one.

Algorithm 3 Maximum Gain Proper Alternating Path
Input: A weighted tree T rooted at r, a key k, and a maximum matching M in T.
Output: A vertex u in T such that the path from r to u forms a proper alternating path with maximum gain in T. All edges along the path (r, …, u) are marked with the key k.

```
(1) Let $p(v)$ be the parent of $v$, and $w_v$ be the weight of
    the edge $(v,p(v))$. For each node $v$ in the tree calculate
    $g(v)$ defined as follows:
        - if $v$ is the root $r$, then $g(r)=0$.
        - if $(v,p(v)) \in M$, then $g(v)=-w_v$.
        - if $(v,p(v)) \notin M$, then $g(v)=w_v$.
(2) Except for the root $r$, each node $v$ checks the values
    of $g(v)$ and $g(p(v))$. If both values are positive, delete
    the edge $(v,p(v))$.
(3) Using list ranking [1], each node $v$ determines its
    distance to the root $r$ as well as the sum of $g(v)$ along
    the path $(r, \ldots, v)$. (See proof of Theorem 3 for de-
    tail.) Let that sum be $G(v)$. Remove all nodes that are
    not able to reach the root $r$.
(4) For each remaining node $v$, if $v$ is not a leaf and
    $(v,p(v)) \notin M$, remove $v$.
(5) Perform a partial sum with respect to $G(v)$ (using the
    maximum operator) on the nodes that are left from
    Step 4, and determine the node $u$ that has the maxi-
    mum gain.
(6) Similar to Step (1) to (4) of Algorithm 2, mark all
    edges along the path $(r, \ldots, u)$ with the supplied key
    $k$.
--- End of Algorithm ---
```

In Algorithm 3, if r is not free, then the edge $(r,u) \in M$ must be in the maximum alternating path. This is because M is maximum. An alternating path with positive gain can be used to obtain a new matching by removing from M all edges that are in P and adding the unmatched edges in P to M. Observe that the resulting matching is at least as large as the original one.

Theorem 3 Algorithm 3 computes the maximum gain proper alternating path in $O(\log p)$ communication rounds with $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ memory per processor.

Proof. The correctness of the algorithm relies on whether we can find the correct gain $G(v)$ for each node v in Step 3. This can be achieved by using the Euler tour of the tree and then performing list ranking. More precisely, build an Euler tour by defining for each tree edge e_i between a node v_i and it's parent $p(v_i)$ two directed edges e_{i_1} = $(p(v_i),v_i)$ and e_{i_2} =$(v_i,p(v_i))$. Assign $g(v_i)$ to e_{i_1} and $-g(v_i)$ to e_{i_2} . See Figure 2 for an illustration.

A partial sum along the Euler tour, using [1], will give the gain for each node. Thus, in Step 5, the resulting path will be the maximum gain alternating path.

Step 1, 2 and 4 can be performed using $O(1)$ communication rounds, $O\left(\frac{n}{p}\right)$ computation and storage. Step 3

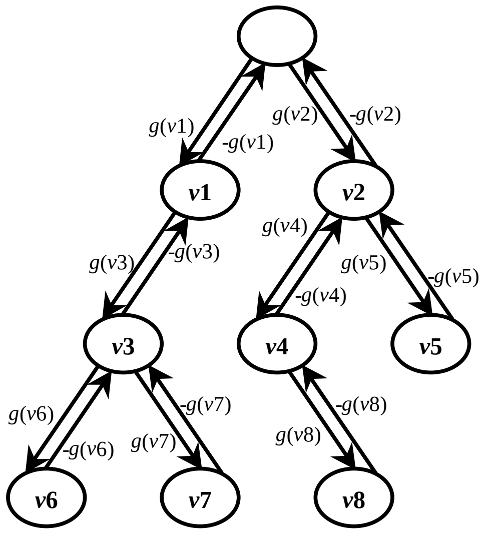

Figure 2: Grain Computation Through Euler Tour List Ranking

can be performed using $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\log p\right)$ computation and $O\left(\frac{n}{p}\right)$ storage. Step 5 and 6 can be executed using $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\right)$ computation and storage. □

5 Maximum Weight Matching For Trees

Algorithm 4 Maximum weight tree matching
Input: A tree T with n vertices and n-1 edges evenly distributed over a p-processor coarse grained multicomputer.
Output: The same tree T with all matched edges marked. All edges along the maximum gain path are marked. The maximum gain value and a boolean representing whether the root is free or not.

```
(1) Mark all vertices as free. If $p$=1, solve the problem
    sequentially and return.
(2) Compute the centroid $c$ of $T$ using Algorithm 1. Re-
    root $T$ at $c$ using Algorithm 2. Partition $T$ into sub-
    trees $T_1, T_2, \ldots, T_k$, where $k$ is the total in-degree of $c$.
    Each subtree $T_i$ is assigned a unique key $k_i$.
(3) For each subtree $T_i$, let $n_i$ be the size of the subtree.
    Assign $\lceil \frac{n_i}{n} p \rceil$ processors to recursively solve the
    problem on the subtree.
(4) Let $e_i$ be the edge connecting $c$ and $T_i$, and let its
    weight be $w_i$. Compute the value:
    $q_i = \begin{cases} w_i & \text{if root of } T_i \text{ is free} \\ w_i + \text{max. gain in } T_i & \text{otherwise} \end{cases}$
(5) Select a $j$ such that $q_j$ is the maximum.
(6) If $q_j \le 0$ then mark $c$ as free and go to Step 9.
(7) If $v_j$ is free then mark $(c, v_j)$ as matched, mark $c$ as
    not-free and go to Step 9.
(8) Mark the edge $(c, v_j)$ as matched. Invert the matching
    for all edges with key $k_j$. That is, for each edge $e$ with
    key $k_j$, mark the edge matched if it was not matched
    originally and mark it unmatched if it was matched
    originally. Mark $c$ as not-free.
(9) Re-root $T$ to $r$ using Algorithm 2.
--- End of Algorithm ---
```

Theorem 4 Algorithm 4 computes the maximum weight matching for T using $O(\log^2 p)$ communication rounds, with $O\left(\frac{n}{p}\log p\right)$ local computation and $O\left(\frac{n}{p}\right)$ memory per processor.

Proof. The main step is Step 4 where we select an edge e_i such that adding e_i to the matching will result in a maximum increase in the total weight in the matching. Note that in Step 3, the matching in each subtree T_i has been maximized. Therefore, if the root of T_i is free, then by adding e_i to the matching, the total increase will be w_i. If the root of T_i is not free then, by adding e_i to the matching and flipping the edges along the maximum gain alternating path, the total increase of the weight will be w_i plus the maximum gain in T_i. At the end of the algorithm, the resulting matching will always be maximized. Therefore, Algorithm 4 is correct.

Step 2 will guarantee that the size of the maximum subtree will be at most one half of the original tree. Thus, after at most $O(\log p)$ recursions, each subtrees will fit into one processor and we can apply the sequential algorithm to solve the subproblem. Step 1 will be executed once for each subtree, which will take no communication and use $O\left(\frac{n}{p}\right)$ local computation and storage. Step 2, 4 and 9 can be done using $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\log p\right)$ computation and $O\left(\frac{n}{p}\right)$ local storage. Step 5 and 8 can be done using $O(1)$ communication rounds, $O\left(\frac{n}{p}\right)$ computation and local storage. Step 6 and 7 can be done without communication and with $O\left(\frac{n}{p}\right)$ computation and local storage. Therefore, the total bounds for each recursion will be $O(\log p)$ communication rounds, $O\left(\frac{n}{p}\log p\right)$ computation and $O\left(\frac{n}{p}\right)$ storage. Since the size of each subproblem is guaranteed to be at most half the size of the original problem, the theorem follows. □

6 Conclusion

In this paper we presented an efficient coarse grained parallel algorithm for computing a maximum weight matching in trees. We also studied coarse grained parallel algorithms for finding the centroid of a tree, re-rooting a tree and finding a maximum gain path. The centroid, re-rooting and maximum gain path algorithms require $O(\log p)$ communication rounds and $O\left(\frac{n}{p}\log p\right)$ local computation, while the maximum weight tree matching algorithm requires $O(\log^2 p)$ communication rounds and $O\left(\frac{n}{p}\log p\right)$ local computation.

References

[1] E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro and S.W. Song, Efficient Parallel Graph Algorithms For Coarse Grained Multicomputers and BSP, in Proceedings ICALP '97 - 24th International Colloquium on Automata, Languages, and Programming, P. Degano and R. Gorrieri and A. Marchetti-Spaccamela (Eds.), 1997.

[2] F. Dehne (Ed.), Coarse grained parallel algorithms, Special issue of Algorithmica, Vol. 24 (2/3), pp 173-426, 1999.

[3] A. Gebremedhin, I. Guerin Lassous, J. Gustedt and J. Tell, Graph Coloring on a Coarse Grained Multiprocessor, to appear in Proc. WG 2000.

[4] S. Pawagi, Parallel Algorithm for Maximum Weight Matching in Trees, in Proceedings of the 1987 International Conference on Parallel Processing, 1987, pp 204-206.

[5] S. Pawagi, Parallel Algorithm for Maximum Weight Matching in Trees, Texhnical Report, Department of Computer Science, SUNY at Stony Brook, 1987.

[6] L. Valiant, A bridging model for parallel computation", Communication of the ACM, Vol 33, No. 8, 1990.