A Coarse-Grained Parallel Algorithm for Spanning Tree and Connected Components*

E. N. Cáceres[1], F. Dehne[2], H. Mongelli[1], S. W. Song[3], and J. L. Szwarcfiter[4]

[1] Universidade Federal de Mato Grosso do Sul, Campo Grande, Brazil,
edson@dct.ufms.br, http://www.dct.ufms.br/∼edson,
mongelli@dct.ufms.br, http://www.dct.ufms.br/∼mongelli
[2] Carleton University, Ottawa, Canada K1S 5B6,
frank@dehne.net, http://www.dehne.net
[3] Universidade de São Paulo, São Paulo, Brazil,
song@ime.usp.br, http://www.ime.usp.br/∼song
[4] Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil,
jayme@nce.ufrj.br, http://www.cos.ufrj.br/docentes/jayme.html

Abstract. Dehne et al. present a BSP/CGM algorithm for computing a spanning tree and the connected components of a graph, that requires $O(\log p)$ communication rounds, where p is the number of processors. It requires the solution of the Euler tour problem which in turn is based on the solution of the list ranking problem. We present a new approach that does not need to solve the Euler tour or the list ranking problem. It is based on the integer sorting algorithm which can be implemented efficiently on the BSP/CGM model [1].

1 Introduction

Computing a spanning tree and the connected components of a graph are basic problems and arise as subproblems in many applications. Parallel algorithms for these problems have been proposed by Hirschberg et al. [2]. An efficient CRCW PRAM algorithm takes $O(\log n)$ time with $O((m+n)\alpha(m,n))/\log n$ processors, where $\alpha(m,n)$ is the inverse of the Ackermann's function [3]. Dehne et al. [4] present a coarse-grained parallel algorithm that requires $O(\log p)$ communication rounds, where p is the number of processors. It is based on the Euler tour problem which in turn is based on the list ranking problem.

We present a new approach that does not need to solve the Euler tour or the list ranking problem. It still requires $O(\log p)$ communication rounds and has the advantage of avoiding the list ranking computation which has been shown to require large constants in practical implementations. The proposed algorithm is based on the integer sorting algorithm which can be implemented efficiently on the BSP/CGM model.

* Partially supported by FINEP-PRONEX-SAI Proc. No. 76.97.1022.00, CNPq Proc. No. 30.0317/02-6, 30.5218/03-4, 47.0163/03-8, 55.2028/02-9, FUNDECT-MS Proc. No. 41/100117/03, and the Natural Sciences and Engineering Research Council of Canada. We would also like to thank the anonymous referees for their review and helpful comments.

2 Preliminaries and Main Ideas

We use the *Coarse-Grained Multicomputer* (CGM) model [5], with p processors each with an $O(N/p)$ local memory, where N is the input size. A CGM algorithm consists of alternating local computation and global communication rounds. The communication cost is modeled by the number of communication rounds.

Consider a bipartite graph $H = (V_1, V_2, E)$ with vertex sets V_1 and V_2 and edge set E where each edge joins one vertex of V_1 and one vertex in V_2. If v is a vertex of a subgraph H' of H, then $d_{H'}(v)$ denotes the degree of v in H'. Let the vertices of V_1 be $u_1, u_2, \cdots, u_{n_1}$ and the vertices of V_2 be $v_1, v_2, \cdots, v_{n_2}$.

We define a *strut ST in* V_1 as a spanning forest of H such that each $v_i \in V_2$ is incident in ST with exactly one edge of E, and (u_j, v_i) is an edge of ST implies (u_k, v_i) is not an edge of H, for any $u_k \in V_1$, $k < j$. To define a *strut in* V_2, the roles for the sets V_1 and V_2 in the above definition are exchanged.

A vertex $u \in V_1$ is called *zero-difference* in ST if $d_H(u) - d_{ST}(u) = 0$. Otherwise, the vertex is referred to as *non-zero-difference*.

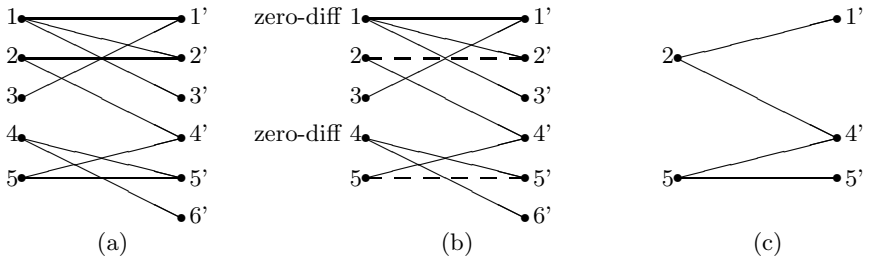

Fig. 1. (a) A bipartite graph (b) a strut (solid lines) (c) the compacted graph.

Fig. 1(a) shows a graph $H = (V_1, V_2, E)$ with $V_1 = \{1, 2, 3, 4, 5\}$, $V_2 = \{1', 2', 3', 4', 5', 6'\}$ and $E = \{(1, 1')\ (1, 2')\ (1, 3')\ (2, 2')\ (2, 4')\ (3, 1')\ (4, 5')\ (4, 6')\ (5, 4')\ (5, 5')\}$. We first compute a spanning forest for H by determining a strut ST in H (see Fig. 1(b)). Now compute the zero-difference vertices in V_1. Consider vertex 1. All the (three) edges in H incident with this vertex is also in ST. Thus $d_H(1) - d_{ST}(1) = 0$ and vertex 1 is zero-difference. Likewise vertex 4 is also zero-difference. Notice that vertex 2 is not zero-difference.

In the example we have two zero-difference vertices. If we have only *one* zero-difference vertex, then the problem is easily solved by adding to ST one arbitrary edge of $H - ST$ incident to each non-zero-difference vertex of ST. In case there are two or more zero-difference vertices we can do the following. For each zero-difference vertex $u \in V_1$ compact all the vertices $v_i \in V_2$ incident with u by compressing all the vertices v_i onto the smallest of the v_i. Repeat this until only one zero-difference vertex remains.

3 The CGM Algorithm for Bipartite Graphs

Let $H(V_1, V_2, E)$ be a bipartite graph with $|V_1| = n_1$, $|V_2| = n_2$ and $|E| = m$. Each of the p processors has $O(m/p)$ or $O((n_1+n_2)/p)$ local memory. Algorithm 1 computes a spanning tree of H, in $O(\log p)$ communication rounds.

Algorithm 1 - CGM Algorithm for Spanning Tree
Input: A bipartite graph $H(V_1, V_2, E)$ where $V_1 = \{u_1, \ldots, u_{n_1}\}$, $V_2 = \{v_1, \ldots, v_{n_2}\}$ and $|E| = m$. An edge (u_i, v_i) of E has a vertex u_i in V_1 and a vertex v_i in V_2. The m edges are equally distributed among the p processors at random.
Output: A spanning tree of G.
Phase I:

1: Initialize $\bar{V_1} := V_1$ and $\bar{V_2} := V_2$ and $\bar{E} := E$.
2: **for** $\log p$ times **do**
3: Sort the edges (u, v) of $\bar{E}$ by v and then by u.
4: **for** each v_i of V_2 **do**
5: Choose the smallest vertex u_j among all edges (u, v_i) and mark the edge (u_j, v_i). Let ST be the set of the marked edges.
6: **end for**
7: Compute the degree of each vertex $u \in \bar{V_1}$ in $H(\bar{V_1}, \bar{V_2}, \bar{E})$.
8: Compute the degree of each vertex $u \in \bar{V_1}$ in $H_{ST}(\bar{V_1}, \bar{V_2}, ST)$.
9: Using the degrees computed in the previous steps compute the number of zero-difference vertices.
10: **if** number of zero-difference vertices = 1 **then**
11: the algorithm finishes
12: **end if**
13: Compact the graph to produce the compacted graph $H(\bar{V_1}, \bar{V_2}, \bar{E})$.
14: **end for**

Phase II:

1: Compute a spanning forest with the edges of graph $H(\bar{V_1}, \bar{V_2}, \bar{E})$ that do not belong to ST and removing those with degree($\bar{u}$)=1 where $\bar{u} \in \bar{V_1}$.
2: Set all processors to *active* mode.
3: **for** k:=1 to $\log p$ **do**
4: Partition the active processors into groups of size two.
5: **for** each group P_i, P_j of active processors, $i < j$, in parallel **do**
6: Processor P_j sends its edge set $\bar{E_j}$ to processor P_i.
7: Processor P_j is set to *passive* mode.
8: Processor P_i computes the spanning forest $(\bar{V_1}, \bar{V_2}, \bar{E_s})$ of the graph $SF = (\bar{V_1}, \bar{V_2}, \bar{E_i} \cup \bar{E_j})$ and sets $\bar{E_i} := \bar{E_s}$.
9: **end for**
10: **end for**

Consider the graph of Fig. 1(a). We use array $EDGE$ to store edges of E: $(1,1')(1,2')(1,3')(2,2')(2,4')(3,1')(4,5')(4,6')(5,4')(5,5')$.

Make a copy of $EDGE$ in $EDGE'$. Lines 3 to 6 of Algorithm 1 obtain a strut ST. Line 3 sorts the edges in $EDGE'$ lexicographically in the following way. Given two edges (i, j) and (k, l) then $(i, j) < (k, l)$ when $j < l$ or $((j = l)$ and $(i <$

$k)$). Array $EDGE'$ contains the sorted edges: $(1,1')(3,1')(1,2')(2,2')(1,3')(2,4')(5,4')$ $(4,5')(5,5')(4,6')$.

Lines 4 to 6 find a strut ST in V_1. It is represented by solid lines of Fig. 1(b). Array $EDGE'$ represents ST: $(1,1')(1,2')(1,3')(2,4')(4,5')(4,6')$.

A strut ST in V_1 determines a spanning forest of H. Lines 7 to 9 find the zero-difference and non-zero-difference vertices of the strut ST. Determine the degrees of each of the vertices in V_1 and store in D_H. In our example $D_H = (3,2,1,2,2)$. Determine now which vertices of V_1 are zero-difference. For this, determine the degree of each of the vertices of V_1 in $EDGE'$ and store in D_{ST}. Again for our example, $D_{ST} = (3,1,0,2,0)$. Thus the zero-difference vertices are vertices $\{1,4\}$ and the non-zero-difference vertices are vertices $\{2,3,5\}$.

Line 13 produces a compacted graph. For each zero-difference vertex $u \in V_1$ compact all the vertices $v_i \in V_2$ incident with u by merging all the vertices v_i onto the smallest of the v_i. The new compacted graph $H(\bar{V_1}, \bar{V_2}, \bar{E})$ is shown in Fig. 1(c). Note that vertices $2'$ and $3'$ are compressed onto vertex $1'$ and therefore the original edge $(2,2')$ now becomes $(2,1')$.

Algorithm 1 computes the spanning tree of $H = (V_1, V_2, E)$ in $O(\log p)$ communication rounds. The proof can be found in [6].

4 Generalization and Main Results

To transform any graph into a bipartite graph, subdivide each edge by adding a new vertex on each edge. Consider the vertices of the original graph as belonging to V_1 and the new added vertices as V_2, then we have a resulting bipartite graph.

To determine the connected components of a graph, in each iteration of Algorithm 1, determine each of the sublists of $EDGE'$ formed by edges $(u,v), u = u_i$ that forms a tree, labeled by $EDGE'_{u_i}$. At the end of the algorithm, we can represent each tree with the smallest vertex. Each of the different vertices represent a connected component of the graph.

References

1. Chan, A., Dehne, F.: A note on coarse grained parallel integer sorting. Parallel Processing Letters **9** (1999) 533–538
2. Hirschberg, D.S., Chandra, A.K., Sarwate, D.V.: Computing connected components on parallel computers. Comm. ACM **22** (1979) 461–464
3. Karp, R.M., Ramachandran, V.: 17. In: Handbook of Theoretical Computer Science - J. van Leeuwen (ed.). Volume A. Elsevier/MIT Press (1990) 869–941
4. Dehne, F., Ferreira, A., Cáceres, E., Song, S.W., Roncato, A.: Efficient parallel graph algorithms for coarse grained multicomputers and BSP. Algorithmica **33** (2002) 183–200
5. Dehne, F., Fabri, A., Rau-Chaplin, A.: Scalable parallel geometric algorithms for coarse grained multicomputers. In: Proc. ACM 9th Annual Computational Geometry. (1993) 298–307
6. Cáceres, E.N., Dehne, F., Mongelli, H., Song, S.W., Szwarcfiter, J.L.: A coarse-grained parallel algorithm for spanning tree and connected components. Technical report, USP http://www.ime.usp.br/∼song/papers/span.pdf (2003)