A Parallel Algorithm for Transitive Closure

E. N. Cáceres*
Universidade Federal de Mato Grosso do Sul
Dept. of Computação e Estatística
Campo Grande-MS-Brazil
edson at dct. ufms. br

S. W. Song†
Universidade de São Paulo
Dept. de Ciência da Computação - IME
São Paulo-SP-Brazil
song at ime. usp. br

J. L. Szwarcfiter‡
Universidade Federal do Rio de Janeiro
Instituto de Matemática and
Núcleo de Computação Eletrônica and COPPE
Rio de Janeiro-RJ-Brazil
jayme at nce. ufrj. br

ABSTRACT
We present a parallel algorithm for the problem of computing the transitive closure for an acyclic digraph D with n vertices and m edges. We use the BSP/CGM model of parallel computing. Our algorithm uses $O(\log p)$ rounds of communications with p processors, where $p \leq n$, and each processor has $O(\frac{mn}{p})$ local memory. The local computation of each processor is equal to the product of the number of edges and vertices of D that are stored in p.

KEY WORDS
Parallel algorithm, transitive closure, graph algorithm, CGM, BSP

1 Introduction

The problem of computing the *transitive closure* of a digraph was first considered in 1959 by B. Roy [15] and a variety of sequential algorithms to solve this problem have been proposed ever since [1, 2, 7, 8, 9, 14, 16, 19].

These sequential solutions, usually are based on the use of the adjacency matrix of the digraph, considered as a Boolean matrix or use the adjacency matrix in more directed terms as a problem representation [13].

Parallel algorithms for this problem where presented by [10, 12] (PRAM), [11] (Arrays and Trees and Meshes of Trees) and [17] (Highly Scalable Multiprocessors).

We present an algorithm for computing the transitive closure of an acyclic digraph using the BSP/CGM Model.

*Partially supported by CNPq and FINEP-PRONEX-SAI Proc. No. 76.97.1022.00.

†Partially supported by FAPESP Proc. No. 99/07390-0, CNPq Proc. No. 52.3778/96-1, 46.1230/00-3, and 521097/01-0 and CNPq/NSF Collaborative Research Program Proc. No. 68.0037/99-3.

‡Partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, and by the Fundação de Amparo a Pesquisa do Estado do Rio de Janeiro, FAPERJ, Brazil.

2 Coarse-Grained Multicomputer (CGM) Model

The PRAM model [12] has been extensively utilized to produce important theoretical results on parallel algorithms. However, many of such algorithms could not be employed on real parallel machines. The limitations of the real machines, as compared to the requirements of the PRAM model, are both the number of processors and the memory range accessed by each of the processors.

In this paper, we present a parallel algorithm for transitive closure that is based on a more practical parallel model. More precisely, we will use a version of the BSP model [18] referred to as the *Coarse Grained Multicomputer* (CGM) model [5, 6]. It uses only two parameters: the input size N and the number of processors p. The term *coarse grained* means the size of the local memory is much larger than $O(1)$. Dehne et al. [6] define "much larger" as $N/p \gg p$. In comparison to the BSP model, the CGM allows only bulk messages in order to minimize message overhead.

A CGM consists of a set of p processors $P_1, \ldots, P_p$ with $O(N/p)$ local memory per processor and each processor is connected by a router that can send messages in a point-to-point fashion (or shared memory). A CGM algorithm consists of alternating local computation and global communication rounds separated by a barrier synchronization.

A computing round is equivalent to a superstep in the BSP model. In a computing round, we usually use the best sequential algorithm in each processor to process its data locally. In each communication round each processor sends $O(N/p)$ data and receives $O(N/p)$ data. Therefore, in terms of the BSP terminology, each communication round consists of routing a single h-relation with $h = O(N/p)$. We require that all information sent from a given processor to another processor in one communication round be packed into one long message, thereby minimizing the message overhead. In the CGM model, the communication cost is modeled by the number of communication rounds.

A CGM computation/communication round corresponds to a BSP superstep with communication cost $g\frac{N}{p}$, where g is the cost, in steps per word, of delivering message data. Finding an optimal algorithm in the coarse grained multicomputer 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 are closer to the actual performance on commercially available parallel machines.

Some algorithms for computational geometry and graph problems require a constant number or $O(\log p)$ communication rounds [3, 6]. Contrary to a PRAM algorithm, which is often designed for $p = O(N^k)$, with $k \in \mathcal{N}$, and with each processor receiving a small number of input data, in this model we consider the more realistic cases where $N \gg p$. The CGM model is particularly suitable in current parallel machines in which the global computing speed is considerably larger than the global communication speed.

3 Notation and Terminology

Let D be an acyclic digraph, with vertex set $V(D)$ and edge set $E(D)$, $|V(D)| = n$ and $|E(D)| = m$. Let $S \subseteq V(D)$. Denote by $D(S)$ the digraph formed exactly by the (directed) edges of D, having at least one of its extremes in S. If A is a path in D denote its length by $|A|$. A *linear extension* of D is a sequence $\{v_1, \ldots, v_n\}$ of its vertices, such that $(v_i, v_j) \in E(D) \Rightarrow i < j$.

The *transitive closure* of D is the digraph D^t, obtained from D by adding an edge (v_i, v_j), if there is a path from v_i to v_j in D, for all $v_i, v_j \in V(D)$.

4 The Algorithm

The following steps define a parallel algorithm for constructing the transitive closure of D, using p processors, where $1 \leq p \leq n$. It uses a sequential transitive closure algorithm.

Algorithm 1 Transitive Closure of a Digraph D
Input: (*i*) An acyclic digraph D, with $|V(D)| = n$ vertices and $|E(D)| = m$ edges; (*ii*) p processors.
Output: D^t, the transitive closure of D.

(1) Find a linear extension L of D;

(2) Let $S_1, \ldots, S_p$ be a partition of $V(D)$, whose parts have cardinalities as equal as possible, and where each S_j is formed by consecutive vertices in L.

For $j = 1, \ldots, p$, assign the vertices of S_j to processor p;

(3) In parallel, each processor p sequentially:

(3.1) constructs the digraph $D(S_j)$ from D

(3.2) computes the transitive closure $D^t(S_j)$ of $D(S_j)$

(3.3) includes in D the edges of $D^t(S_j) \setminus D(S_j)$

(4) After all processors have completed Step 3, verify if $D^t(S_j) = D(S_j)$, for all processors j. If true, the algorithm is terminated and D is the transitive closure of the input digraph. If false, go to Step 3.

— End of Algorithm —

5 Correctness and Complexity

The following proposition asserts the correctness of the algorithm.

Theorem 1 *The above algorithm computes correctly the transitive closure of the input digraph. Moreover, it requires at most* $1 + \lceil \log p \rceil$ *iterations of Step 3.*

Proof. Denote by D_i the digraph D at the end of the $i-$th iteration of Step 3. Let D_0 be the input digraph and D_i^t the transitive closure of D_i, $i = 0, 1, \ldots$. Since $D_i(S_j)$ is a subgraph of D_i, it follows that all edges of the transitive closure $D_i^t(S_j)$ of $D_i(S_j)$ also belong to D_i^t, and hence to D_0^t. Consequently, in order to show that the algorithm correctly computes the transitive closure D_0^t of D_0, it suffices to show that every edge of D_0^t is also an edge of some $D_i^t(S_j)$.

With this purpose, let $(v, w) \in E(D_0^t)$. We show that $(v, w) \in E(D_i^t(S_j))$, for some i, j. Because (v, w) is an edge of D_0^t, D_0 contains a path $z_1, \ldots, z_\ell$, from $v = z_1$ to $w = z_\ell$. For each k, $1 \leq k \leq \ell$, denote by $P(z_k)$ the processor to which z_k is assigned. Because the assignment of the vertices to the processors obeys a linear extension ordering, it follows that $P(z_1), \ldots, P(z_\ell)$ is a non-decreasing sequence. Therefore, vertices assigned to a same processor are consecutive in the sequence. Consequently, after completing the first iteration of Step 3, we know that D_1 contains a $v - w$ path A_1, formed solely by (a subset of) vertices of $z_1, \ldots, z_\ell$, assigned to distinct processors. Hence $|A_1| \leq p$. If $|A_1| = 1$ then $(v, w) \in E(D_1^t(s_j))$, implying the correctness of the algorithm. Otherwise, let z'_{k-1}, z'_k, z'_{k+1} be three consecutive vertices in A. Let $P(z'_k) = j$. Consequently, $(z'_{k-1}, z'_k), (z'_k, z'_{k+1}) \in E(D_1(S_j))$. The latter means that at the end of the second iteration of Step 3, (z'_{k-1}, z'_{k+1}) $E(D_2)$. Consequently, D_2 contains a $v - w$ path A_2, formed by a subset of vertices of A_1 satisfying $|A_2| = \lceil |A_1|/2 \rceil$. By induction, it follows that $|A_{\lceil \log |A_1| + 1 \rceil}| = 1$, that is, $(v, w) \in E(D^t_{\lceil \log |A_1| + 1 \rceil}(S_j))$, as required. Moreover, no more than $1 + \lceil \log p \rceil$ iterations of Step 3 were needed. □

In the sequel, we evaluate the complexity of the method. Basically, the algorithm consists of at most $1 + \lceil \log p \rceil$ parallel computations of a sequential transitive closure algorithm. We employ a sequential algorithm whose complexity is the product of the number of vertices and edges of the closure [9, 16]. Consider a worst case example, where D consists of a single path. Then D^t is a complete acyclic digraph. In this case, each processor j may compute the transitive closure $D^t(S_j)$ of a digraph $D(S_j)$, where $|V(D(S_j))| = p$ and $|E(D^t(S_j))| = (n/p)(n - n/p) = O(n^2/p)$. Since at most $1 + \lceil \log p \rceil$ iterations are required, the overall complexity is $O((\log p)(\frac{n^3}{p}))$.

The problem of finding a linear extension of an acyclic digraph can also be solved by a parallel CGM algorithm, with no difficulty.

Finally, observe that each processor handles at most $O(n^2/p)$ data. In fact, the algorithm fits into the coarse grained model.

6 Conclusion

We have described a parallel algorithm for computing the transitive closure of an acyclic digraph, using p processors. The algorithm is suitable for the CGM model. It employs at most $1 + \lceil \log p \rceil$ iterations of a sequential transitive closure algorithm. In fact, any such sequential algorithm could be applied. Therefore any possible improvement on a sequential algorithm would reflect in the parallel case.

References

[1] A. V. Aho, M. R. Garey and J. D. Ullman, The transitive reduction of a directed graph, *SIAM J. Comput.* 1 (1972) 131–137.

[2] V. L. Arlazarov, E. A. Dinic, M. M. Kronrod and I. A. Faradzev, On economical constructions of the transitive closure of an oriented graph, *Dokl. Akad. Nauk. SSSR* 194(3), *Sov. Math. Dokl.* 11 (1970) 1209–1210.

[3] E. Cáceres, 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: P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, eds., *Proceedings ICALP '97 - 24th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science, V. 1256 (1997) 390–400.

[4] H. Y. Chao and M. P. Harper, Minimizing redundant dependence and interprocessor synchronizations, *International Journal of Parallel Programming* 23 (1995) 245–262.

[5] F. Dehne (Ed.), Coarse grained parallel algorithms, in: *Special Issue of Algorithmica* 24 (1999) 173–426.

[6] F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable parallel computational geometry for coarse grained multicomputers, in: *Proc. ACM 9th Annual Computational Geometry* (1993) 298–307.

[7] M. J. Fisher and A. R. Meyer, Boolean matrix multiplication and transitive closure, in: *Proc. of 12th Ann. Symp. on Switching and Automata Theory* (1971) 129–131.

[8] M. E. Furman, Application of a Method of a Fast Multiplication of Matrices in the Problem of Finding the transitive closure of a graph, *Dokl. Akad. Nauk. SSSR* 194(3), *Sov. Math. Dokl.* 11 (1970) 1252.

[9] M. Habib, M. Morvan and J.-X. Rampon. On the calculation of transitive reduction-closure of orders, *Discrete Mathematics* 111 (1993) 289–303.

[10] J. JáJá, *Introduction to Parallel Algorithms*, (Addison-Wesley Publishing Company, 1992).

[11] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*, (Morgan Kaufmann Publishers, 1992).

[12] R. M. Karp and V. Ramachandran, Parallel Algorithms for Shared-Memory Machines, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science Vol. A*, (The MIT Press/Elsevier, 1990) Chapter 17, 869–941.

[13] J. van Leeuwen, Graph Algorithms, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science - Vol. A*, (The MIT Press/Elsevier, 1990) Chapter 10, 525–631.

[14] P. Purdon Jr, A transitive closure algorithm, it BIT 10 (1970) 76-94.

[15] B. Roy, Transitivité et connexité, *C.R. Acad. Sci. Paris* 249 (1959) 216–218.

[16] K. Simon, An improved algorithm for transitive closure on acyclic digraphs, *Theoretical Computer Science* 58 (1988) 325–346.

[17] A.A. Toptsis, Parallel transitive closure for multiprocessor, in: *International Conference on Computing and Information*, Lecture Notes in Computer Science, V. 497 (1991) 197–206.

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

[19] S. Warshall, A theorem on Boolean matrices, *J. Assoc. Comput. Mach.* 9 (1962) 11-12.