BSP/CGM Algorithms for the Transitive Closure Problem

Edson Norberto Cáceres
Cristiano Costa Argemom Vieira
Federal University of Mato Grosso do Sul
Dept. de Computação e Estatística
Campo Grande - MS, Brazil
E-mail:{edson,ccav}@dct.ufms.br

Abstract— **We present new implementations of BSP/CGM algorithms for the Transitive Closure Problem. Our strategies deal with size of the message and communication rounds, problems that cause inefficiency in the implementations of the transitive closure algorithms. The algorithms were implemented using LAM/MPI in two Beowulfs. The implementation results show the efficiency and scalability of the presented algorithms, improve the previous results and compare favorably with other parallel implementations. Besides we show that the presented algorithms can be used to solve real problems.**

I. Introduction

The search for efficient algorithms to compute the *transitive closure* of a directed graph (*digraph*) has been around for many years. It was first considered in 1959 by in the paper [Roy, 1959]. The best sequential algorithms that solve this problem have $O(mn)$ time complexity, where m and n are the number of edges and vertices of the digraph [Habib and Rampom, 1993] and [Simon, 1988]. There are other algorithms that are based on matrix multiplication that can be used for dense digraphs.

Using an approach based on the adjacency matrix of the digraph [Warshall, 1962] presented an $O(n^3)$ algorithm. This algorithm can be implemented using an adjacency bit matrix [Baase, 1978] with $O(\frac{n^3}{\alpha})$ time complexity, where α is the size of bits that can be stored in a primitive data.

Another approach to compute the transitive closure is using digraph traversal. Using *breadth first search* BFS (or *depth first search* DFS) we can find all the vertices that are reachable from a given vertex v. Applying this search to each vertex of the digraph we can compute the transitive closure. A BFS (or DFS) can be computed in $O(n+m)$[Cormen et al., 2001] time complexity. Then the transitive closure can be computed in $O(n\ (n+m))$ time complexity.

Parallel algorithms for this problem have been proposed to the PRAM model [JáJá, 1992, Karp and Ramachandran, 1990] BSP/GCM model [Tiskin, 2001, Cáceres et al., 2002, Alves et al., 2003, Cáceres and Vieira, 2004] and other architectures [Pagourtzis et al., 2001, 2002, Grama et al., 2003]. Tiskin observes that "is not clear yet whether the presented algorithm [Tiskin, 2001] is practical".

Using the BSP/CGM model, [Cáceres et al., 2002] presented an algorithm to compute the transitive closure of acyclic digraph using $\log p+1$ communication rounds where p is the number of processors and uses $O(mn/p)$ local computation. This algorithm assumes that the input is an acyclic digraph and relies on the so-called linear extension labeling of the input digraph vertices.

Using the ideas of this algorithm and the sequential algorithm of Warshall, [Alves et al., 2003] implemented a transitive closure algorithm with the MPI library in a Beowulf with 64 nodes with good speed-up. This implementation do not compute the linear extension of the digraph, and it can spend p communication rounds in the worst case, and uses $O(n^3/p)$ local computation. They describe the worst case where p communication rounds are necessary. With randomly generated digraphs, in their experiments, the algorithm always found the transitive closure with $O(\log p)$ communication rounds. Besides the fact of the good speed up, in this algorithm, the processors exchange huge amount of data ($O(n^2/p)$ in the worst case) in each communication round. Nevertheless, it is very fast and it obtained better execution times than the previous ones ([Pagourtzis et al., 2001, 2002]).

Combining the ideas of [Alves et al., 2003] and [Baase, 1978]. [Cáceres and Vieira, 2004] implemented a version of transitive closure algorithm with $\frac{p}{\alpha}$ communication rounds and $O(\frac{n^3}{\alpha p})$ local computation, where where α is the size of bits that can be stored in a primitive data. Since in this implementation, as p grows, the local computation is very fast, the total time of the algorithm is bounded by the communication time.

In this paper we deal with the main problems of the previous BSP/CGM algorithms for the transitive closure problem: the size of the messages, the local computation time and the number of communication rounds. We present four new implementations. The first and second deal with the size of the messages and using data compression we can decrease the amount of data exchanged in each communication round to $O(n/p)$. The third uses Bit matrix to improve the local computation time. The fourth increases the amount of memory used by each processor and can solve the transitive closure with a constant number of communication rounds. This algorithm uses a graph traversal strategy to compute the transitive closure of the input digraph.

II. Notation and Computational Model

Let $D=(V,E)$ be an acyclic digraph, $n=|V|$ and $m=|E|$ the number of vertices and edges of D, respectively. The vertices $v_i \in V$ are labeled with $1 \leq i \leq n$. We use both digraphs representations: adjacency list and adjacency matrix.

Partially supported by CNPq, FUNDECT-MS and CAPES

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

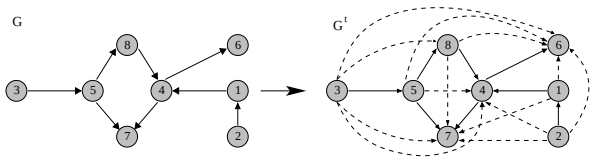

Fig. 1. Transitive Closure D^t of digraph D.

We denote a path between vertices $v_i, v_j \in V$ as follows: *(i)* $v_i \rightarrow v_j$, a path consisting of only one edge starting at vertex v_i and ending at vertex v_j.
(ii) $v_i \rightsquigarrow^k v_j$, a path starting at vertex v_i and ending at vertex v_j with k intermediate vertices ($k+1$ edges).
(iii) $v_i \rightsquigarrow^{\cdots k} v_j$, a path starting at vertex v_i and ending at vertex v_j with $v_1, v_2, ..., v_k$ as intermediate vertices.

When necessary, we denote an edge $e \in E$ as $e_{ij} = v_i \rightarrow v_j$. The edge e_{ij} is a leaving edge with respect to v_i and an incoming edges with respect to v_j.

Let $\Phi_q = \{q\frac{n}{p} + 1, .t.., (q+1)\frac{n}{p}\}$ be the set of the $\frac{n}{p}$ consecutive vertices that each processor receives in order to compute the transitive closure.

The transitive closure D^t of a digraph $D = (V, E)$ is obtained from D by adding an edge (v_i, v_j) if there is a path $v_i \rightsquigarrow^k v_j$, $0 < k \leq n$, in D. Figure 1 represents a digraph and its respective transitive closure.

In this paper, we use the BSP/CGM Model (Bulk Synchronous Parallel/Coarse Grained Multicomputer) [Valiant, 1990, Dehne, 1999]. Let N be the input size of the problem. A BSP/CGM computer consists of a set of p processors each with local memory and each processor is connected by a router that can send/receive messages in a point-to-point fashion. A BSP/CGM algorithm consists of alternating local computation and global communication rounds separated by a barrier synchronization.

In a computing round, we usually use the best sequential algorithm in each processor to process its data locally. 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 BSP/CGM model, the communication cost is modeled by the number of communication rounds. Each processor can send/receive at most $O(N/p)$ data in each communication round.

Finding an efficient algorithm on the BSP/CGM model is equivalent to minimizing the number of communication rounds as well as the total local computation time.

III. The Parallel Algorithms

Now we present the three new BSP/CGM parallel algorithms for the transitive closure problem. First we deal with the size of the messages that are exchanged between the processor. Then we decrease the local computation time and finally we design an algorithm that does not use communication rounds.

A. Dealing with Message Size - I

Analyzing the previous transitive closure algorithms we observed that the implementation presented by [Alves et al., 2003] and [Cáceres and Vieira, 2004] the communication cost is very expressive. In most cases, the messages have $O(n^2/p)$ size. One of the causes is the fact that the "new" edges are exchanged by the processors during the communication rounds without any further consideration.

We introduce a very simple technique that shrinks expressively the size of the messages that are exchanged by the processors. We define a *super-vertex* the set of all reachable vertices from v_i, i.e., $S_{v_i} = \{v : v_i \rightarrow v\}$. When a vertex v_i does not have any leaving edge, we denote $S_{v_i} = \varnothing$.

Let $V = \{v_i : 1 \leq i \leq n\}$ the set of vertices of D. We label the vertices v_i with labels $(v_i) = 2^{i-1}$ and we define $N_{v_i} = \sum_{\{v \in S_{v_i}\}} (v)$.

The Algorithm 1 shows the N_{v_i} computational steps for a S_{v_i} input and the Algorithm 2 shows the reverse (N_{vi} to S_{v_i}).

Algorithm 1 BuildN_{v_i}

Input: S_{v_i}
Output: $N_{v_i} = \sum S_{v_i}$.
1: $N_{v_i} \leftarrow 0$
2: **for all** $v_x \in S_{v_i}$ **do**
3: label(v_x) $\leftarrow 2^{x-1}$
4: **end for**
5: **for all** $v_x \in S_{v_i}$ **do**
6: $N_{v_i} \leftarrow N_{v_i}$ + label(v_x)
7: **end for**

It is easy to see that this labeling technique (Algorithm 1) guarantees a unique representation for each vertex. For example, if $S_{v_i} = \{1, 3, 4\}$, then the labels are $(v_1) = 1$, $(v_3) = 4$, $(v_4) = 8$ and $N_{v_i} = 13$. There is no other combination of the labels such that the sum is 13.

Algorithm 2 RestoreS_{v_i}

Input: N_{v_i}.
Output: S_{v_i}.
1: $S_{v_i} \leftarrow O$
2: $x \leftarrow 1$
3: **while** $x < N_{v_i}$ **do**
4: $x \leftarrow 2x$
5: **end while**
6: $x \leftarrow \frac{x}{2}$
7: **while** $N_{v_i} > 0$ **do**
8: **if** $N_{v_i} - x \geq 0$ **then**
9: $N_{v_i} \leftarrow N_{v_i} - x$
10: $S_{v_i} \leftarrow S_{v_i} \cup \{v_x\}$
11: **end if**
12: $x \leftarrow \frac{x}{2}$
13: **end while**

The restore procedure (Algorithm 2) computes the original values of the vertices, the set S_{v_i}. This is done by repeatedly subtracting (v_x) from N_{v_i}, starting with the greatest value of (v_x) to the smallest. The loop starting at the line **3** computes the greatest label that was stored in N_{v_i}.

In the [Alves et al., 2003] algorithm, each processor p_i receives the sub-matrices $D[(j-1)\frac{n}{p} + 1..j\frac{n}{p}][1..n]$ and $D[1..n][(j-1)\frac{n}{p} + 1..j\frac{n}{p}]$ and compute the local transitive closure. Before each processor p_i sends the computed transitivity edges to processor p_j, we apply algorithm 1, to

the message, creating a super-vertex for each line/column of the sub-matrix, and then send the compressed message to processor p_j. After processor p_j receives a message with super-vertices from processor p_i it applies the algorithm 2 restoring the original values of the vertices.

We describe the results of the implementation of this technique in Section IV. The size of the messages shrunk significantly decreasing the time of each communication round.

B. Dealing with Message Size - II

Now we introduce another simple technique that shrinks expressively the size of the messages that are exchanged by the processors.

For each new edge w_{ij}, $0 \leq i, j, < n$, the values of i and j (row and column) must be packed and during the communication rounds processor p_k sends the new edges that are in the l-th vertical strip and l-th horizontal strip to the processor p_l. This can be done in two ways:
i) Each edge can be represented as a value a where $a = i * n + j$.
ii) We represent each strip with a bit matrix where each edge can be stored as a bit.

Let Q^k be the number of new edges that will be send by processor p_k. Before each communication round we verify if we will send only the new edges or the whole strips (vertical and horizontal).

The Algorithm 3 implements the above ideas.

Algorithm 3 Parallel-Warshall-II

Input: D given as an $n \times n$ matrix W; p the number of processors; and each processor p_z ($0 \leq z \leq p-1$) stores the submatrix $W[(z\frac{n}{p}+1..(z+1)\frac{n}{p}][1..n]$ and $W[1..n][z\frac{n}{p}+1..(z+1)\frac{n}{p}]$
Output: D^t represented as W^t distributed by the p processors.

```
1:  repeat
2:    for k ← l(n/p) + 1 to (l + 1)(n/p) − 1 do
3:      for i ← 1 to n do
4:        for j ← 1 to n do
5:          if w_ik = 1 and w_kj = 1 then
6:            w_ij ← 1
7:          end if
8:        end for
9:      end for
10:   end for
11:   if Q^k > n^2/c then
12:     Send/Receive only the edges belongs another processor
13:   else
14:     Send/Receive the full strips another processor
15:   end if
16: until no new edges are computed
17: return D^t
```

Theorem 1: The Algorithm 3 uses $O(\frac{n^3}{p})$ local computation time with $O(p)$ communication rounds and needs $O(\frac{n^2}{p})$ local space.

We describe the results of the implementation of this technique in Section IV. The size of the messages shrunk significantly decreasing the time of each communication round.

C. Improving the Local Computation Time by Using the Bit Matrix

Representing the digraph with a bits adjacency matrix, [Cáceres and Vieira, 2004] implemented the transitive closure algorithm with expressive results. The implementation decreased significantly the local computation. The only drawback is that they use $O(n^2)$ local memory. Using $O(n^2)$ local memory we can sequentially compute the linear extension of the digraph in one processor and assure that with $\lg p+1$ communication rounds the algorithm computes the transitive closure. Besides the linear extension computation we can aggregate the ideas of Algorithm 3 and implement a new version of the [Cáceres and Vieira, 2004] algorithm.

The Algorithm 4 describes the above ideas.

Algorithm 4 PTC-Bit

Input: D given as an $n \times n$ bit matrix W; p the number of processors; and each processor p_z ($0 \leq z \leq p-1$) stores a copy of W.
Output: D^t represented as W^t distributed by the p processors.

```
1:  if z=0 then
2:    Compute L and Φ_{0,...,p−1}
3:  end if
4:  repeat
5:    for all k ∈ Φ_z in order do
6:      for i ← 1 to n do
7:        if w_ik = 1 then
8:          for j = 1 to n/α do
9:            w_ij ← w_ij ∨ w_kj
10:         end for
11:       end if
12:     end for
13:   end for
14:   if Q^k > n^2/c then
15:     Send/Receive only the edges belongs another processor
16:   else
17:     Send/Receive the full strips another processor
18:   end if
19: until no new edges are computed
20: return D^t
```

Because the vertex distribution to obey a linear extension of the digraph, $O(\log p)$ communication rounds are enough [Cáceres et al., 2002].

Theorem 2: The Algorithm 4 uses $O(\frac{n^3}{p\alpha})$ local computation time with $O(\log p)$ communication rounds and needs $O(n^2)$ local space.

D. Avoiding Communication Rounds

Since the amount of communication spent by the previous algorithms is very large, and limits the performance, we tried to trade communication with local space. Using the Warshall approach, seems that no improvement can be obtained with more local space. Actually, in most cases, increasing the local space does not guarantee any improvement to the parallel algorithm.

Using the digraph traversal (BFS or DFS) approach to compute the transitive closure with $O(n^2/p)$ local memory seems difficult, since parallel algorithms for these searches are not efficient. But we devise that if each processor stores the whole adjacency list of the digraph, each processor can apply sequential BFS or DFS to its n/p vertices. Then, processor p_l computes the transitive closure of the vertices in Φ_l, without any communication with other processors.

Let Γ^i be the spanning tree obtained by the breadth first search in the digraph D starting at the vertex v_i, the Algorithm 5 presents the steps for computing the transitive closure D^t.

Algorithm 5 TCP-BFS

Input: adjacency list Q of D
Output: D_l^t
1: **for all** $v_i \in \Phi_l$ **do**
2: $Q_i^t \leftarrow NULL$
3: Compute Γ^i.
4: Build the edge $q_{i,j}^t = \{(v_i, v_j) : j \in \Gamma^i\}$
5: **end for**

At the end of the algorithm, the transitive closure will be distributed by the processors.

Theorem 3: The BSP/CGM Algorithm 5 computes the transitive closure D^t of the input digraph D with $O(\frac{n}{p}(n + m))$ local computation, $O(n/p+m)$ local space and without any communication rounds.

Besides the fact that different processors can compute the same transitivity edge (we do not have communication rounds), the implementation of the algorithm showed an expressive performance when compared with the algorithms that use Warshall's approach.

IV. The Implementation Results

The digraphs used in the tests were generated randomly with probabilities varying from 15% to 20% of existing an edge between any two graph vertices.

TABLE I: **I**-Execution times of [1], **II**-Execution times of the Super-Vertex Algorithm and **III**-Execution times of the BFS Algorithm. ⋆(Alg.A with 1920 x 1920)

	512 x 512			1024 x 1024			2048 x 2048⋆		
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**I**	**II**	**III**
1	25.4	10.6	3.71	143	84.7	30	1614	679	232
2	9.3	5.3	1.87	123	42.7	15	603	341	116
4	8.3	2.8	0.93	84.4	21.7	7.6	257	171	58.2
8	3.9	1.5	0.47	36.4	11.3	3.8	123	87.7	29.1
16	2.6	1.3	0.24	18.0	6.0	1.9	69	45.6	14.5
32	1.9	0.9	0.12	15.6	6.5	0.9	68	24.4	7.2
64	3.3	0.7	0.06	12.1	7.5	0.4	49	24.4	3.6

We implemented the algorithms on two Beowulfs (clusters).

The first one with 64 nodes consisting of low cost microcomputers with 256MB RAM, 256MB swap memory, CPU Intel Pentium III 448.956 MHz, 512KB cache, in addition to two access nodes consisting of two microcomputers each with 512MB RAM, 512MB swap memory, CPU Pentium 4 2.00 GHz, and 512KB cache. The cluster is divided into two blocks of 32 nodes each. The nodes of each block are connected through a 100 Mb fast-Ethernet switch. Each of the access nodes is connected to the switch that connects the block of nodes and the two switches are connected. This cluster is the same used in [Alves et al., 2003].

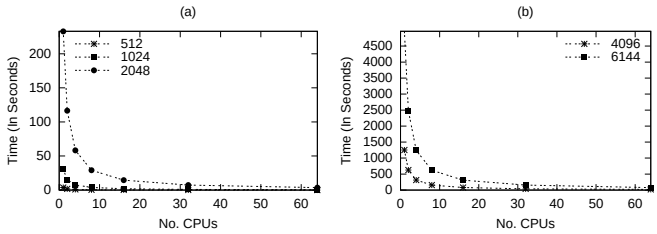

Fig. 2. Execution times of the BFS Algorithm

The second cluster is composed by 12 nodes consisting of 6 CPU Intel Pentium IV of 1.7Ghz and 6 CPU AMD Athlon de 1.6GHz. The nodes are connected by a 1Gb fast-Ethernet switch.

Our code is written in standard ANSI C++ using the LAM-MPI library. The execution times are expressed in seconds.

The Table IV presents a comparison between the implementation presented by [Alves et al., 2003] and our implementations.

The column II describe the times that uses Algorithm 1 an Algorithm 2 (super-vertex strategy). Its implementation got better execution times than the results presented by [Alves et al., 2003]. With the super-vertex strategy, actually we do a compression on the messages that are exchanged by the processors. In our tests we represented N_v with a data type with $\alpha = 32$ bits. The column III uses Algorithm 5, the breadth first search approach. The implementations were executed in digraphs with 512, 1.024, 2.048 and 4096 vertices and 50.000, 210.000, 800.000 and 3.300.000 edges, respectively.

The Table IV describes the results of [Alves et al., 2003] (I), and our implementation of Algorithm 3 (II-III). The Table IV describe the results of [Cáceres and Vieira, 2004] (I) and our implementation (II-III). Our results are much faster than the previous implementations.

The implementations were executed in digraphs with 1.024, 2.048 and 4.096 vertices and 210.000, 800.000 and 3.300.000 edges, respectively.

The Figure 3 shows the curves of Table IV.

The Figure 4 shows the curves of Table IV.

The Figure 5 shows the comparison of the communication times with/without using bit arrays sending/receiving messages in the two clusters.

V. Conclusions

We presented four new implementations of BSP/CGM parallel algorithm for the transitive closure. The first and second deal with the size of the messages that are exchanged

TABLE II: **I**-Execution times of [1], **II**-Execution times of the PTC-Edge Insertions Algorithm on the first cluster and **III**-Execution times of the PTC-Edge Insertions Algorithm on the second cluster. *(I - with 1920 x 1920)

	1024 x 1024			2048 x 2048			4096 x 4096	
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**II**	**III**
1	143	86	23	1614	688	190	5300	1521
2	123	44	12	603	348	100	2780	788
4	84	22	6	257	176	50	1427	407
8	36	12	3	123	91	27	723	206
16	18	6	-	69	48	-	377	-
32	15	5	-	68	26	-	205	-

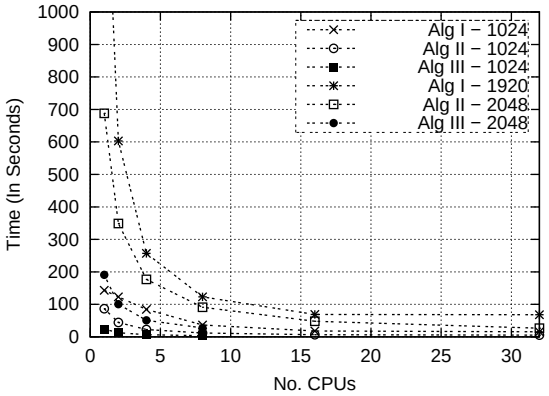

Fig. 3. Curves of Table IV

between processor. We introduced the techniques of of super-vertex and bit arrays that shrink the messages size. With these simple techniques we obtained a better performance implementing a BSP/CGM transitive closure algorithm. Then we notice that in the Cáceres and Vieira one processor could compute the linear extension of the input digraph and apply the same idea of the first implementation to decrease the size of the messages. Finally we implemented a BSP/CGM algorithm that uses a BFS search to compute the transitive closure of a digraph instead of the Warshall approach.

We implemented the algorithms in two Beowulfs with 64 and 12 nodes, respectively, using LAM-MPI. Since the BFS Algorithm does not use communication rounds it has the best performance between the proposed algorithms. The implementation results show the efficiency and scalability of the presented algorithms, improve the previous results and compare favorably with other parallel implementations.

TABLE III: **I**-Execution times of [4], **II**-Execution times of the PTC-BIT Algorithm on the first cluster. and **III**-Execution times of the PTC-BIT Algorithm on the second cluster

	1024 x 1024			2048 x 2048			4096 x 4096	
Procs.	**I**	**II**	**III**	**I**	**II**	**III**	**II**	**III**
1	7.7	7.6	2.1	63.3	61.6	17.5	505.3	134.6
2	5.2	3.9	1.2	36.4	31.3	9.9	254.4	77.8
4	4.0	1.9	0.7	23.7	15.8	5.1	127.9	39.5
8	3.4	1.0	0.3	17.4	8.1	2.6	64.7	20.5
16	3.7	0.5	-	18.3	4.4	-	33.2	-
32	3.5	0.4	-	15.2	2.5	-	17.9	-

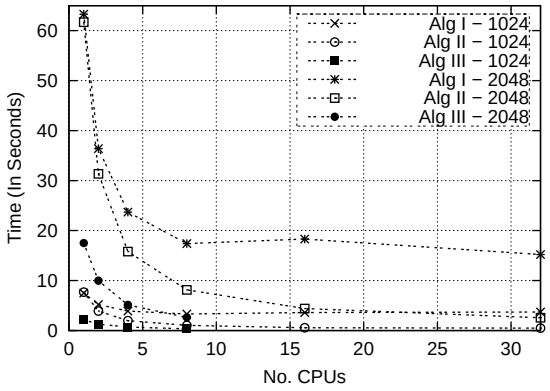

Fig. 4. Curves of Table IV

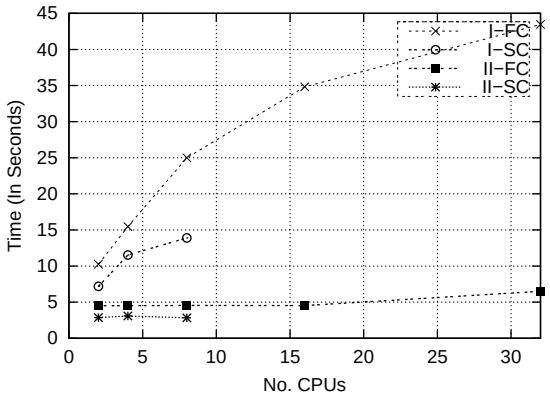

Fig. 5. I-FC: Alves et. al. Algorithm on the First Cluster, I-SC: Alves et. al. Algorithm on the Second Cluster, II-FC: Algorithm 3 on the First Cluster, II-SC: Algorithm 3 on the Second Cluster.

REFERENCES

C. E. R. Alves, E. N. Cáceres, A.A. Castro Jr., S. W. Song, and J.L. Szwarcfiter. Efficient parallel implementation of transitive closure of digraphs. *Lecture Notes in Computer Science*, 2840:126–133, 2003.

S. Baase. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 2nd edition, 1978.

E. N. Cáceres and C. C. A. Vieira. Revisiting a bsp/cgm transitive closure algorithm. *SBAC*, pages 174–179, 2004.

E. N. Cáceres, S. W. Song, and J.L. Szwarcfiter. A parallel algorithm for transitive closure. *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS 2002, Cambridge, USA, November 4-6*, pages 116–118, 2002.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 2nd edition, 2001.

F. Dehne. Coarse grained parallel algorithms. *Algorithmica*, 24(3/4):173–426, 1999.

A. Grama, V. Kumar, A. Gupta, and G. Karypis. *An Introduction to Parallel Computing: Design and Analisys of Algorithms*. Addison-Wesley, 2nd edition, 2003.

M. Habib, M.and Morvan and J. Rampom. On the calculation of transitive reduction-closure of orders. *Discrete Mathematics*, 111:289–303, 1993.

J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

R. M. Karp and V. Ramachandran. *Parallel Algorithms for*

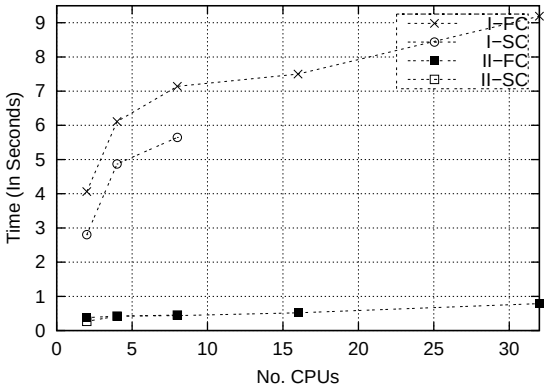

Fig. 6. I-FC: Cáceres and Vieira Algorithm on the First Cluster, I-SC: Cáceres and Vieira Algorithm on the Second Cluster, II-FC: Algorithm4 on the First Cluster, II-SC: Algorithm4 on the Second Cluster....

Shared-Memory Machines - Handbook of Theoretical Computer Science, volume A, chapter 17, pages 869–941. The MIT PRESS/Elsevier, 1990.

A. Pagourtzis, I. Patapov, and W. Rytter. Pvm computation of the transitive closure: The dependency graph approach. *Lectures Notes in Computer Science*, 2131:249–256, 2001.

A. Pagourtzis, I. Patapov, and W. Rytter. Observations on parallel computation of transitive and max-closure problems. *Lectures Notes in Computer Science*, 2474:217–225, 2002.

R. Roy. Transitivité et connexité. *C.R. Acad. Sci. Paris*, 249: 216–218, 1959.

K. Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58:325–346, 1988.

A. Tiskin. All-pairs shortest paths computation in the bsp model. *Lectures Notes in Computer Science*, 2076:178–189, 2001.

L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, 1990.

S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

AUTHOR BIOGRAPHIES

EDSON N. CÁCERES is a professor of computer science at the Universidade Federal de Mato Grosso do Sul. His research interests are parallel algorithms, cluster and grid computing, and peer-to-peer computing. Cáceres received a DSc from the Universidade Federal do Rio de Janeiro. He is Director of Education of the Brazilian Computer Society and he is a member of the IEEE Computer Society and ACM.

CRISTIANO C.A. VIEIRA is a assistant professor of computer science at the Universidade Federal de Mato Grosso do Sul. His research interests are cluster and grid computing and distance learning. Vieira received a MSc from Universidade Federal de Mato Grosso do Sul.