A parallel algorithm for solving tridiagonal linear systems on coarse grained multicomputer[*]

E. L. G. Saukas and S. W. Song

Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação
05508-900 São Paulo, SP, Brasil
telephone: (011) 818-6141 e 818-6135
email: {einar, song}@ime.usp.br

Abstract

The CGM (coarse-grained multicomputer) model has been proposed to be a model of parallelism sufficiently close to existing parallel machines. Despite its simplicity it intends to give a reasonable prediction of performance when parallel algorithms are implemented. Under the CGM model we design a communication-efficient parallel algorithm for the solution of tridiagonal linear systems with n equations and n unknowns. This algorithm requires only a constant number of communication rounds. The amount of data transmitted in each communication round is proportional to the number of processors and independent of n. In addition to showing its theoretical complexity, we have implemented this algorithm on a real distributed memory parallel machine. The results obtained are very promising and show an almost linear speedup for large n indicating the efficiency and scalability of the proposed algorithm.

Key-words: Coarse grained multicomputer, parallel algorithm, tridiagonal linear systems, band matrices, odd-even reduction algorithm

[*]Supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 96/12535-9, CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) Proc. No. 52 3778/96-1 and Commission of the European Communities through Project ITDC-207-82167.

A parallel algorithm for solving tridiagonal linear systems on coarse grained multicomputer

1 Introduction

A main problem in parallel computing is the well-known “software bottleneck”. Many current applications in parallel machines are restricted to trivially parallelizable problems with low communication requirements. In real machines communication time is usually much greater than computation time. Therefore for non-trivial problems many theoretically efficient parallel algorithms for the PRAM (shared memory model) or fine-grained network model often give disappointing speedups when implemented on real parallel machines.

The CGM (Coarse Grained Multicomputer) model was proposed [2, 3, 4] to be an adequate model of parallelism sufficiently close to existing parallel machines. It is a simple model and nevertheless intends to give a reasonable prediction of performance when parallel algorithms on this model are implemented.

In the CGM model the effort to reduce communication is centered on reducing the number of communication rounds. Under this model, we design a communication efficient parallel algorithm for the solution of tridiagonal linear systems with n equations and n unknowns. This algorithm requires only a constant number of communication rounds. The amount of data transmitted in each communication round is proportional to the number of processors and independent of n. In addition to showing the theoretical complexity, we have implemented the proposed algorithm on a real distributed memory parallel machine. The experimental results obtained are very promising and show an almost linear speedup, indicating the efficiency and scalability of the algorithm.

In section 2 we present the CGM model and its main characteristics. The parallel algorithm for the solution of tridiagonal linear systems is presented in section 3. We also present and discuss experimental results of this algorithm implemented on a parallel machine. Execution time curves of the proposed algorithm are given at the end of this paper. Finally in section 4 we make some concluding remarks.

2 The Coarse Grained Multicomputer (CGM)

The CGM (Coarse Grained Multicomputer) model was proposed by Dehne [2, 3, 4]. It is similar to Valiant’s BSP (Bulk-Synchronous Parallel) model [10]. However it uses only two parameters, n and p, where n is the size of the input and p the number of processors each with $O(n/p)$ local memory. The term “coarse grain” means the local memory size is considerably greater than $O(1)$. Each processor is connected by a router that can deliver messages in a point to point fashion. A CGM algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronizations. An algorithm under the CGM model can be expressed by the following generic scheme.

CGM Algorithm

```
for $i := 1$ to $R$ do
    begin
    $i$th local computation round
    $i$th communication round
    end
```

A computation round is equivalent to a computation superstep in the BSP model. In the computation round, we usually use the best possible sequential algorithm in each processor to process its data locally.

A communication round consists of a single h-relation with $h \leq n/p$, that is, each processor exchanges at most a total of $O(n/p)$ data with other processors in one communication round. The proposed algorithm requires the transmission of only $O(p)$ data in each communication round.

In the CGM model the communication cost of a parallel algorithm is modeled by the number of communication rounds. The objective is to design algorithms that require a small amount of communication rounds. Many algorithms for graph and geometric problems [1, 2, 3, 4] require only a constant or $O(\log p)$ rounds. Contrary to PRAM algorithms that frequently are designed for $p = O(n)$ and each processor receives a small number of input data, here we consider the more realistic case of $n >> p$.

The CGM model is particularly suitable in nowadays parallel machines where the overall computation speed is considerably larger than the overall communication speed.

3 Tridiagonal linear systems

In this section, we present a communication-efficient parallel algorithm for the solution of tridiagonal linear systems. Solution of a tridiagonal linear system is a very basic problem. It arises in the solution of many other systems such as partial differential equations using line relaxations or multigrid. It is therefore very important to have very efficient algorithms to solve tridiagonal linear systems.

Consider a tridiagonal linear system $Ax = b$ where matrix A is tridiagonal of order $n \times n$. The elements of a tridiagonal matrix $A = (a_{ij})$ are all zero except those located on the main diagonal and immediately above or immediately below the diagonal. In other words, A has elements $a_{ij} = 0$ for all pairs of i, j such that $|i - j| > 1$.

The proposed algorithm is based on the odd-even reduction (also called cyclic reduction) algorithm. The idea of the odd-even reduction is similar to some other numerical algorithms proposed in [6, 5]. It main idea is as follows [8]. Let

$$A = \begin{pmatrix} d_1 & u_1 & 0 & 0 & \dots & 0 \\ l_2 & d_2 & u_2 & 0 & \dots & 0 \\ 0 & l_3 & d_3 & u_3 & \dots & 0 \\ 0 & 0 & l_4 & d_4 & \dots & 0 \\ & & & \dots & & \\ 0 & 0 & 0 & 0 & \dots & d_n \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \\ x_n \end{pmatrix} \text{ and } b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \dots \\ b_n \end{pmatrix}$$

Each occurrence of x_i with odd index i is replaced by a linear function of x_{i-1} and x_{i+1}, both with even indices. The resulting system therefore consists of variables with even indices. The new linear system is also tridiagonal and thus we can apply the same idea recursively until only one equation for x_n remains. After solving for x_n, we work backwards and solve for $x_{n/2}$, then $x_{n/4}$, $x_{n/8}$ and so on. Having all the even-index values of x we proceed to solve the odd-index values of x. As an example consider the tridiagonal system from [8]

$$\begin{pmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 5 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 3 \\ \frac{18}{23} \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Replace x_1, x_3, x_5 and x_7 by

$$\begin{aligned} x_1 &= \frac{1}{4}(1 - x_2) \\ x_3 &= -\frac{1}{4}(1 - 2x_2 - x_4) \\ x_5 &= \frac{1}{3}(\frac{18}{23} - x_4 - 2x_6) \\ x_7 &= \frac{1}{5}(-2x_6 - 2x_8) \end{aligned}$$

This results in the following new reduced system.

$$\begin{pmatrix} \frac{15}{4} & \frac{1}{4} & 0 & 0 \\ 0 & \frac{5}{3} & -\frac{2}{3} & 0 \\ 0 & 0 & \frac{13}{5} & -\frac{2}{5} \\ 0 & 0 & -\frac{2}{5} & \frac{18}{5} \end{pmatrix} \begin{pmatrix} x_2 \\ x_4 \\ x_6 \\ x_8 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{63}{23} \\ 1 \\ 0 \end{pmatrix}$$

Replacing again x_2 and x_6 by

$$\begin{aligned} x_2 &= \frac{1}{15}(2 - x_4) \\ x_6 &= \frac{1}{13}(5 + 2x_8) \end{aligned}$$

results in:

$$\begin{pmatrix} \frac{5}{3} & \frac{4}{39} \\ 0 & \frac{46}{13} \end{pmatrix} \begin{pmatrix} x_4 \\ x_8 \end{pmatrix} = \begin{pmatrix} \frac{2687}{897} \\ \frac{2}{13} \end{pmatrix}$$

Finally replace x_4 by

$$x_4 = \frac{1}{5}(\frac{2687}{299} + \frac{4}{13}x_8)$$

We now have only one remaining equation for x_8. This gives us $x_8 = \frac{1}{23}$. Using this value we find $x_4 = \frac{9}{5}$. Then we find $x_2 = \frac{1}{75}$ and $x_6 = \frac{9}{23}$. Having found the even-index values we obtain finally the odd-index values of $x_1 = \frac{37}{150}, x_3 = \frac{31}{150}, x_5 = -\frac{3}{5}, x_7 = -\frac{4}{23}$.

During the odd-even reduction algorithm division by zero occurs when any of the odd-index diagonal elements of any matrices produced during the algorithm is zero. We assume here the tridiagonal matrix belongs to the class of symmetric positive definite matrices or diagonally dominant matrices. In such matrices division by zero never occurs during the algorithm [8].

The odd-even reduction algorithm requires $O(\log n)$ time. Leighton shows how it can be implemented on a X-tree [8]. A prefix-based algorithm for solving the tridiagonal linear system is due to Stone [9].

We propose here a CGM algorithm that requires a constant number of communication rounds. The algorithm to be presented here is similar to the algorithm proposed in [7]. By using the CGM paradigm, however, the algorithm proposed in this paper has been conceived independently in a relatively natural way, following the CGM principles, namely, minimizing communication rounds and using as much local processing as possible. Furthermore, we have implemented this algorithm on a distributed memory parallel machine to verify its efficiency in practice.

Consider a distributed memory parallel computer of p processors P_1, P_2, ..., P_p, with $n >> p$. Assume that each processor has sufficient local memory to store $O(n/p)$ elements. (See Fig. 1.)

We subdivide matrix A and the vector b into horizontal blocks or submatrices of n/p consecutive rows each. Each processor stores a submatrix of A and b. See below

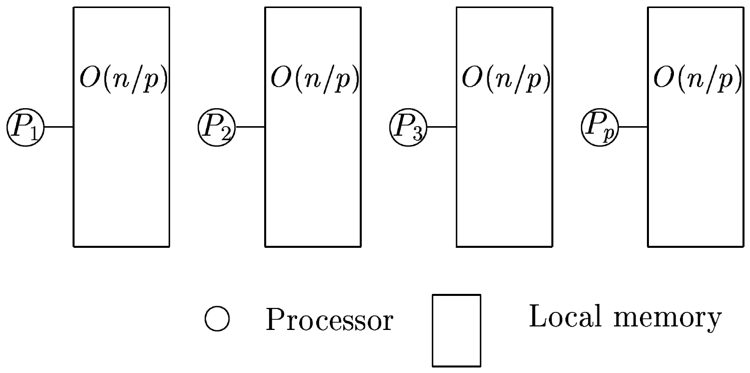

Figure 1: A parallel computer with distributed memory

an example of how submatrices of A and b are stored in the p processors assuming $n = 8$ and $p = 2$.

$$A = \left(\begin{array}{cccccccccc} 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & & \\ -1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & & \\ 0 & 2 & -4 & 1 & 0 & 0 & 0 & 0 & \Rightarrow P_1 & \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & & \\ -- & -- & -- & -- & -- & -- & -- & -- & & \\ 0 & 0 & 0 & 1 & 3 & 2 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 2 & 5 & 2 & \Rightarrow P_2 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & & \end{array} \right)$$

$$b = \left(\begin{array}{ll} 1 & \\ 0 & \\ 1 & \Rightarrow P_1 \\ 3 & \\ --- & \\ \frac{18}{23} & \\ 1 & \\ 0 & \Rightarrow P_2 \\ 0 & \end{array} \right)$$

The proposed algorithm makes use of a modified version of the odd-even reduction algorithm. Each processor applies odd-even reduction to its n/p equations and eliminates all equations but the first and the last ones. Thus each processor will have only four unknowns. Each processor then sends the two remaining equations to processor 1. Processor 1 applies odd-even reduction locally and solves for the unknowns. Each processor receives the solved unknowns from processor 1 and solves for the remaining unknowns locally. The algorithm consists of the following five phases alternating between local processing and communication.

Algorithm

(1) Each processor applies the odd-even algorithm locally to eliminate all rows except the first and the last rows in its submatrix. With this, each processor P_i eliminates $\frac{n}{p} - 2$ equations and $\frac{n}{p} - 2$ unknowns, namely, $x_{\frac{ni}{p}+2}$, $x_{\frac{ni}{p}+3}$, ..., $x_{\frac{n(i+1)}{p}-1}$.

(2) Each processor P_i sends its two remaining equations (with 4 unknowns $x_{\frac{ni}{p}}$, $x_{\frac{ni}{p}+1}$, $x_{\frac{n(i+1)}{p}}$, $x_{\frac{n(i+1)}{p}+1}$) to a same processor, say P_1. This processor thus obtains a system with $2p$ equations and $2p$ unknowns. Notice that this resulting system also consists of a tridiagonal matrix.

(3) Processor P_1 solves the system locally by odd-even reduction or any other sequential method. It thus obtains the solution for the $2p$ unknowns.

(4) Processor P_1 sends to each processor P_i the computed value for the 4 unknowns in the respective equations received in step 2.

(5) Each processor performs the inverse process of odd-even reduction used in step 1, by using the solution received for its two equations to solve the remaining equations.

Theorem 1 *A tridiagonal linear system with n equations and n unknowns can be solved on a CGM with p processors and $O(\frac{n}{p})$ local memory per processor using $O(1)$ communication rounds with the transmission of $O(p)$ data per round.*

Proof: Steps 1, 3 and 5 relate to local processing only. Steps 2 and 4 require one communication round each. In the first communication round (step 2) each processor sends a constant amount of data to processor 1, which in turn receives a total of $O(p)$ data. In the second communication round (step 4) processor 1 sends a total of $O(p)$ data and each of the remaining processors receive a constant amount of data. The theorem follows. $\square$

In the following we discuss very encouraging experimental results of this algorithm implemented on a distributed memory parallel machine. The parallel machine utilized for this experiment is the Parsytec PowerXplorer 16/32 Parallel Computing System, with 16 nodes under a 2-D topology. Each node consists of one PowerPC601 application processor (80 MHz, 80 Mflops peak performance) and one T805 communication processor, and 32 MBytes of local memory.

To ensure portability, the implementation is done in PVM (Parallel Virtual Machine), a standard communication interface. It is also implemented in Parix (the native system).

The sequential time was obtained with an optimized sequential algorithm run in a single processor (not the parallel algorithm run in one processor).

For the experiment we use the following system (with the solution of all $x_i = 1$).

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ & & & & \dots & & & \\ 0 & 0 & 0 & 0 & \dots & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

We obtain an almost linear speedup for large n, regardless of the communication protocol utilized, as shown by the following results. (See Fig. 2 and Fig. 3 for the time curves; Fig. 4 and Fig. 5 for the speedup curves.) The times are given in units of clock ticks of the machine (1 clock tick $= 10^{-6}$ seconds).

Execution time (in clock ticks) using interface PARIX:

n	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$
1024	8727	5301	3093	2487	3205
2048	18896	11434	5626	3731	3824
4096	39032	23635	11739	6238	5027
8192	78618	47582	23779	12322	7538
16384	155944	95717	47894	24275	13430
32768	308148	189636	95966	47896	25446
65536	613099	375044	189924	95115	48922
131072	1223813	746868	375206	187791	96193
262144	2429358	1492012	746879	370547	188907
524288	4840124	2960372	1493397	738832	372044

Execution time (in clock ticks) using interface PVM:

n	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$
1024	8727	5767	3372	3080	4714
2048	18896	11665	5887	3885	4794
4096	39032	23901	11730	6457	5903
8192	78618	47190	23654	12358	8077
16384	155944	95979	46693	24191	14005
32768	308148	189749	93867	47397	25552
65536	613099	375037	185122	94915	48508
131072	1223813	746283	365247	187116	95927
262144	2429358	1496103	727908	369437	187960
524288	4840124	2968610	1457379	736184	369990

4 Conclusion

One main objective of our research has been the investigation of methods that lead to the effective utilization of distributed memory parallel computers. The CGM (coarse-grained multicomputer) model was proposed by Dehne [2, 3, 4] to be an adequate model of parallelism sufficiently close to existing parallel machines. The effort to reduce communication is centered on reducing the number of communication rounds.

Under the CGM model we have designed a communication-efficient parallel algorithm for the solution of tridiagonal linear systems. This algorithm requires only a constant number of communication rounds with the transmission of $O(p)$ data per round. In addition to showing its theoretical complexity, we have implemented this algorithm on a real distributed memory parallel machine. The experimental results show an almost linear speedup for large n. This is a very significant result since the particular machine we used presents a considerable communication latency and low communication bandwidth. It indicates the efficiency and scalability of the proposed algorithm.

We conclude that the CGM model is particularly suitable in cases where the overall computation speed is considerably larger than the overall communication speed and the problem size is considerably larger than the number of processors, which is usually the case in practice.

References

[1] Caceres, E., Dehne, F., Ferreira, A., Flocchini, P., Rieping, I., Roncato, A., Santoro, N., and Song, S. W. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. Proceedings ICALP'97 - 24th International Colloquium on Automata, Languages, and Programming, P. Degano and A. Marchetti-Spaccamela (editors). Lecture Notes in Computer Science (1997)

[2] Dehne, F., Fabri, A. and Rau-Chaplin, A. Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers, in Proc. ACM 9th Annual Computational Geometry (1993) 298–307

[3] Dehne, F., Fabri, A. and Kenyon, C. Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time, in Proc. 6th IEEE Symposium on Parallel and Distributed Processing (1994) 586–593

[4] Dehne, F., Deng, X., Dymond, P., Fabri, A. and Kokhar, A. A. A randomized parallel 3D convex hull algorithm for coarse grained multicomputers, in Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA'95) (1995) 27–33

[5] Ericksen, J. Iterative and direct methods for solving Poisson's equation and their adaptability to ILLIAC IV. Center for Advanced Computation Document 60, University of Illinois, 1972.

[6] Kockney, R. The potential calculation and some applications. Meth. Comput. Physics **9** (1970) 135–211

[7] Krechel, A., Plum, H. J. and Stben, K. Solving tridiagonal linear systems in parallel on local memory MIMD machines. GMD technical report **372** (1989)

[8] Leighton, T. Introduction to parallel algorithms and architectures: arrays, trees and hypercubes. Morgan Kaufmann (1991)

[9] Stone, H. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. Journal of the ACM **20** (1973) 27–38

[10] Valiant, L. G. et al. General Purpose Parallel Architectures, *Handbook of Theoretical Computer Science*, Edited by J. van Leeuwen, MIT Press/Elsevier (1990) 943–972

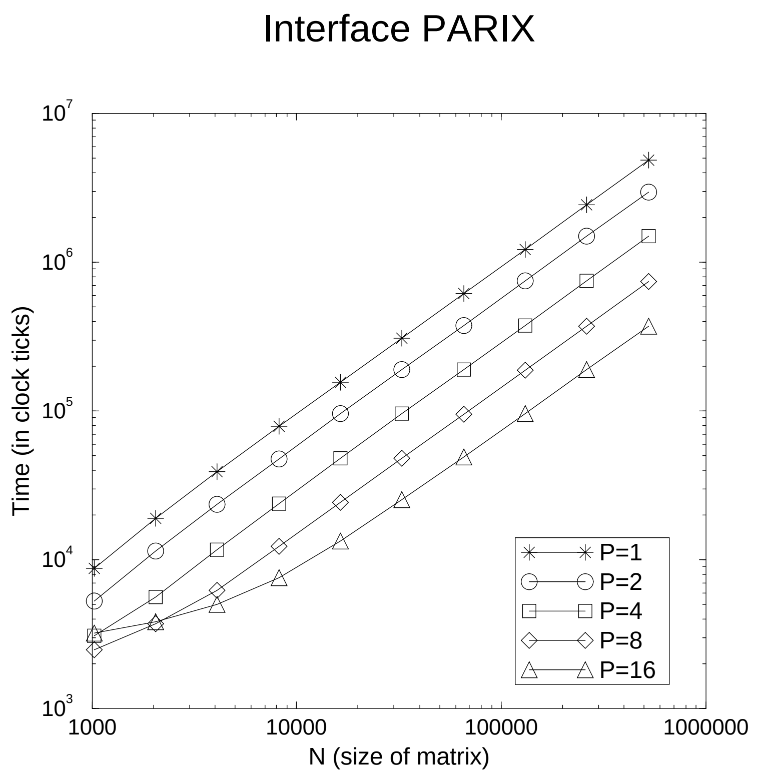

Figure 2: Total time of the tridiagonal algorithm in Parix

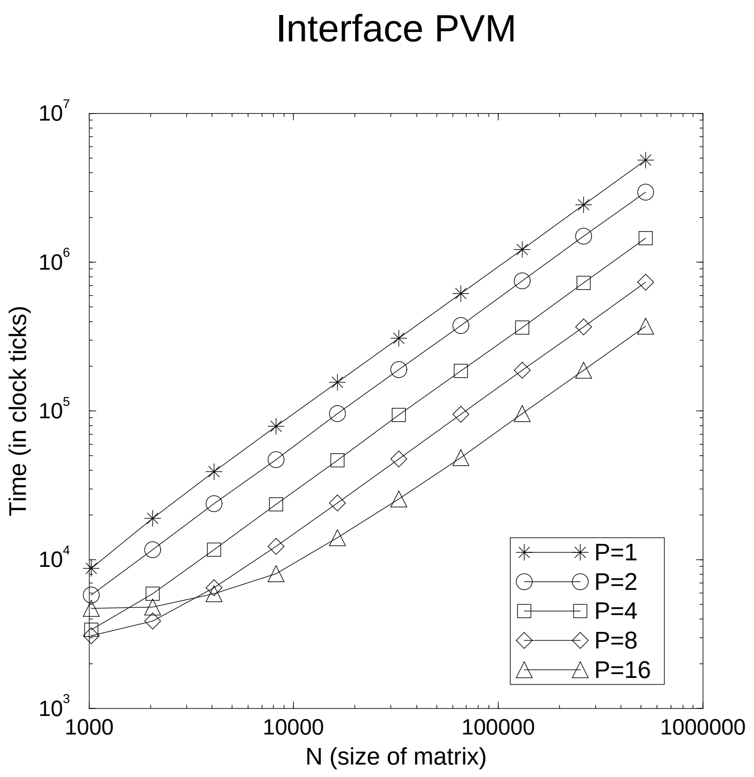

Figure 3: Total time of the tridiagonal algorithm in PVM

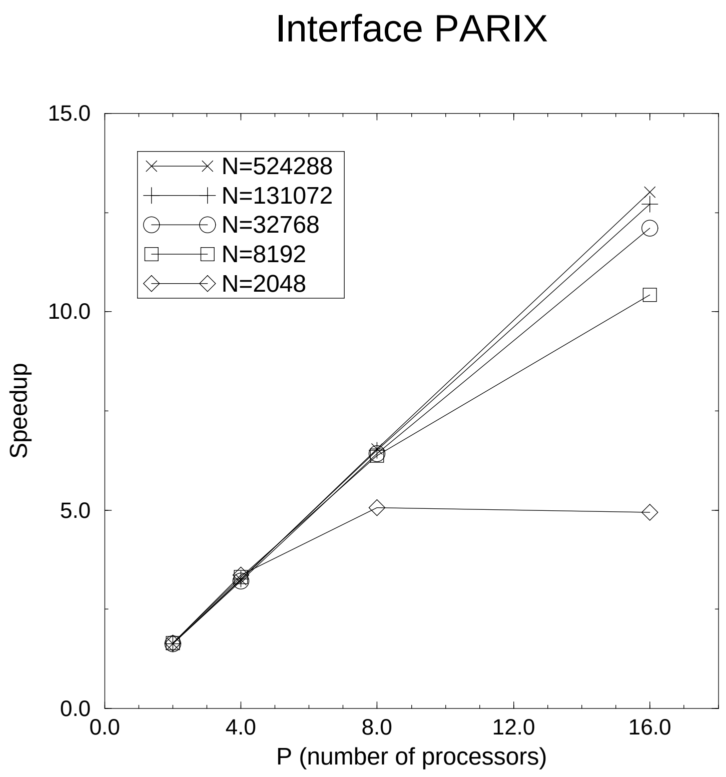

Figure 4: Speedup of the tridiagonal algorithm in Parix

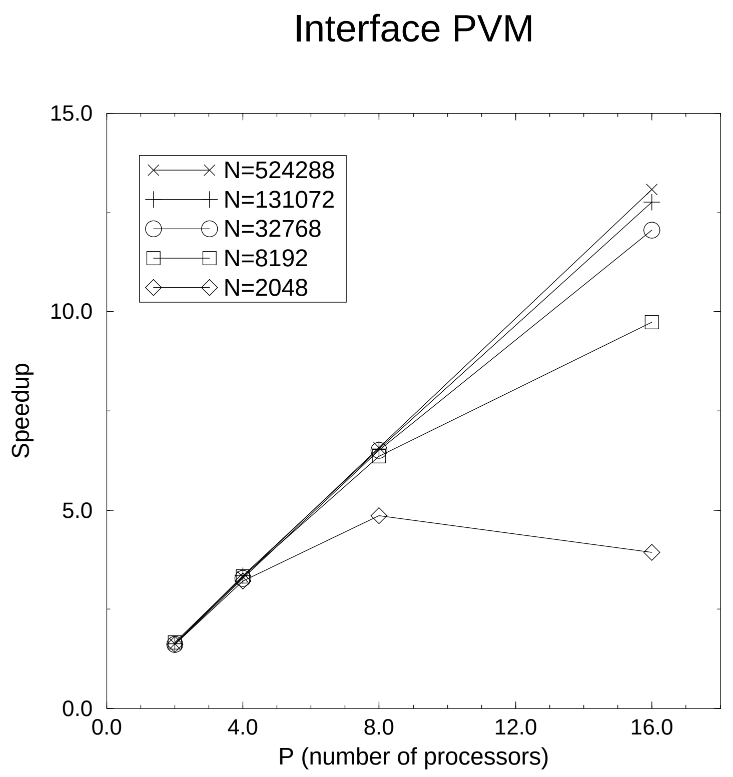

Figure 5: Speedup of the tridiagonal algorithm in PVM