A PARALLEL ALGORITHM FOR SOLVING TRIDIAGONAL LINEAR SYSTEMS

DUMITRU FANACHE[1]
[1]*Valahia* University of Targoviste, Department of Mathematics, Bd. Unirii 18, Targoviste, Romania
email_adress: d_fanache@yahoo.com

***Abstract**. The coarse-grainded architecture model has been proposed to be a model of parallelism sufficiently close existing parallel machines. Under this model we design a communication-efficient parallel algorithm for the solution of tridiagonal linear systems with n equation 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 shoing 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 proposed algorithm.*

***Keywords**: coarse grainded multicomputer, parallel algorithm, tridiagonal linear system, band matrices, odd-even cyclic reduction algorithm.*

1. Introduction

Many current application in parallel machines are restricted to trivially parallelizable problem with low communication requirements. In real machines communication time is usually much greater than computation time. Therefore for non-trivial problems many theoreticaly efficient parallel algorithms for the **PRAM** (shared memory model) or fine-grained network model often give disappoiting speedup when implemented on real parallel machines.

The **C**oarse **G**rained **M**ulticomputer(**CGM**) model 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* equation and *n* unknowns. This algorithms 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, indicate the efficiency and scalability of the algorithm.

2. The CGM model

The **CGM** model 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. 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 round and communication rounds separated by barrier synchronizations.

we usually the best possible sequential algorithm in each
ly. A communication round consists of a single *h*-relation
ssor 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 communications rounds. Many algorithms for graph an geometric problems [4] require only a constant or $O(\log p)$. 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 nowdays parallel machines where the overall computation speed is considerably larger than the overall communication speed.

3. The odd-even cyclic reduction method

Solution of a tridiagonal system is a very basic problem [**4**]. It arises in the solution of many other systems such as partial differential equation using line relaxations or multigrid [2]. A tridiagonal system is one in which all elements, except possibly those on the main diagonal, and the ones just above or below it, are 0'*s*. Instead of the usual notation a_{ij} for the element in row *i*, column *j*, of *A*, we use d_i to represent the main diagonal element a_{ii}, u_i for upper diagonal element $a_{i,i+1}$ and l_i for the lower diagonal element $a_{i,i-1}$. For the sake of uniformity, we define $l_0 = u_{n-1} = 0$. With the notation defined above, a tridiagonal system of linear equation can be written as follows, where $x_{-1} = x_n = 0$ are dummy variables that are introduced for uniformity:

$$\begin{aligned}
&l_0x_{-1} + d_0x_0 + u_0x_1 &&= b_0 \\
&l_1x_0 + d_1x_1 + u_1x_2 &&= b_1 \\
&l_2x_1 + d_2x_2 + u_3x_3 &&= b_2 \\
&\vdots \\
&l_{n-1}x_{n-2} + d_{n-1}x_{n-1} + u_{n-1}x_n &&= b_{n-1}
\end{aligned} \tag{1}$$

Odd-even cyclic reduction [3] is a recursive method for solving tridiagonal systems of size $n = 2^m - 1$. This method is divided into two parts: *reduction* and *back substitution*. The first step of reduction is to remove each odd-indexed x_i and create a tridiagonal system of size $2^{m-1}-1$. We then do the same to this new system and continue on in the same manner until we are left with a system of size 1. Observe that the *i*th equation can be rewritten as

$$x_i = (1/d_i)(b_i - l_i x_{i-1} - u_i x_{i+1}) \qquad (2)$$

Taking the above equation (2) for each odd i and substituing into even-numbered equations (the ones with even indices for *l*, *d*, *u* and *b*), we obtain for each even $i(0 \le i \le n)$ an equation of the form:

$$-\frac{l_{i-1}l_i}{d_{i-1}}x_{i-2} + \left(d_i - \frac{l_i u_{i-1}}{d_{i-1}} - \frac{u_i l_{i+1}}{d_{i+1}}\right)x_i - \frac{u_i u_{i+1}}{d_{i+1}}x_{i+2} = b_i - \frac{l_i b_{i-1}}{d_{i-1}} - \frac{u_i b_{i+1}}{d_{i+1}} \qquad (3)$$

In this way, the *n* equation is reduced to $\lceil n/2 \rceil$ tridiagonal linear equation in the even-indexed variables. Applying the same method recursively, leads to $n/4$ equations, then $n/8$ equations, and, eventually, a single equation in x_0. Solving this last equation to obtain the value of x_0, and substituting backwards, allows us to compute the value of each of the *n* variables. Figure 1 shows the structure of the odd-even reduction method.

Forming each new equation requires six multiplications, six divisions, and four additions, but these can all be done in parallel using $p = n/2$ processors. Assuming unit-time arithmetic operations, we obtain the recurrence $T(n) = T(n/2) + 8 \approx 8\log_2 n$ for the total number of computational steps. The six division operations can be replaced with one reciprocation per new equation, to find $1/d_j$ for each odd *j*, plus six multiplications. Obviously, the above odd-even reduction method is applicable only if none of the d_j values obtainded in the course of the computation is 0.

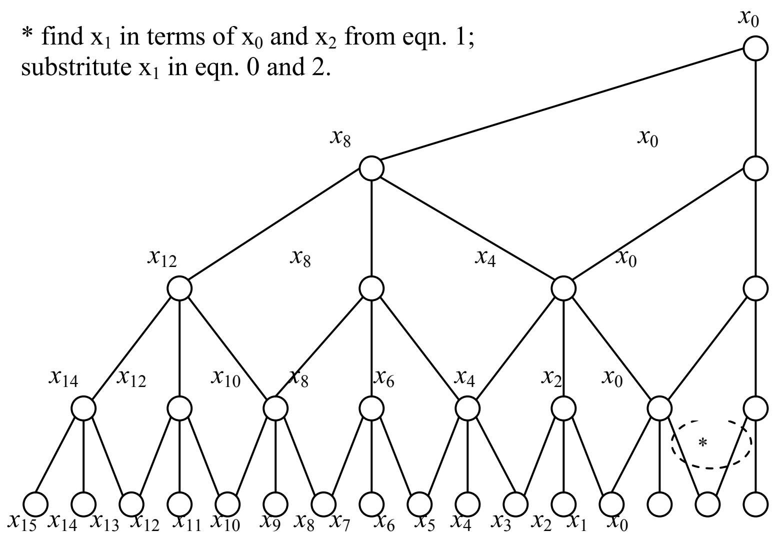
* find x1 in terms of x0 and x2 from eqn. 1;
substritute x1 in eqn. 0 and 2.
x0
x8
x0
x12
x8
x4
x0
x14
x12
x10
x8
x6
x4
x2
x0
*
x15 x14 x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 x0

Figure 1. The structure of odd-even reduction for solving tridiagonal system of linear equation

In the above analysis, interprocessor communication time was not taken into account. The analysis is thus valid only for the **PRAM** or for an architecture whose topology matches the communication structure shown in Figure 1. A binary *X*-tree architecture whose communication structure closely matches the needs of the above computation.

To perform odd-even reduction on linear array of $p = n$ processors, we can assume that each processor initially holds one of the *n* equations. Direct one-step communication between neighboring processors leads to even-numbered processors obtaining the reduced set of $n/2$ equations as discussed above. The next reduction phase requires two-step communication, then four-step, and eventualy $(n/2)$step, leading to linear running time(of the same order as sequential time). On an *n*-processor 2*D* mesh, odd-even reduction can be easily organized to require $\Theta(\sqrt{p})$ time. It is worth noting that solving a tridiagonal system of linear equation can be converted to a parallel prefix problem as follows [3, 5].

Define the 3×3 matrix G_i as

$$G_i = \begin{bmatrix} -d_i/u_i & -l_i/u_i & b_i/u_i \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Then, the *i*th equation can be written in matrix form as

$$\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \begin{bmatrix} -d_i/u_i & -l_i/u_i & b_i/u_i \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix} = G_i \times G_{i-1} \times \cdots \times G_0 \times \begin{bmatrix} x_0 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

In particular, we have

$$\begin{bmatrix} x_{n-1} \\ x_{n-2} \\ 1 \end{bmatrix} = G_{n-2} \times G_{n-3} \times \cdots \times G_0 \times \begin{bmatrix} x_0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

Solving this last set of three equations provides the values of x_0, which can then used to determine the values of all other variables, given the prefix results $G_i \times G_{i-1} \times \cdots \times G_0$ for odd *i*.

We propose here a **CGM** algorithm that requires a constant number of comunication rounds. 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. Furthemore, 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 $P_0, P_2, \cdots, P_{p-1}$ with $n >> p$. Assume that each processor has sufficient local memory to store $O(n/p)$ elements. (see Figure 2). 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*.

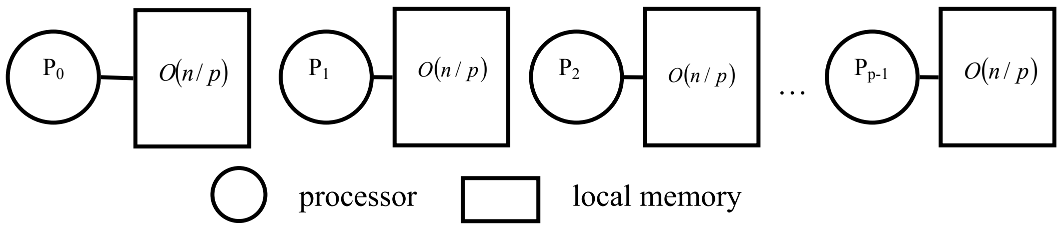

Figure 2. A parallel computer with distributed memory

The proposed algorithm makes use of a modified version of the odd-even reduction algorithm[1,2,6]. Each processor applies odd-even reduction to its n/p equations and eliminates all equation but the first and the last ones. Thus each processor will have only four unknowns. Each processor then send the two remaining equations to procesor 0. Processor 0 applies odd-even reduction locally and solves for unknowns. Each processor receives the solved unknowns from processor 0 and solves for the remaining unknowns locally. The algorithm consists of the following *five phases* (see Figure 3) alternating between local processing and communication.

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

Proof: Step 1, 3 and 5 relate to local processing only. Step 2 and 4 require one communication round each. In the first communication round (step 2) each processor send a constant amount of data to processor 0, which in turn receives a total of $O(p)$ data. In the second communication round (step 4) processor 0 sends a total of $O(p)$ data and each of remaining processors receive a constant amount of data (The theorem follows).

The sequential time was obtained with an optimized sequential algorithm run in a single processor (not the parallel algorithm run time one processor). For the experiment we use the following system: $l_i = u_i = -1$ for all $i = 1,\cdots,n-1$ and $d_i = 2$ for all $i = 0,\cdots,n-1$ (with the solution of all $x_i = 1$ for $i = 0,1,\cdots,n-1$).

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

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
Execution time(in clock ticks) using MPI					

S1. 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 $n/p-2$ equations and $n/p-2$ unknowns, namely $x_{\frac{ni}{p}+2}, x_{\frac{ni}{p}+3}, \cdots, x_{\frac{n(i+1)}{p}-1}$.

S2. Each processor P_i sends its two remaining equation (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_0. This processor thus obtains a system with $2p$ equations and $2p$ unknowns. Notice that this resulting system also consists of a tridiagonal matrix.

S3. Processor P_0 solves the system locally by od-even reduction or any other sequential method. It thus obtains the solution for the $2p$ unknowns.

S4. Processor P_0 sends to each processor P_i the computed value for 4 unknowns in the respective equations received in step 2.

S5. 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.

Figure 3. Parallel algorithm odd-even reduction

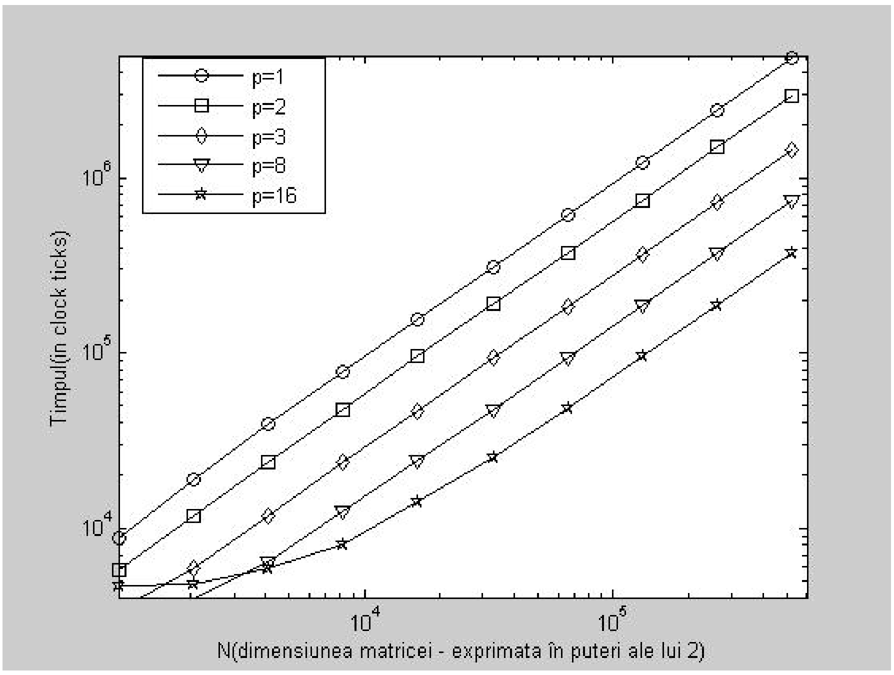

Figure 4. Total time of the tridiagonal algorithm in MPI

4. Conclusions

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 round with 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 indicate the efficiency and scalability of the proposed algorithm.

References

[1]. Lixing Ma, Frederick C. Haris Jr. *A Parallel Algorithm for Solving a Tridiagonal Linear System with ADI Method*, Department of Computer Science University of Nevada, Reno,NV 89557
[2]. Eunice E. Santos, *Optimal and Efficient Parallel Tridiagonal Solvers Using Direct Methods , The Journal of Supercomputing, 30, 97-115,2004 Kluwer Academic Publishers*, The Netherlands
[3]. Parhani, B., *Introducing to Parallel Processing* (*Algorithms and Architectures* (electronic format), University of California at Santa Barbara
[4]. Fanache, D., Smeureanu, I., *A Linear Algorithm for Black Scholes Economic Model*, Economic Computation and Economic Cybernetics Studies and Research, nr 2, 2008, ISSN 0585-7511
[5]. C.M. Da Fonseca, *On the eigenvalues of some tridiagonal matrices*, Departamento de Matemática, Universidade de Coimbra, 3001-454 Coimbra, Portugal
[6]. Usmani, R., *Inversion of tridiagonal Jacobi matrix*, Linear Algebra Appl.212/213(1994) 413-414

THE POSITION VECTOR OF A POINT EXERCISES

DUMITRESCU ION[1]
[1]*Petru Cercel* Highschool Târgovişte

The vectorial method is is applicable for studying a large class of properties of the euclidian space (coliniariy, coplanarity, parallelism, perpendicularity, calculation of angles, distances, volumes, etc)

There are some exercices in geometry in which the vectorial method is more direct and eloquent. It is good to know several methods. But more important is to know to choose, adapt and use the best suited method.

There will be presented several geometry problems solved by using the vectorial method.

SENTENCE: Points A, B, M, where $M \neq B$ and $r \in R - \{-1\}$, $\overrightarrow{AM} = r \cdot \overrightarrow{MB}$. Then, for any point $O \in P$ we have

$$\overrightarrow{OM} = \frac{\overrightarrow{OA} + r \cdot \overrightarrow{OB}}{1 + r}$$

and reciprocal.