CONVERGENCE ANALYSIS FOR A PARALLEL JACOBI ALGORITHM

M. El-Attar
Department of Engineering Mathematics, Faculty of Engineering,
Alexandria University, Alexandria, Egypt.

ABSTRACT

An iterative parallel scheme for the solution of banded systems of equations is introduced. It is based on the implementation of the theory of matrix splitting. A convergence analysis for the proposed algorithm is carefully introduced and the speed-up of the parallel implementation is studied.

INTRODUCTION

The solution of large systems of linear or linearized equations is one of the central problems in engineering applications and computational mathematics. Systems which have a banded coefficient symmetric matrix, usually resulting from finite element modelling, are of great interest, however banded non-symmetric systems are also common in the analysis of geomaterials modelled with non-associative plasticity. The solution of these systems in conventional computer architectures is algorithmically simple problem, but suffers from extensive time and memory limitations. Parallel and distributed computation is currently an area of intense research activity, the availability of powerful parallel computers is generating interest in the development of new distributed algorithms in order to solve problems that were not addressed in the past due to cost and speed constraints. Recent studies [1-5], have addressed the generation of parallel schemes for the solution of large systems of linear equations.

In this paper an iterative parallel block Jacobi type algorithm for solving large banded linear systems is introduced. The algorithm is a two stage iterative type based on the theory of matrix multisplitting. A theoretical convergence analysis for the proposed algorithm is described. The present algorithm is tested for the solution of a banded systems on a small transputer unit (A transputer is a microprocessor with its own local memory and with links that can connect one transputer to another). The architecture used in this study is based on a mesh connection of 16 transputers as shown in Figure (1). Processors 1 and 16 are connected to IBM AT which acts as a controller that downloads instructions and data to the transputer unit. From a practical point of view this connection represents a grid, a hypercube, a pipeline and a ring. The pipeline configuration is the one used to implement the algorithm of this paper.

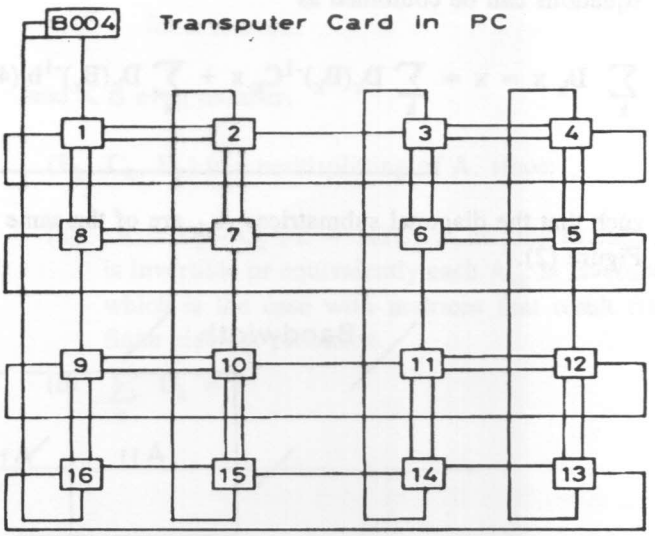

Figure 1: Transputer mesh architecture

The results for different linear systems with same bandwidth are then carefully discussed.

DESCRIPTION OF THE ALGORITHM

Consider the linear system of equations

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \qquad (1)$$

where **A** is a banded matrix of size ($n \times n$) and bandwidth w, **x** and **b** are the unknown and known vectors respectively. Based on the multisplitting of the matrix, an iterative method is presented which is structured so that operations can be performed in parallel.

Definition: Let A, B_k, C_k and D_k be $(n \times n)$ matrices, where $k = 1,2,\ldots,K$. Then (B_k, C_k, D_k) is called a multisplitting of A if

(i) $A = B_k - C_k$, $k = 1,2,\ldots,K$ where each B_k is invertible.

(ii) $\sum_k D_k = I$, where the matrices D_k are diagonal and $D_k \geq 0$.

Using (i) above, equation (1) may be written as

$$B_k x = C_k x + b \quad , k = 1,2,\ldots,K. \qquad (2)$$

or

$$x = (B_k)^{-1}C_k x + (B_k)^{-1}b \quad , k = 1,2,\ldots,K. \qquad (3)$$

Using the weighting matrices D_k, these K sets of equations can be combined as

$$\sum_k D_k x = x = \sum_k D_k(B_k)^{-1}C_k x + \sum_k D_k(B_k)^{-1}b \qquad (4)$$

Equation (4) yields the following algorithm:
Choose x_0 arbitrarily, then for i = 0,1,2,..... until convergence

$$x^{i+1} = H x^i + G b \qquad (5)$$

where

$$H = \sum_k D_k(B_k)^{-1}C_k \quad \text{and} \quad G = \sum_k D_k(B_k)^{-1}$$

The algorithm of equation (5) is implemented using a two stage splitting, which is described as follows: The system (1) can be written in the submatrix form

$$\begin{bmatrix} A_{11} & A_{12} & 0 & 0 & \ldots & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & \ldots & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & A_{K,K-1} & A_{KK} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_K \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_K \end{bmatrix}$$

such that the diagonal submatrices A_{kk} are of the same size $(m \times m)$ when k is odd and $(q \times q)$ when k is even, see Figure (2).

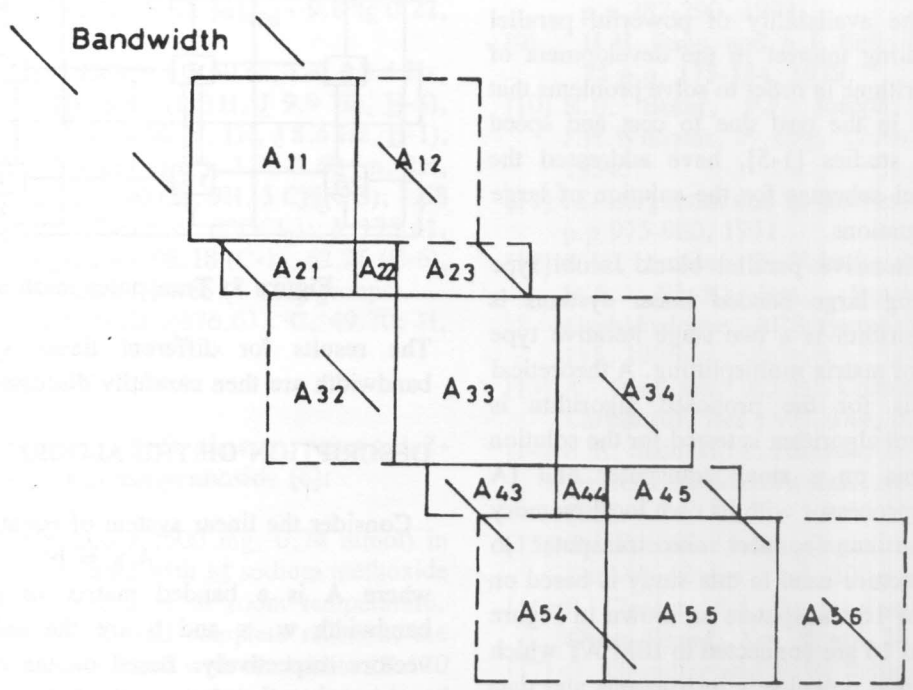

Figure 2: Two stages multisplitting.

Then the set of multisplitting matrices are

$$C_k = \begin{bmatrix} I' - A_{11} & -A_{12} & 0 & \cdots & & & & & 0 & 0 \\ -A_{21} & I'' - A_{22} & -A_{23} & \cdots & & & & & 0 & 0 \\ \vdots & & & & & & & & \vdots & \vdots \\ \vdots & & & & & & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -A_{k,k-1} & 0 & -A_{k,k+1} & \cdots & 0 & 0 \\ \vdots & & & & & & & & \vdots & \vdots \\ \vdots & & & & & & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & & & & & -A_{K,K-1} & I' - A_{KK} \end{bmatrix}$$

$$B_k = \begin{bmatrix} I' & 0 & \cdots & & 0 \\ 0 & I'' & \cdots & & 0 \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & A_{k,k} & 0 \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & & I' \end{bmatrix} \qquad D_k = \begin{bmatrix} 0 & 0 & \cdots & & 0 \\ 0 & 0 & \cdots & & 0 \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & \delta_k & 0 \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & & 0 \end{bmatrix}$$

$$\delta_k = \begin{cases} I' & \text{for k is odd} \\ I'' & \text{for k is even} \end{cases}$$

where I' and I'' are identity matrices of size $(m \times m)$ and $(q \times q)$ respectively,

and K is even number.

(B_k, C_k, D_k) is a multisplitting of A, since:

(a) $A = B_k - C_k$, $k = 1,2,\ldots,K$ as long as each B_k is invertible or equivalently each A_{kk} is invertible, which is the case with matrices that result from finite element problems.

(b) $\sum_k D_k = I$

Then

$$B_k^{-1} C_k = \begin{bmatrix} I' - A_{11} & -A_{12} & 0 & \cdots & & & & & 0 & 0 \\ -A_{21} & I'' - A_{22} & -A_{23} & \cdots & & & & & 0 & 0 \\ \vdots & & & & & & & & \vdots & \vdots \\ \vdots & & & & & & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -(A_{kk})^{-1} A_{k,k-1} & 0 & -(A_{kk})^{-1} A_{k,k+1} & \cdots & 0 & 0 \\ \vdots & & & & & & & & \vdots & \vdots \\ \vdots & & & & & & & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & & & & & -A_{K,K-1} & I' - A_{KK} \end{bmatrix}$$

and

$$H=\sum_k D_k(B_k)^{-1}C_k=\begin{bmatrix} 0 & -(A_{11})^{-1}A_{12} & 0 & \cdots & 0 & 0 \\ -(A_{22})^{-1}A_{21} & 0 & -(A_{22})^{-1}A_{23} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & -(A_{kk})^{-1}A_{k,k-1}\ 0\ -(A_{kk})^{-1}A_{k,k+1} & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -(A_{KK})^{-1}A_{K,K-1} & 0 \end{bmatrix}$$

$$G = \sum_k D_k\,(B_k)^{-1}$$

$$= \begin{bmatrix} (A_{11})^{-1} & 0 & \cdots & 0 \\ 0 & (A_{22})^{-1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & (A_{k,k})^{-1} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & (A_{KK})^{-1} \end{bmatrix}$$

Now that the matrices **H** and **G** are defined, the algorithm takes the following specific form:

Choose x_0 arbitrary, e.g. $x_0 = [0,0,\ldots\ldots,0]^T$.

Introduce subsequently the following iterative scheme

For i = 0,1,2,.... until convergence:

$$x^{i+1} = H\,x^i + G\,b \qquad (6)$$

For the subsystem k, this has the following form:

$$(x_k)^{i+1} = -(A_{kk})^{-1}\,A_{k,k-1}\,(x_{k-1})^i - (A_{kk})^{-1}\,A_{k,k+1}\,(x_{k+1})^i + (A_{kk})^{-1}\,b \qquad (7)$$

or

$$A_{kk}\,(x_k)^{i+1} = b_k - A_{k,k-1}\,(x_{k-1})^i - A_{k,k+1}\,(x_{k+1})^i$$

This method is essentially a tridiagonal Jacobi.

PARALLEL IMPLEMENTATION OF THE ALGORITHM

The above scheme can be implemented in a group of (K/2) processors. Each subsystem is transferred through the links between processors until each processor has two subsystem, i.e. processor 1 has the subsystems 1 and 2 while processor K/2 has the subsystems K-1 and K. All processors perform the first step of Gaussian elimination. For example, processor k brings the matrices $A_{2k-1,2k-1}$ and $A_{2k,2k}$ to upper triangular form and updates accordingly the matrices $A_{2k-1,2k-2}$, $A_{2k-1,2k}$, $A_{2k,2k-1}$, $A_{2k,2k+1}$ and the vectors b_{2k-1}, b_{2k}. This is computationally the most expensive operation but is performed only once. Back substitution is performed parallel in all processors with the odd numbered subsystems. The even numbered subsystems receive the solution from the odd numbered ones. Parallel, each processor updates the right hand-side vector of the even numbered subsystem then back substitution is performed parallel in all processors for only the even numbered subsystems. The solutions of the even numbered subsystems are communicated to the odd numbered ones and their right hand side vectors are updated. This is the end of the first iteration, where each processor obtains part of the solution vector. A convergence check after each iteration, based on the criterion $\| x^i - x^{i-1} \|_\infty <$ Tolerance, is carried out, where x^i is the solution of the i^{th} iteration and $\| \cdot \|_\infty$ is the infinity norm. If this is true the process is terminated.

In this parallel implementation two serial stages are combined in order to reduce the number of calculations than a possible one stage algorithm.

CONVERGENCE ANALYSIS

In the sequence $x^{i+1} = H x^i + G b$, **H** is called the iteration matrix. The necessary and sufficient condition of convergence is that all of the eigenvalues of H have a magnitude smaller than 1, that is, if the spectral radius $\rho(H)$ is smaller than 1. This is the most general possible convergence condition, but it is of limited use because the eigenvalues of H are rarely known exactly. Thus, a more refined tool is introduced based on suitable distance function(norm) from the desired point of convergence. Convergence is then guaranteed if each iteration reduces the value of this norm.

Definition: Given a vector $\omega > 0$, we define the weighted maximum norm $\| . \|_\omega$ by

$$\| x \|_\omega = \max_i | x_i/\omega_i |.$$

The vector norm $\| . \|_\omega$ induces a matrix norm, also denoted by $\| . \|_\omega$, defined as

$$\| A \|_\omega = \max_{x \neq 0} \frac{\| Ax \|_\omega}{\| x \|_\omega} \qquad (7)$$

where **A** is $n \times n$ matrix.

An alternative but equivalent definition of this norm is based upon assuming $\| x \|_\omega = 1$, then

$$\| A \|_\omega = \max_i \frac{1}{\omega_i} \sum_{j=1}^{n} | a_{ij} | \omega_j \qquad (8)$$

where a_{ij} are the entries of **A**.

Theorem (1): If **A** is an $n \times n$ nonegative matrix then, for every $\varepsilon > 0$ there exists some $\omega > 0$ such that

$$\rho(A) \leq \| A \|_\omega .$$

Proof: Let λ be an eigenvalue of **A** such that $|\lambda| = \rho(A)$. Let $x \neq 0$ be an eigenvector of **A** corresponding to the eigenvalue λ normalized so that $\| x \|_\omega = 1$. Then

$$\| A \|_\omega \geq \| Ax \|_\omega = \| \lambda x \|_\omega = | \lambda | \| x \|_\omega = \rho (A)$$

From the above theorem we can state that, if **A** is a square nonegative matrix such that $\rho(A) < 1$, then there exists some $\omega > 0$ such that $\| A \|_\omega < 1$.

Definition: A square block matrix **A** with blocks A_{ij} is(block) diagonally dominant if

$$\sum_{j \neq i} \| A_{ij} \|_\omega < \| A_{ii} \|_\omega$$

An equivalent definition is $\sum_{j \neq i} \| A_{ii}^{-1} A_{ij} \|_\omega < 1$.

Theorem (2): If **A** is a block diagonally dominant, then the block Jacobi method (6) converges.

Proof: From the definition of the iteration matrix **H**

$$H_{ij} = -(A_{ii})^{-1} A_{ij}$$

then

$$\| H_{ij} \|_\omega = \sum_{j \neq i} \| A_{ii}^{-1} A_{ij} \|_\omega < 1 \qquad i \neq j$$

Also, $H_{ii} = 0$, then we can write $\| H \|_\omega < 1$, and from theorem (1) we conclude that $\rho(H) < 1$ and hence the theorem is proved.

NUMERICAL RESULTS

The efficiency of a parallel algorithm is judged by its ability of significant speed gains as more processors participate in the solution of a problem. Table (1) gives the solution time as a function of the number of processors (p is the number of processors and N is the number of equations in the system solved).

Table 1. Solution Time (Sec), bandwidth = 71.

p\N	486	972	1458	1944
2	68.2	-	-	-
4	12.6	93.7	-	-
6	3.3	42.5	108.6	-
8	-	21.4	74.3	138.2
10	-	8.7	58.2	104.7
12	-	2.6	36.6	79.1
14	-	-	20.1	63.4
16	-	-	12.4	52.1

Four different systems were considered but all with the same bandwidth (w=71). It is interesting to note here that, unless the number of iterations is very large(say over 100), the solution times are not affected significantly. This is because the forward elimination (which is performed only once) is computationally much more expensive than the back substitution(which is performed in every iteration). In all cases presented in Table (1), back substitution took only a fraction of a second(less than 4 sec for the 10 to 15 iterations of the most computationally intensive cases such as 1944 equations on 12 processors). The observation of Table (1) may lead to a wrong conclusion that the speed-up and efficiency of the method are higher than they actually are. The parallel algorithm involves more calculations than the direct one-processor method for the same size system. As a result, some systems are solved faster with processor than with two or sometimes four processors. In the example of 972 equations, the solution of the system with 14 processors is over one hundred times faster than the solution with 4 processors, however it is only 15 times faster than the direct solution with one processor. Therefore, the speed-up provided by 14 processors is approximately 15, where the speed-up is defined as the ratio $s = T_s/T_p$ where T_s is the total time it takes for the best direct sequential method to solve the system on one processor and $T_p = T_{fw} + T_{bs} \bullet NIT$ is the total time spent in one processor in the parallel algorithm, where T_{fw} is the time spent for forward elimination, T_{bs} is the time spent for back substitution and NIT is the number of iterations performed to obtain the solution within a specific tolerance.

The efficiency of the algorithm is defined mathematically as e = s/K , where s is the speed-up and K is the number of processors. It is clear that the efficiency of the proposed algorithm is very low for small number of processors but it reaches values as high as 0.8 to 0.9 when 16 processors were used.

CONCLUSION

A parallel algorithm to solve banded systems of equations is presented in this study. The algorithm is based on the theory of matrix multisplitting. It is iterative in nature and implemented on a pipeline architecture consisting of 16 Transputers. Since the number of available processors is small, the proposed algorithm is presented as a two stage splitting which also performs best with thin systems(small bandwidth compared to size). Theoretical convergence criteria was presented and the results of a numerical experiment were discussed and it is concluded that the proposed algorithm can use parallel architectures very efficiently.

REFERENCES

[1] Y. Saad and M.H. Schultz, "Parallel direct methods for solving banded linear systems", *Lin. Algeb. & Appl. 88/89* , pp 623-650, 1987.

[2] I. Gohberg, T. Kailath, I. Koltracht and P. Lancaster, "Linear complexity algorithms for linear systems of equations with recursive structure ", *Lin. Algeb. & Appl.* 88/89, pp. 271-315, 1987.

[3] K. Datta, "Parallel Complexities and Computations of Cholesky's Decomposition and QR Factorization", *Int. J. comput. math.* ,18, pp. 67-82, 1985.

[4] S.L. Johnson, "Communication Efficient Basic Linear Algebra Computations on Hypercube architectures", *J. Parallel & Distr. Comput.*,4, pp. 133-172, 1987.

[5] S.L. Johnson, "Solving Tridiagonal Systems on Ensemble Architectures", *SIAM J. Sci. & Stat. Comput.*, 8, pp. 354-392, 1987.