Parallel Computing 4 (1987) 339–343
North-Holland

Short Communication

Parallel Gauss–Jordan elimination for the solution of dense linear systems *

Rami MELHEM

Department of Computer Science, The University of Pittsburgh, Pittsburgh, PA 15260, U.S.A.

Received September 1986

Abstract. Any factorization/back substitution scheme for the solution of linear systems consists of two phases which are different in nature, and hence may be inefficient for parallel implementation on a single computational network. The Gauss–Jordan elimination scheme unifies the nature of the two phases of the solution process and thus seems to be more suitable for parallel architectures, especially if reconfiguration of the communication pattern is not permitted. In this communication, a computational network for the Gauss–Jordan algorithm is presented. This network compares favorably with optimal implementations of the Gauss elimination/back substitution algorithm.

Keywords. Gauss–Jordan elimination, linear algebra, computational network, parallel solution of dense linear systems.

1. Introduction

Many researchers have considered the parallel solution of dense linear systems of equations on special purpose computational arrays [1–6]. A typical solution process consists of two phases: a factorization and partial solution phase and a back substitution phase. More specifically, given an $n \times n$ matrix A and an n-dimensional vector b, the solution of $Ax = b$ begins with (1) the decomposition of A into the product of, say, a lower triangular matrix L and an upper triangular matrix U and the simultaneous solution of $Ly = b$, which is then followed by (2) the solution of $Ux = y$ by back substitution.

Any attempt to execute these two phases on processor arrays will require either the use of two different arrays with a possible need for a form of interface between the arrays [5], or the use of a single array with reconfiguration capabilities [1]. In this latter case, only part of the array is doing actual computation during the second phase.

We suggest to unify the nature of the two phases by applying a Gauss–Jordan elimination scheme. For this, a network similar to the one suggested by Chen and Wu [1] is used without

* This work is, in part, supported under ONR contract N00014-85-K-0339.

any reconfiguration of the communication pattern. Of course the Gauss-Jordan scheme requires more computation than the factorization/back substitution scheme. But the additional work is performed by cells that would be otherwise idle [1], and hence no additional resources are required.

The idea of applying the Gauss–Jordan algorithm to the solution of dense linear systems on mesh-connected arrays was introduced by Kimura in [4]. However, the network suggested here is more efficient than the one in [4]. Namely, it is faster and it uses about half the number of cells. A brief quantitative comparison of the two networks is given in Section 4.

2. The Gauss–Jordan algorithm

In the following algorithm, no pivoting is used and each diagonal element in the matrix A is used to eliminate all the elements in the corresponding column. The right side vector b is considered as the column $n+1$ of A, and upon completion of the elimination, the solution vector x is stored in this last column.

Step 1. For $i = 1,\ldots,n$ Do
  1.1. $a_{i,j} = a_{i,j}/a_{i,i}$, $j = i+1,\ldots,n+1$
  1.2. For $k = 1,\ldots,i-1$ And $k = i+1,\ldots,n$ Do
    1.2.1. $a_{k,j} = a_{k,j} - a_{k,i}a_{i,j}$, $j = i+1,\ldots,n+1$

If the matrix A is a banded matrix with half-bandwidth β, then it is possible to replace the loop bound $k = i+1,\ldots,n$ in Step 1.2 by $k = i+1,\ldots,i+\beta$. However, the bound $k = 1,\ldots,i-1$ may not be changed because of the fill-in introduced in the upper triangle of A. This fill-in makes the Gauss–Jordan algorithm relatively unattractive for banded systems.

Assuming that the matrix A is dense, we denote by D_i the $n-i+1$ operations in Step 1.1, and by $U_{i,k}$ the $n-i+1$ operations in Step 1.2.1. In other words, $U_{i,k}$ represents the update of row k caused by contributions from row i, during the elimination of $a_{k,i}$.

3. The computational network

The network used here is composed of n stages, say $s = 1,\ldots,n$, each composed of $n-s+2$ cells. The network is shown in Fig. 1 where each cell is labeled by a pair (r, s) indicating its position with respect to the shown axes. Column s in Fig. 1 corresponds to stage s.

For simplicity, we assume that any diagonal cell (s, s) may broadcast data to the other cells in the same stage. However, it should be clear that this broadcast may be easily replaced by local vertical interconnections if the input data are skewed properly.

The elements of a row i of the matrix A enters the network at cycle i and crosses stages $s = 1,\ldots,i-1$, where at each stage s the operation $U_{s,i}$ takes place. When row i arrives at stage i (at cycle $2i-1$), cell (i, i) broadcasts $a_{i,i}$ to the other cells and the operation D_i is executed. The updated elements of row i are then stored in stage i for $n-1$ cycles during which rows $k = i+1,\ldots,n$ followed by rows $k = 1,\ldots,i-1$ cross stage i. The operation $U_{i,k}$ is executed when row k crosses row i. At the end of cycle $2i+n-1$, row i leaves stage i and crosses stages $i+1,\ldots,n$ where at each stage s, $U_{s,i}$ is executed. Clearly, row i comes out of stage n after $2n+i-1$ cycles with $a_{i,i} = 1$, $a_{i,j} = 0$ for $j = 1,\ldots,n$, $j \neq i$ and $a_{i,n+1} = x_i$. In other words, the outputs $x_1,\ldots,x_n$ are produced at cycles $2n,\ldots,3n-1$.

Figure 2 shows the operation performed by each stage at consecutive time units for the case $n = 6$. The arrows indicate the flow of data (rows of A) between stages, with a vertical arrow

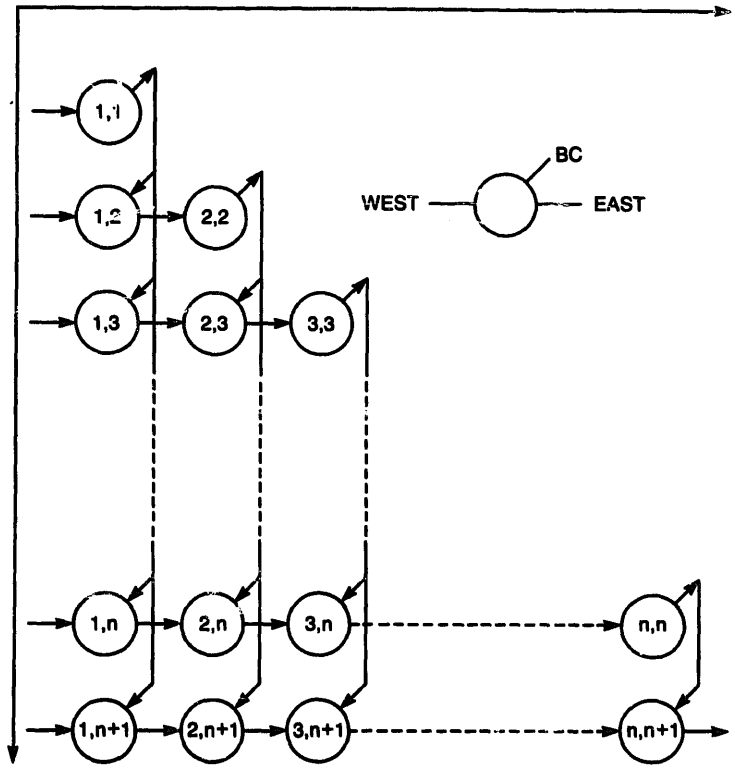

Fig. 1. The computational network.

indicating data that resides in the same stage. Note that row i resides in stage i during the cycles $2i-1,\ldots,2i+n-1$. Note also that the solution of different problems may be pipelined (as shown in the figure) at a rate of one problem every $n+1$ cycles. In general, this is not possible in networks which require a reconfiguration of the communication pattern.

In order to describe the operation of the individual cells, we give names to the ports connected to cells. Namely, WEST, EAST and BC, where BC is the port connected to the broadcast link. We also assume that each cell (i, j), $i \neq j$, contains a register R. The notation [port] and $[R]$ is used to denote the content of 'port' and 'R', respectively, and 'port $\leftarrow x$' and '$R \leftarrow x$' is used to indicate that x is written on 'port' and on 'R', respectively. With this notation, the operation of the individual cells may be expressed by the following algorithms, where the statement SYNC means 'wait until the end of the time unit'.

For cell (i, i), $1 \leqslant i \leqslant n$. /* *Start at time* $2i-1$ */
Step 1. For cycle $= 1,\ldots, n$ Do
1.1. BC $\leftarrow$ [WEST]; SYNC;
Step 2. SYNC; /* sit idle for one cycle */

For cell (i, j), $1 \leqslant i \leqslant n$, $i+1 \leqslant j \leqslant n+1$. /* *Start at time* $2i-1$ */
Step 1. $R \leftarrow$ [WEST]/[BC]; SYNC; /* [WEST] $= a_{j,i}$, [BC] $= a_{i,i}$ */
Step 2. For cycle $= 2,\ldots, n$ Do /* for $k = i+1,\ldots, n$, then $k = 1,\ldots, i-1$ */
2.1. EAST $\leftarrow$ [WEST] $-$ [BC]$*$[R]; /* [WEST] $= a_{k,j}$, [BC] $= a_{k,i}$ */
2.2. SYNC;
Step 3. EAST $\leftarrow$ [R]; SYNC; /* row i leaves stage i */

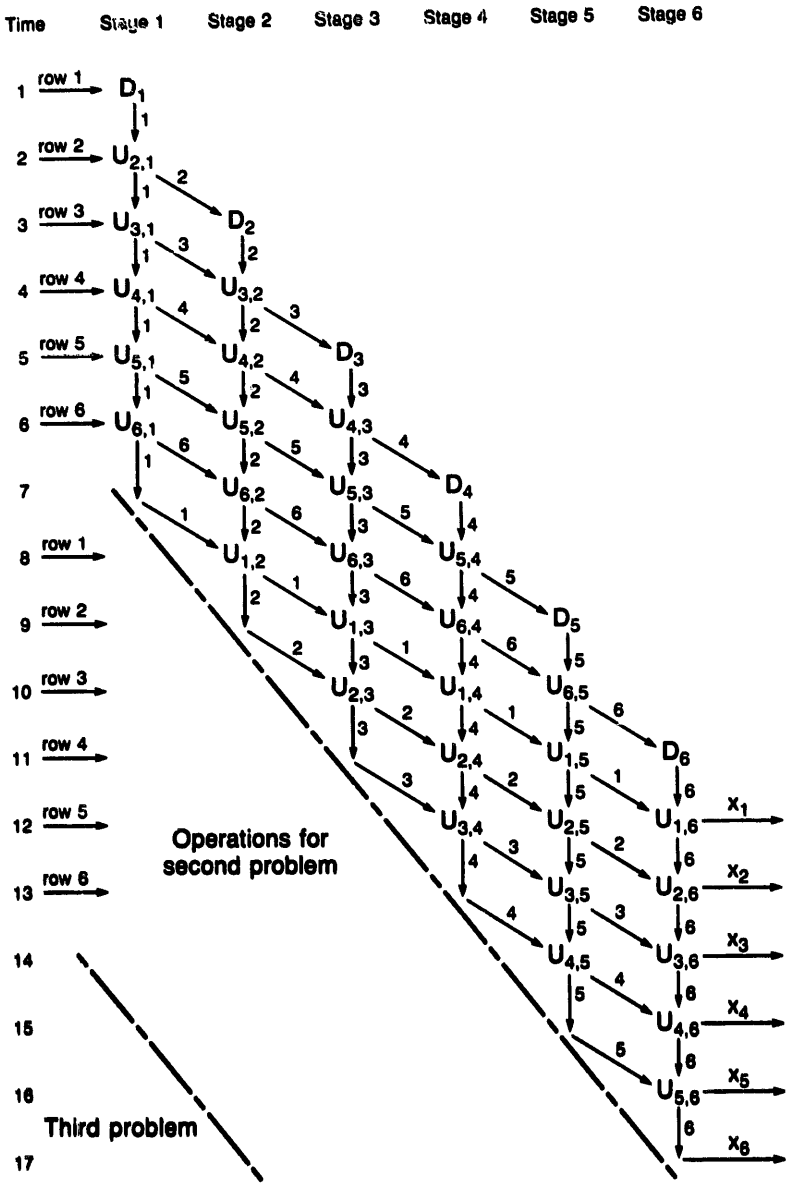

Fig. 2. Schedule of operations.

For pipelined operation, the above algorithms may be repeated as many times as desired. Also, partial or total pivoting may be accomplished in ways similar to those discussed in the literature (see the survey by Johnsson [3]).

4. Concluding remarks

The Gauss–Jordan elimination is an inefficient algorithm for the sequential solution of dense linear systems. However, it is readily seen that the parallel implementation of this algorithm, which we presented here, is as efficient as the optimal parallel implementation of the Gauss elimination/back substitution scheme suggested in [1]. Moreover, our implementation has some additional advantages, namely,

(1) it does not require a reconfiguration of the network,

(2) it does not leave the results inside the network,

(3) it may be pipelined efficiently, and
(4) broadcast may be eliminated with minimal additional costs.

The network which is described here for the Gauss–Jordan algorithm uses $\frac{1}{2}n(n+3)$ cells and the operation of each cell may be easily controlled by a counter which keeps track of the cycle number and triggers the switching between the three steps of the algorithm within each cell. This network is more efficient than the one suggested in [4], which uses $n(n+1)$ cells and uses three local registers per cell to switch between seven possible steps. Also our network completes execution in $3n-1$ cycles while the one in [4] requires $4n$ cycles. However, a more objective comparison of speeds may be obtained if the broadcast is removed from our network, thus increasing its execution time to $4n-1$ cycles and the preloading of the operands and the unloading of the results are accounted for in the network of [4], thus increasing its execution time to $5n+1$ cycles.

Acknowledgment

I would like to thank Robert Voigt for pointing out reference [4] to me.

References

[1] A. Chen and C. Wu, Optimum solution to dense linear systems of equations, *Proc. 1984 International Conference on Parallel Processing* (1984) 417–424.
[2] K. Hwang and Y. Cheng, VLSI computing structures for solving large-scale linear systems of equations, *Proc. 1980 International Conference on Parallel Processing* (1980) 217–227.
[3] L. Johnsson, Highly concurrent algorithms for solving linear systems of equations, *Monterey Conference on Elliptic Problem Solvers* (1983).
[4] T. Kimura, Gauss–Jordan elimination by VLSI mesh-connected processors, in: C. Josshope and R. Hockney, eds., *Infotech State of the Art Report: Supercomputers, Vol. 2* (Infotech, Maidenhead, United Kingdom, 1979) 271–290.
[5] H.T. Kung and C. Leiserson, Systolic arrays for VLSI, in: C. Mead and L. Conway, eds., *Introduction to VLSI Systems* (Addison-Wesley, Reading, MA, 1980).
[6] S.Y. Kung, K.S. Arun, R.J. Gab-ezer and B. Rao, Wavefront array processor: Language, architecture and applications, *IEEE Trans. Comput.* **31** (1982) 1054–1066.