Proceedings of the ISCA
International Conference

PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

Las Vegas, Nevada U.S.A.
October 6-8, 1994

A Publication of
The International Society for
Computers and Their Applications - ISCA

ISBN: 1-880843-09-9

Solving Block Bidiagonal Systems on Distributed Memory Computers

P. Amodio T. Politi
Dipartimento di Matematica
Università di Bari
Bari, 70125, Italy

M. Paprzycki
Department of Math. and Computer Sci.
Univerity of Texas of the Permian Basin
Odessa, Texas 79762, USA

Abstract

Two parallel algorithms for solving block bidiagonal linear systems on distributed memory computers are introduced. The first is a simple modification of the sequential algorithm and is suitable for a small number of processors. The second is based on parallel methods for banded systems and is much better suited for parallel computations. Experimental results from a linear array of 30 Transputers are presented.

1 Introduction

In many application areas, matrices arise with exploitable sparse block structure. For example, several numerical methods for the solution of ODEs, PDEs and BVPs lead to matrices with only few nonzero block-diagonals [2, 6].

We consider the parallel solution of a linear system

$$Ax = b \tag{1}$$

where the coefficient matrix A has a block bidiagonal structure

$$\begin{pmatrix} D_1 & & & \\ C_2 & D_2 & & \\ & \ddots & \ddots & \\ & & C_n & D_n \end{pmatrix} \tag{2}$$

with D_i and C_i dense blocks of size m (D_i is nonsingular for $i = 1, \ldots, n$), and

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Although this problem has not been extensively considered in the literature, its solution is the kernel of a number of mathematical applications. The known approaches come from the solution of almost block diagonal (ABD) systems arising from particular discretizations of ODEs with boundary conditions (see [6]). After temporarily neglecting boundary conditions, the resulting block bidiagonal system is solved by parallel reduction algorithms. Since ABD matrices differ from block bidiagonal matrices by a block row (or column) in the upper triangular portion, these solvers produce parallel factorizations with both lower and upper fill-in vectors. In our case, it is possible to develop algorithms that do not destroy the block triangular structure of the coefficient matrix. Moreover, these algorithms include row pivoting which should be sufficient to guarantee the stability of the solution [3].

The following is the generic sequential algorithm for the solution of (1):

```
$P_1 L_1 U_1 = D_1$
$x_1 = D_1^{-1} b_1$
for $i = 2, n$
  $P_i L_i U_i = D_i$
  $x_i = D_i^{-1}(b_i - C_i x_{i-1})$
end
```

The algorithms presented in this paper require more arithmetical operations than the sequential one but they may be used on a large number of processors. All of the proposed algorithms will be based on the following decomposition of matrix A (2):

$$\begin{pmatrix} A^{(1)} & & & & & & \\ c_2^{(1)^T} & D_k & & & & & \\ & c_1^{(2)} & A^{(2)} & & & & \\ & & c_2^{(2)^T} & D_{2k} & & & \\ & & & \ddots & \ddots & & \\ & & & & c_1^{(p)} & A^{(p)} \end{pmatrix}$$

where $n = kp - 1$, p is the number of processors involved, e_i denotes a block vector of length $k-1$ with only the ith block equal to the identity matrix, $c_1^{(i)} = C_{ik}e_{k-1}$, $c_2^{(i)} = C_{(i-1)k+1}e_1$, and

$$A^{(i)} = \begin{pmatrix} D_{(i-1)k+1} & & & \\ C_{(i-1)k+2} & D_{(i-1)k+2} & & \\ & \ddots & \ddots & \\ & & C_{ik-1} & D_{ik-1} \end{pmatrix}.$$

Except for the last, each processor stores two block rows of the partitioning of A (i.e. the ith processor contains the block rows with $A^{(i)}$ and D_{ik}, and the last contains the block row with $A^{(p)}$).

2 Quasi-sequential algorithm

The first algorithm we consider is a simple modification of the sequential one.

The most expensive part of the sequential algorithm is the inversion of the main diagonal blocks D_i, the LU factorization of which may be performed in parallel. Then the algorithm continues in the following way: the first processor solves its part of the system and sends the last block component of the solution to the second processor. The second processor, waiting for the data from the first, scales its first q_2 block equations (where q_2 is a positive integer less than or equal to k). When it receives the data, it updates on the first q_2 block equations, solves the remaining equations and sends its last block of the solution to the third processor. Each processor j performs the same operations as processor two; waiting for the data from processor $(j-1)$, it scales the first q_j block equations (obviously the value of q_j should be proportional to j), then it solves the equations and sends the data to processor $j+1$. The algorithm for processor j, can be thus summarized as follows:

```
for $i = (j-1)k + 1, jk$
  $P_i L_i U_i = D_i$
end
for $i = (j-1)k + 1, (j-1)k + q_j$
  $E_i = D_i^{-1} C_i$
  $g_i = D_i^{-1} b_i$
end
receive $x_{(j-1)k}$ from processor $j-1$
for $i = (j-1)k + 1, (j-1)k + q_j$
  $x_i = g_i - E_i x_{i-1}$
end
for $i = (j-1)k + q_j + 1, jk$
  $g_i = b_i - C_i x_{i-1}$
  $x_i = D_i^{-1} g_i$
end
send $x_{jk}$ to processor $j+1$
```

This algorithm has the typical characteristics of the sequential algorithm, such as the sequential communication between processors (each processor waits for data from another processor before sending its data to the next one). This does not make it very efficient especially if the number of processors is large relative to the block sizes (see Section 4). At the same time, however, it has some advantages: it does not produce fill-in vectors and requires only vector transmissions. If a large number of processors is available (and if each of these processors has a substantial local memory), it is possible to tune the algorithm performance by gradually increasing the number of blocks stored in further processors.

3 Parallel factorization algorithm

The second algorithm is based on the domain decomposition methods already used in [1] to derive parallel factorizations of tridiagonal matrices, in [4] to solve ABD systems and in [5] to solve linear recurrence systems. The matrix A is factorized as:

$$A = N \cdot Q, \qquad (3)$$

where

$$N = \begin{pmatrix} A^{(1)} & & & & \\ c_2^{(1)^T} & D_k & & & \\ & o_{k-1} & A^{(2)} & & \\ & & c_2^{(2)^T} & D_{2k} & \\ & & & \ddots & \ddots \end{pmatrix},$$

$$Q = \begin{pmatrix} I_{mk} & & & & \\ o_{k-1}^T & I_m & & & \\ & v^{(2)} & I_{mk} & & \\ & V_{2k} & o_{k-1}^T & I_m & \\ & & & \ddots & \ddots \end{pmatrix},$$

and the unknown elements of V_{jk} and $v^{(j-1)} = (V_{(j-1)k+1}^T \cdots V_{jk-1}^T)^T$ are obtained from (3) by direct identification:

$$\begin{aligned} v^{(j)} &= (A^{(j)})^{-1} c_1^{(j)} && \text{for } j = 2, \dots, p, \\ V_{jk} &= -D_{jk}^{-1} c_2^{(j)^T} v^{(j)} && \text{for } j = 2, \dots, p-1. \end{aligned} \qquad (4)$$

The factorization (3), the solution of N and the updating of the right hand side by $v^{(i)}$ are performed in parallel with no data transmission between processors. Data transmissions are only required when solving the reduced linear system with coefficient matrix:

$$\begin{pmatrix} I_m & & & & \\ V_{2k} & I_m & & & \\ & V_{3k} & I_m & & \\ & & \ddots & \ddots & \\ & & & V_{(p-1)k} & I_m \end{pmatrix} \qquad (5)$$

and unknowns $x_k, x_{2k}, \ldots, x_{(p-1)k}$. The following is the algorithm for the generic processor j:

```
for $i = (j-1)k+1, jk$
   $P_i L_i U_i = D_i$
end
$V_{(j-1)k+1} = D^{-1}_{(j-1)k+1} C_{(j-1)k+1}$
$b_{(j-1)k+1} = D^{-1}_{(j-1)k+1} b_{(j-1)k+1}$
for $i = (j-1)k+2, jk$
   $V_i = -D_i^{-1} C_i V_{i-1}$
   $b_i = D_i^{-1}(b_i - C_i b_{i-1})$
end
compute $x_{(j-1)k}$ and $x_{jk}$ by solving
   the reduced system
for $i = (j-1)k+1, jk-1$
   $x_i = b_i - V_i x_{(j-1)k}$
end
```

As far as the reduced system on a distributed memory parallel computer with a large number of processors is concerned, its solution may be quite expensive. Each processor (except the last) has one block-row of the matrix (5) and the corresponding elements of the right hand side. To reduce the communication overhead we suggest the following algorithm for each processor j (assuming that processor j performs the ith send operation if $j + 2^i \le p$ and the ith receive operation if $j - 2^i \ge 1$):

```
for $i = 0, \lceil \log_2(p-1) \rceil - 2$
   send $b_{jk}$ and $V_{jk}$ to processor $j + 2^i$
   receive $b_{(j-2^i)k}$ and $V_{(j-2^i)k}$ from
      processor $j - 2^i$
   $b_{jk} = b_{jk} - V_{jk} b_{(j-2^i)k}$
   $V_{jk} = -V_{jk} V_{(j-2^i)k}$
end
$i = \lceil \log_2(p-1) \rceil - 1$
send $b_{jk}$ to processor $j + 2^i$
receive $b_{(j-2^i)k}$ from processor $j - 2^i$
$x_{jk} = b_{jk} - V_{jk} b_{(j-2^i)k}$
send $x_{jk}$ to processor $j + 1$
```

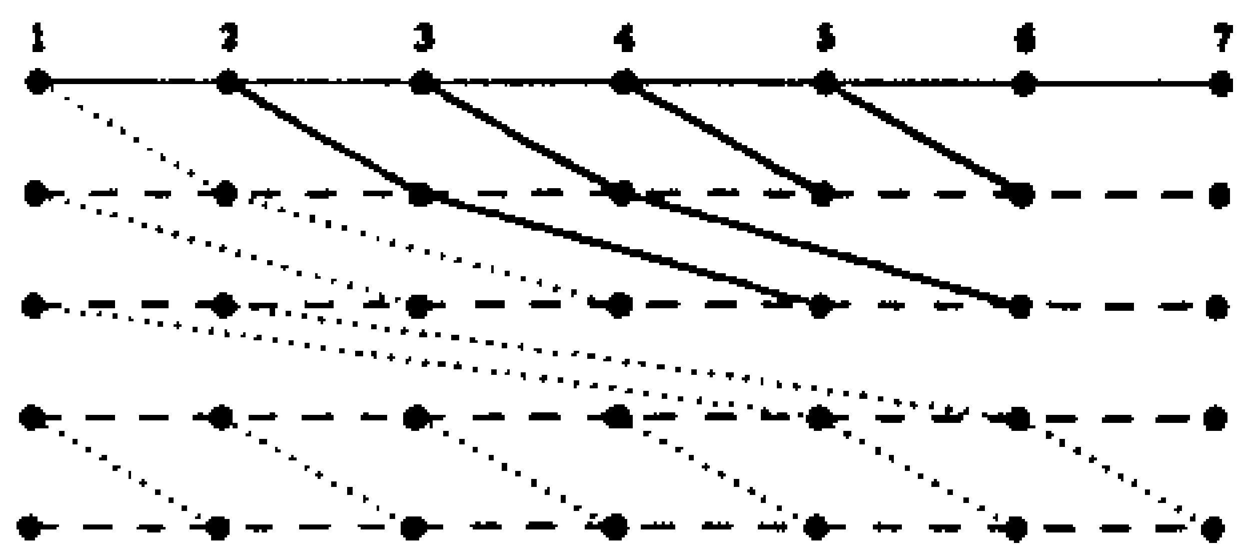

Figure 1: *Communications needed for the solution of the reduced system on 7 processors. With the solid line we express transmissions of vectors and matrices. With the dotted line transmissions of vectors.*

```
receive $x_{(j-1)k}$ from processor $j - 1$
```

Figure 1 shows the communication needed for this approach on $p = 7$ processors. Observe that if $p > 1$, the solution of the reduced system requires on each processor at most $\lceil \log_2(p-1) \rceil - 1$ matrix transmissions and $\lceil \log_2(p-1) \rceil + 1$ vector transmissions.

4 Numerical Tests

The two algorithms presented in the previous sections have been coded in Parallel Fortran [7] with the Express communication library [8] on a MicroWay Multiputer, which has a network of 30 transputers T800–20, each one with a local memory of 1 Mb. The sequential algorithm has been implemented in Fortran on a single transputer T800–20 with 16 Mb of memory. The codes use the level 3-BLAS routines DGEMM to perform matrix-matrix products, DGEMV to perform matrix-vector products, and two subroutines of LINPACK package: DGEFA, which factorizes the main diagonal blocks with only local row pivoting, and DGESL, which solves triangular linear systems factorized by DGEFA. Presented speedups are calculated against the sequential algorithm and all the results are averages of multiple runs.

We have chosen general nonsingular block bidiagonal systems with dense blocks. In Figure 2–4 the measured speedups of parallel factorisation and quasi-sequential algorithms are reported for $p = 4$, $p = 15$ and $p = 30$ processors, and different values of m and $k = \lfloor 800/m \rfloor$. For the quasi-sequential algorithm we have chosen $q_j = (j-1)q$, (where q is a positive integer number fixed for all processors, j is the number of the processor, $j = 1, \ldots, p$) in such a way that the speedup is the highest.

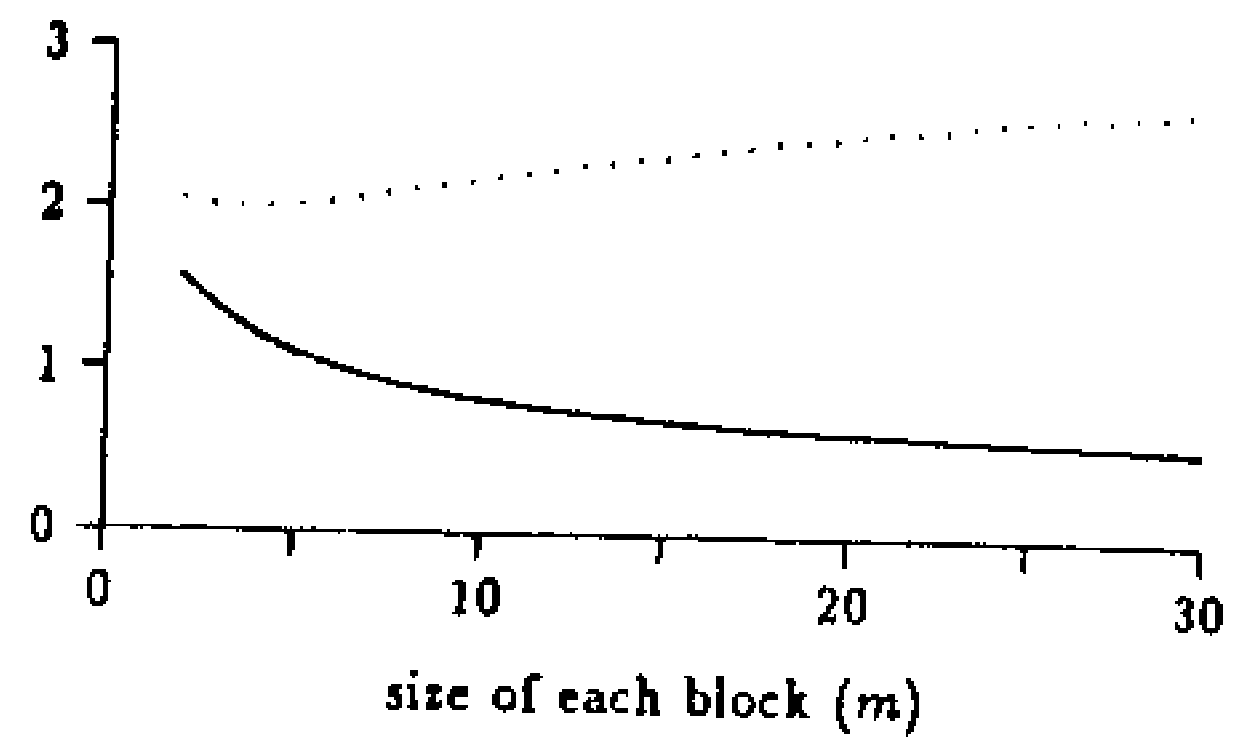

Figure 2: *Speedups of the algorithms for $p = 4$ and m and $k = \lfloor 800/m \rfloor$ variables. Solid line for parallel factorization, and dotted line for quasi-sequential algorithm.*

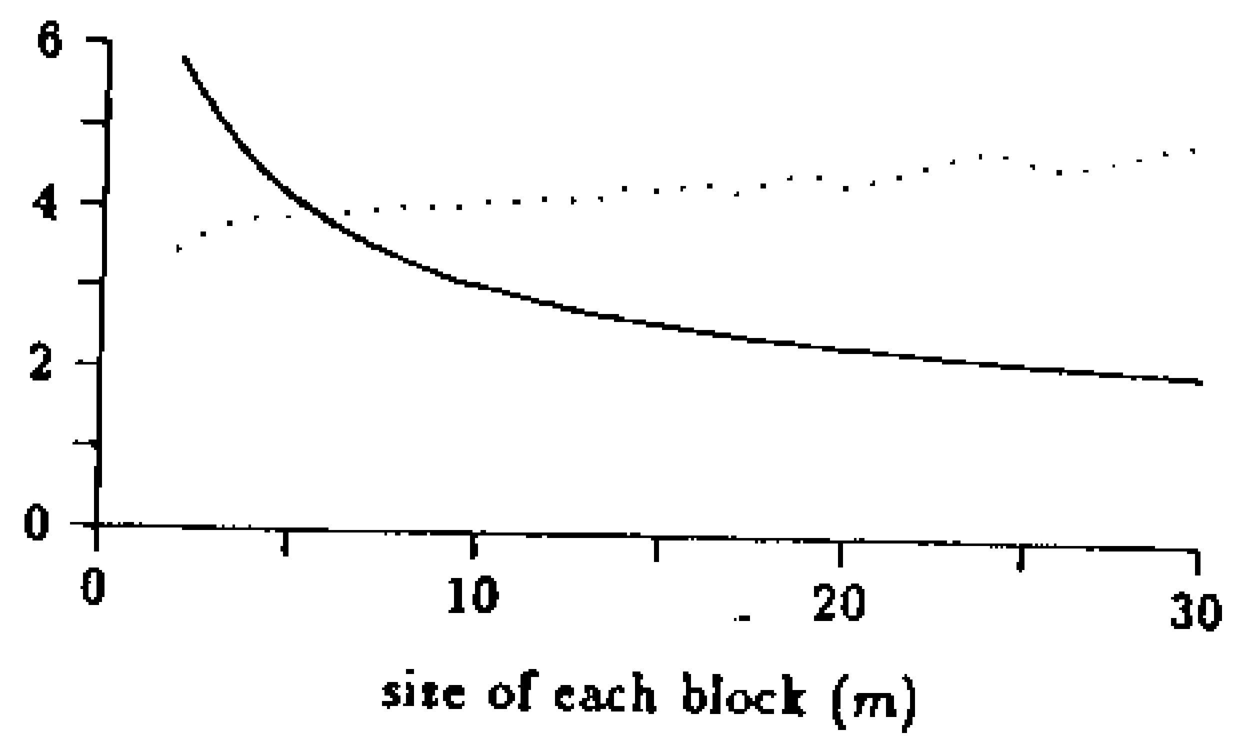

Figure 3: *Speedups of the algorithms for $p = 15$ and m and $k = \lfloor 800/m \rfloor$ variables. Solid line for parallel factorization, and dotted line for quasi-sequential algorithm.*

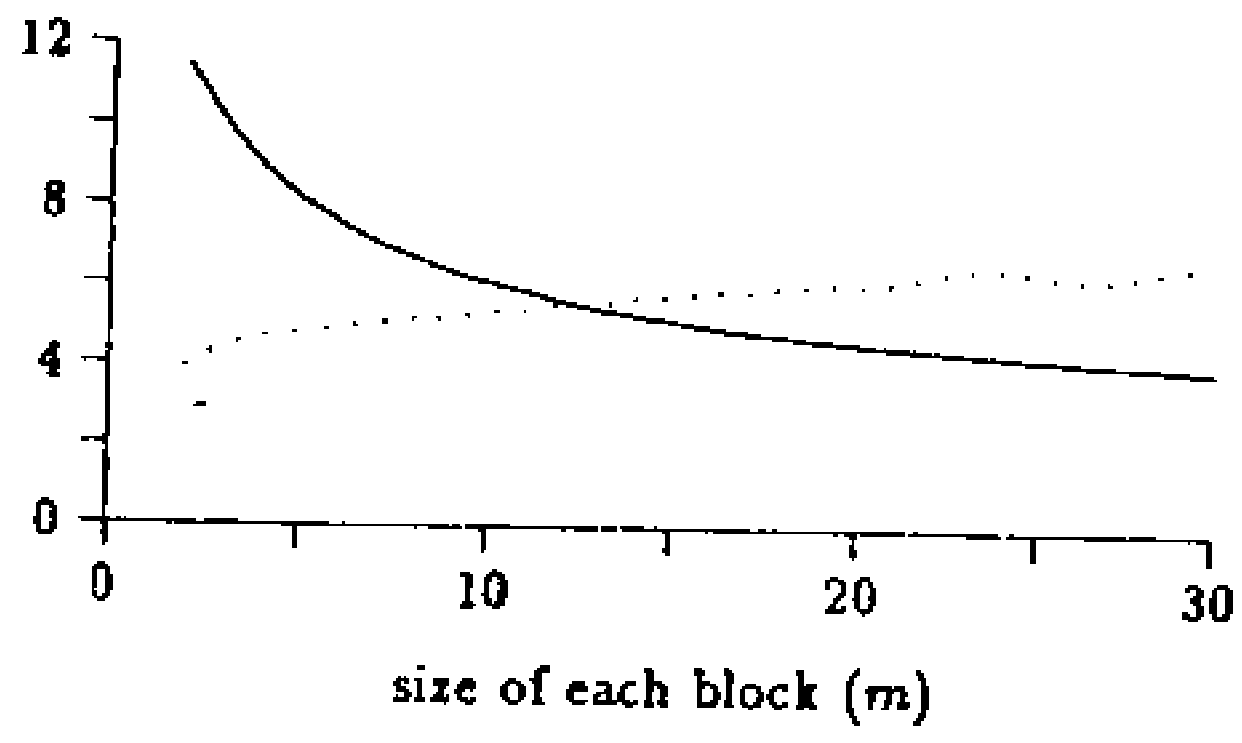

Figure 4: *Speedups of the algorithms for $p = 30$ and m and $k = \lfloor 800/m \rfloor$ variables. Solid line for parallel factorization, and dotted line for quasi-sequential algorithm.*

We observe that when using a small number of processors, the quasi-sequential algorithm has good speedups and is always better than the parallel factorisation. This second algorithm becomes preferable when a larger number of processors is available. Unfortunately we are not able to use a number of processors greater than 30, but the experimental results suggest that for $p > 2m$ the parallel factorisation algorithm leads to better speedups than the quasi-sequential one.

References

[1] P. Amodio, L. Brugnano, T. Politi, "Parallel Factorizations for Tridiagonal Matrices", *SIAM J. Numer. Anal.*, Vol. 30, pp. 813-823, 1993.

[2] I. Gladwell, M. Paprzycki, "Parallel Solution of Almost Block Diagonal Systems on the CRAY Y-MP using level 3 BLAS", *J. Comput. Appl. Math.*, Vol. 45, pp. 181-189, 1993.

[3] G. H. Golub, C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1989.

[4] M. Paprzycki, I. Gladwell, "Solving Almost Block Diagonal Systems on Parallel Computers", *Parallel Comput.*, Vol. 17, pp. 133-153, 1991.

[5] M. Paprzycki, P. Stpiczyński, "Solving Linear Recurrence Systems on Parallel Computers", in: R. Kalia, P. Vashishta (eds.) *Proceedings of the Mardi Gras '94 Conference*, Baton Rouge, Feb. 10-12, 1994, Nova Science Publishers, New York, 1994 (to appear).

[6] S. J. Wright, "Stable Parallel Algorithms for Two-Point Boundary Value Problems", *SIAM J. Sci. Statist. Comput.*, Vol. 13, pp. 742-764, 1992.

[7] *Parallel Fortran User Guide*, 3L Ltd., Livingstone, 1988.

[8] *Express Fortran User's Guide*, ParaSoft Corp., Pasadena, 1990.