Computers Math. Applic. Vol. 31, No. 7, pp. 111–127, 1996

0898-1221/96 $15.00 + 0.00

A Survey of Parallel Direct Methods for Block Bidiagonal Linear Systems on Distributed Memory Computers

P. AMODIO*
Dipartimento di Matematica, Università di Bari
Via Orabona 4, 70125 Bari, Italy
001105700vm.csata.it

M. PAPRZYCKI
Department of Mathematics and Computer Science
University of Texas of the Permian Basin, Odessa, TX 79762, U.S.A.
paprzycki_m@gusher.pb.utexas.edu

T. POLITI
Dipartimento di Matematica, Politecnico di Bari
Via Orabona 4, 70125 Bari, Italy
pptt@max.uniba.it

(Received February 1995; accepted May 1995)

Abstract—Four parallel algorithms for the solution of block bidiagonal linear systems on distributed memory computers are presented. All the algorithms belong to the class of direct methods. The first is a variant of the sequential algorithm and is suitable for a small number of processors. The remaining three algorithms are based on the parallel methods for banded systems and are much better suited for parallel computations on multiple processors. The arithmetical complexity functions of the proposed algorithms are derived. The results of experiments with the four algorithms implemented in Parallel Fortran on a linear array of 32 Transputers are presented and discussed.

Keywords—Block bidiagonal systems, Parallel algorithms.

1. INTRODUCTION

Matrices with exploitable sparse block structure arise in many applications. For example, several numerical methods for the solution of ODEs, PDEs and BVPs lead to block matrices with only few non-null diagonals [1–3].

We consider the parallel solution of a linear system

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

where the coefficient matrix A has a block bidiagonal structure

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

We would like to thank the anonymous referees for their useful comments leading to a number of improvements in the paper. We would also like to express our thanks to K. Paprzycka for her help with the English.
*Work supported by the Ministero della Ricerca Scientifica, 60% project.

with D_i and C_i dense blocks, and

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_n \end{pmatrix}.$$

It is also assumed that for $i = 1, \ldots, n$, each D_i is a nonsingular square block of size m_i.

This problem has not been extensively considered in the literature, even though it is the kernel of several differential equation solvers. For example, a block bidiagonal system could be used as preconditioning when an initial value problem is solved by means of a Boundary Value Method (see [4,5]).

The known approaches come from the solution of almost block diagonal (ABD) systems arising from particular discretizations of boundary value ODEs (see [2,3,6]). In this case, 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 approach, the introduction of fill-in may be substantially reduced. For one of the presented algorithms it is even possible to eliminate the fill-in altogether, even though local row pivoting is used in all cases to assure the stability of the algorithms (see [7]).

The generic sequential algorithm for the solution of (1) has the form:

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

On shared memory parallel computers with a limited number of processors, the parallelization may be introduced inside the factorization of each D_i and inside the calculation of x_i's (similarly to the level 3 BLAS based algorithm suggested in [8] for ABD systems). This method is successful only when the size of the blocks is large. At the same time, because of the communication overhead, it is not well suited for distributed memory parallel computers. For the distributed memory computers it is more important to reduce the communication costs than the operation count. The algorithms presented in this paper have an higher overall computational cost with respect to the sequential algorithm, but they may be executed on a large number of processors. All of the proposed algorithms will be based on the following decomposition of the matrix A (2) (supposing $n = kp - 1$, where p is the number of processors involved):

$$A = \begin{pmatrix} A^{(1)} & & & & & & \\ C_k \boldsymbol{e}_{k-1}^\top & D_k & & & & & \\ & \boldsymbol{e}_1 C_{k+1} & A^{(2)} & & & & \\ & & C_{2k} \boldsymbol{e}_{k-1}^\top & D_{2k} & & & \\ & & & \ddots & \ddots & & \\ & & & & C_{(p-1)k} \boldsymbol{e}_{k-1}^\top & D_{(p-1)k} & \\ & & & & & \boldsymbol{e}_1 C_{(p-1)k+1} & A^{(p)} \end{pmatrix},$$

where $\boldsymbol{e}_i$ denotes a block vector of length $k-1$ with the i^{th} block equal to an identity matrix and the others equal to zero matrices (e.g., if $m_i = m$ for all i, $\boldsymbol{e}_1$ consists of an identity matrix of

size m followed by a block of zeroes of size $(k-2)m \times m$, and $\boldsymbol{e}_{k-1}$ consists of a block of zeroes of size $(k-2)m \times m$ followed by an identity matrix of size m), 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 2 block rows of the partitioning of A (e.g., processor i contains the block rows with $A^{(i)}$ and D_{ik}, and the last processor contains the block row with $A^{(p)}$). Section 2 presents a modification of the sequential algorithm that is suitable for a parallel computer with a small number of processors. Sections 3 and 4 introduce the two algorithms based on different parallel factorizations. Section 5 is devoted to the generalized cyclic reduction algorithm. Section 6 contains the study of the arithmetical complexity functions and the memory requirements of the proposed algorithms. In Section 7 the results of experiments on a network of 32 transputers are presented and discussed.

2. QUASI-SEQUENTIAL ALGORITHM

The first algorithm we consider is a simple modification of the sequential one. First, the LU decomposition is used in parallel to invert the main diagonal blocks D_i (this is the most expensive part of the sequential algorithm). 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 number less than or equal to k). When it receives the data, it updates the first q_2 block equations, solves the remaining equations and sends its last block of the solution to the next processor. Each processor j performs the same operations as the second processor; waiting for a vector from processor $j-1$, it scales the first q_j block equations (obviously the value of q_j must be proportional to j, see Sections 6 and 7), 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
    determine P_i, L_i and U_i such that 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 is characterized by sequential communication between processors (each processor waits for data from another processor before sending its data to the next one). If the number of processors is relatively large in comparison to the block sizes, this does not make it very efficient (see Sections 6 and 7). At the same time, the algorithm has some advantages: it does

not produce fill-in vectors and requires only vector transmissions. If a very large system is to be solved on a machine with a large number of processors (and if each of these processors has a substantial local memory), it is possible to improve the algorithm by gradually increasing the number of blocks stored in later processors.

3. REDUCTION ALGORITHM

The second algorithm is similar to the approach used by Brugnano in [9] to solve tridiagonal linear systems and by Ascher and Chan in [6] to solve ABD systems. This algorithm leads to the solution of a reduced block system of size p in which the unknowns are located in the first block component of the solution in each of the processors. It may be considered as a variant of the sequential algorithm in the sense that we calculate the first block of the vector solution in each processor in order to solve the remaining system using the sequential solver. First we scale the matrix A in order to obtain identity matrices on the main block diagonal. Then supposing we know the first block component of the solution in each processor j, we can express, by using the sequential algorithm iteratively, each vector $\boldsymbol{x}_{jk+i}$, for $i = 2, \ldots, k$, as a function of $\boldsymbol{x}_{jk+1}$, for $j = 0, \ldots, p-1$. Hence, always using the same recursion, we relate $\boldsymbol{x}_{jk+1}$ to $\boldsymbol{x}_{(j-1)k+1}$, for $j = 1, \ldots, p-1$, obtaining a block bidiagonal linear system of size p with unitary block main diagonal

$$\begin{pmatrix} I_1 & & & & \\ T_2 & I_{k+1} & & & \\ & T_3 & I_{2k+1} & & \\ & & \ddots & \ddots & \\ & & & T_p & I_{(p-1)k+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_{k+1} \\ \boldsymbol{x}_{2k+1} \\ \vdots \\ \boldsymbol{x}_{(p-1)k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d}_1 \\ \boldsymbol{d}_2 \\ \boldsymbol{d}_3 \\ \vdots \\ \boldsymbol{d}_p \end{pmatrix}, \tag{3}$$

where I_j represents the identity matrix of order m_j,

$$\begin{aligned} T_j &= (-1)^{k+1} \prod_{i=1}^{k} E_{(j-2)k+2-i}, && \text{for } j = 2, \ldots, p, \\ \boldsymbol{d}_1 &= \boldsymbol{g}_1, \\ \boldsymbol{d}_j &= \boldsymbol{g}_{(j-1)k+1} - \sum_{i=(j-2)k+2}^{(j-1)k} \left(\prod_{l=0}^{(j-1)k-i} (-E_{(j-1)k-l+1}) \right) \boldsymbol{g}_i, && \text{for } j = 2, \ldots, p, \end{aligned}$$

E_i are the subdiagonal blocks of the scaled matrix, and $\boldsymbol{g}_i$ are the corresponding right-hand side vectors.

After the reduced system has been solved, the other components of the solution are easily obtained in parallel by using the sequential algorithm. The following summarizes the algorithm for a generic processor j:

```
for i = (j-1)k+1, jk
    determine P_i, L_i and U_i such that P_i L_i U_i = D_i
    E_i = D_i^{-1} C_i
    g_i = D_i^{-1} b_i
end
T_j = E_{(j-1)k+2}
d_j = g_{(j-1)k+2}
for i = (j-1)k+3, jk
    T_j = -E_i T_j
    d_j = g_i - E_i d_j
end
send T_j and d_j to processor j+1
```

```
receive T_{j-1} and d_{j-1} from processor j - 1
T_j = -E_{(j-1)k+1} T_{j-1}
d_j := g_{(j-1)k+1} - E_{(j-1)k+1} d_{j-1}
obtain x_{(j-1)k+1} by solving the reduced system
for i = (j - 1)k + 2, jk
    x_i = g_i - E_i x_{i-1}
end
```

This algorithm can also be expressed in terms of the factorization of matrix $D^{-1}A$, where D contains the main block diagonal elements of A:

$$D^{-1}A = N \cdot Q, \tag{4}$$

where

$$N = \begin{pmatrix}
I^{(1)} & & & & & & & \\
o_{k-1}^{\top} & I_k & & & & & & \\
e_1(w^{(2)})^{\top} & e_1 E_{k+1} & I^{(2)} & & & & & \\
 & & o_{k-1}^{\top} & I_{2k} & & & & \\
 & & & \ddots & \ddots & & & \\
 & & & & & o_{k-1}^{\top} & I_{(p-1)k} & \\
 & & & & & e_1(w^{(p)})^{\top} & e_1 E_{(p-1)k+1} & I^{(p)}
\end{pmatrix},$$

and

$$Q = \begin{pmatrix}
R^{(1)} & & & & & & & \\
E_k e_{k-1}^{\top} & I_k & & & & & & \\
 & o_{k-1} & R^{(2)} & & & & & \\
 & & E_{2k} e_{k-1}^{\top} & I_{2k} & & & & \\
 & & & \ddots & \ddots & & & \\
 & & & & o_{k-1} & R^{(p-1)} & & \\
 & & & & & E_{(p-1)k} e_{k-1}^{\top} & I_{(p-1)k} & \\
 & & & & & & o_{k-1} & R^{(p)}
\end{pmatrix}.$$

The block vector $w^{(j)}$ is of length $k-1$:

$$w^{(j)} = \left(T_j, (-1)^k \prod_{i=1}^{k} E_{(j-2)k+2-i}, \ldots, -E_{(j-1)k+1} E_{(j-1)k} \right)^{\top},$$

$$R^{(j)} = (D^{(j)})^{-1} A^{(j)}, \qquad \text{for } j = 1, \ldots, p,$$

where $D^{(j)}$ is the following block diagonal matrix:

$$D^{(j)} = \begin{pmatrix}
D_{(j-1)k+1} & & & \\
 & D_{(j-1)k+2} & & \\
 & & \ddots & \\
 & & & D_{jk-1}
\end{pmatrix},$$

o_{k-1} is a null block vector of length $k-1$ and $I^{(j)}$ is the identity matrix of size equal to that of $A^{(j)}$, for $j = 1, \ldots, p$.

On a distributed memory parallel computer, the solution of the reduced system may be quite expensive if the number of processors is large. To minimize data transmissions we suggest the following algorithm for each processor j (the j^{th} block of the solution is stored in d_j and each processor performs the i^{th} **send** operation if $j+2^i \leq p$ and the i^{th} `receive` operation if $j-2^i \geq 1$):

```
for i = 0, log2 p   1
    send d_j and T_j to processor j + 2^i
    receive d_{j-2^i} and T_{j-2^i} from processor j - 2^i
    d_j = d_j - T_j d_{j-2^i}
    T_j = -T_j T_{j-2^i}
end
```

Figure 1 shows the communications required by this algorithm on $p = 10$ processors. Observe that if $j \leq 2^i$, then T_j is a null matrix, and hence only vector $\boldsymbol{d}_j$ is transmitted. If $p > 1$, the solution of the reduced system requires at most $\lceil \log_2 p \rceil - 1$ matrix transmissions and $\lceil \log_2 p \rceil$ vector transmissions on each processor (see also Section 6).

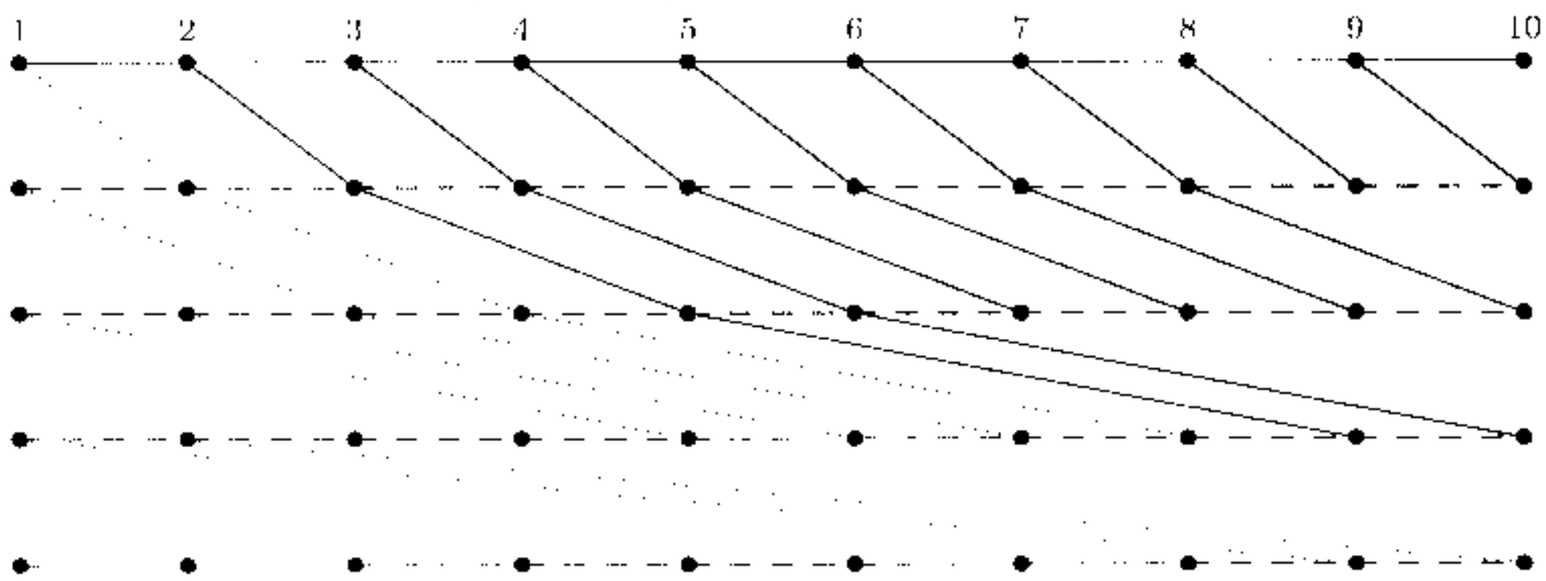

Figure 1 Communications required for the solution of the reduced system resulting from the reduction algorithm on 10 processors. The solid line expresses transmissions of matrices in the factorization step and of vectors in the solution of the linear system. The dotted line represents transmission of vectors in the solution step.

4. PARALLEL FACTORIZATION ALGORITHM

The third algorithm is based on the domain decomposition methods similar to those used in [10,11] to derive parallel factorizations of tridiagonal matrices and in [8] to solve ABD systems. The matrix A is factored as

$$A = N \cdot Q, \tag{5}$$

where

$$N = \begin{pmatrix} A^{(1)} & & & & & & \\ C_k e_k^{\top} & D_k & & & & & \\ & o_{k-1} & A^{(2)} & & & & \\ & & C_{2k} e_k^{\top} & D_{2k} & & & \\ & & \ddots & \ddots & & & \\ & & & o_{k-1} & A^{(p-1)} & & \\ & & & & C_{(p-1)k} e_{k-1}^{\top} & D_{(p-1)k} & \\ & & & & & o_{k-1} & A^{(p)} \end{pmatrix},$$

and

$$Q = \begin{pmatrix} I^{(1)} & & & & & & \\ o_k^{\top} & I_k & & & & & \\ & v^{(2)} & I^{(2)} & & & & \\ & T_2 & o_k^{\top} & I_{2k} & & & \\ & & \ddots & \ddots & & & \\ & & & v^{(p-1)} & I^{(p-1)} & & \\ & & & T_{p-1} & o_{k-1}^{\top} & I_{(p-1)k} & \\ & & & & & v^{(p)} & I^{(p)} \end{pmatrix}.$$

The unknown elements of N and Q are obtained from (5) by direct identification:

$$\begin{aligned} \boldsymbol{v}^{(j)} &= (A^{(j)})^{-1}\boldsymbol{e}_1 C_{(j-1)k+1}, && \text{for } j = 2,\ldots,p, \\ T_j &= -D_{jk}^{-1}C_{jk}\boldsymbol{e}_{k-1}^{T}\boldsymbol{v}^{(j)}, && \text{for } j = 2,\ldots,p-1. \end{aligned} \tag{6}$$

The factorization (5), the solution of N, and the updating of the right-hand side by $\boldsymbol{v}^{(i)}$ are performed in parallel with no data transmission between processors. Data transmissions are only required when solving the reduced linear system

$$\begin{pmatrix} I_k & & & & \\ T_2 & I_{2k} & & & \\ & T_3 & I_{3k} & & \\ & & \ddots & \ddots & \\ & & & T_{p-1} & I_{(p-1)k} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_k \\ \boldsymbol{x}_{2k} \\ \boldsymbol{x}_{3k} \\ \vdots \\ \boldsymbol{x}_{(p-1)k} \end{pmatrix} = \begin{pmatrix} \boldsymbol{g}_k \\ \boldsymbol{g}_{2k} \\ \boldsymbol{g}_{3k} \\ \vdots \\ \boldsymbol{g}_{(p-1)k} \end{pmatrix}. \tag{7}$$

The following is the algorithm for the generic processor j:

```
for i = (j-1)k+1, jk
    determine P_i, L_i and U_i such that P_i L_i U_i = D_i
end
V_{(j-1)k+1} = D_{(j-1)k+1}^{-1} C_{(j-1)k+1}
for i = (j-1)k+2, jk-1
    V_i = D_i^{-1} C_i V_{i-1}
end
T_j = -D_{jk}^{-1} C_{jk} V_{jk-1}
g_{(j-1)k+1} = D_{(j-1)k+1}^{-1} b_{(j-1)k+1}
for i = (j-1)k+2, jk
    b_i = b_i - C_i g_{i-1}
    g_i = D_i^{-1} b_i
end
obtain x_{(j-1)k} and x_{jk} by solving the reduced system
for i = (j-1)k+1, jk-1
    x_i = g_i - V_i x_{(j-1)k}
end
```

The algorithm for the solution of the reduced system created by the parallel factorization is quite similar to that seen in the previous section for the solution of the system created by the reduction algorithm. The main difference is that the parallel factorization algorithm gives a reduced system of smaller size (the last processor does not work to solve it). Moreover, since each processor j also needs to know vector $\boldsymbol{x}_{(j-1)k}$ at the end of the solution of the reduced system one additional vector transmission is required. The algorithm for the processor j is thus:

```
for i = 0, ⌈log2(p-1)⌉-1
  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
send x_jk to processor j+1
receive x_{(j-1)k} from processor j-1
```

Figure 2 shows the data communications required for the solution of the reduced system on $p = 10$ processors.

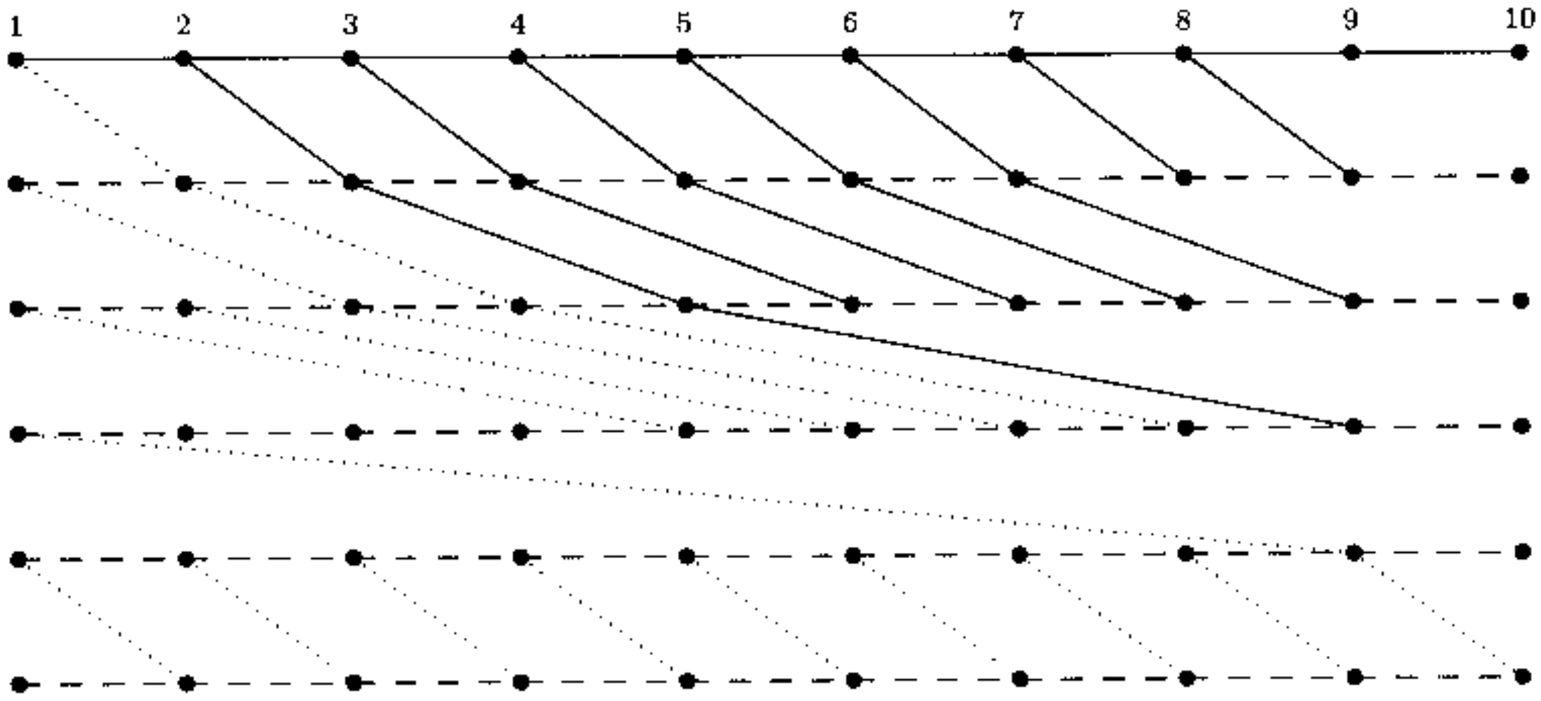

Figure 2. Communications required for the solution of the reduced system resulting from the Parallel Factorization on 10 processors. The solid line expresses transmissions of matrices in the factorization step and of vectors in the solution of the linear system. The dotted line represents transmission of vectors in the solution step.

5. GENERALIZED CYCLIC REDUCTION ALGORITHM

The last algorithm derives from cyclic reduction, which is one of the most important algorithms for the solution of tridiagonal and block tridiagonal linear systems on parallel computers [12,13]. Since we want to apply our approach to the coefficient matrix A in (2), we modify the generalized cyclic reduction algorithm presented in [10,11], where it is no longer necessary to assume that matrix A has a dimension which is a power of 2. The algorithm proceeds with $\lceil \log_2 k \rceil$ steps. Thus, the determined reduced system is exactly the same as the one resulting from the parallel factorization algorithm.

As in the reduction algorithm, first we scale the main diagonal blocks D_i of the matrix A. Then at the second step, by means of the block permutation matrix

$$P = \begin{pmatrix} P^{(1)} & & & & & \\ & I_k & & & & \\ & & P^{(2)} & & & \\ & & & \ddots & & \\ & & & & I_{(p-1)k} & \\ & & & & & P^{(p)} \end{pmatrix},$$

the coefficient matrix $D^{-1}A$ is factored in the following form:

$$D^{-1}A = PLMUP^{\top},$$

where M contains a block bidiagonal submatrix. To emphasize the structure of M, L and U, it is sufficient to consider another block permutation matrix Q, which first places the block elements on rows $1, 3, 5, \ldots, k+1, k+3, \ldots, 2k+1, 2k+3, \ldots$, and then those on rows $2, 4, \ldots, k$, $k+2, k+4, \ldots, 2k, 2k+2, 2k+4, \ldots$ (again, observe that it is not necessary that k be even). Then

$$QPMP^{\top}Q^{\top} = \begin{pmatrix} I & \\ & A_1 \end{pmatrix},$$

where I is a block identity matrix and

$$A_1 = \begin{pmatrix} I_2 & & & & & \\ \hat{C}_4 & I_4 & & & & \\ & \ddots & \ddots & & & \\ & & \hat{C}_k & I_k & & \\ & & & \hat{C}_{k+2} & I_{k+2} & \\ & & & & \ddots & \ddots \end{pmatrix}.$$

In a similar way,

$$QPLP^{\mathsf{T}}Q^{\mathsf{T}} = \begin{pmatrix} I & \\ L_1 & I \end{pmatrix}, \quad \text{and} \quad QPUP^{\mathsf{T}}Q^{\mathsf{T}} = \begin{pmatrix} I & U_1 \\ & I \end{pmatrix},$$

where the block identity matrices on the first and the second row of M, L and U have, respectively, $p\lfloor k/2 \rfloor$ and $p\lceil k/2 \rceil$ elements.

Observe that if k is even, then this step corresponds to the first step of the classical cyclic reduction applied to the whole matrix. The reduction process is repeated by each processor on successive block matrices with block bidiagonal submatrices (see Figure 3) until (after $\lceil \log_2 k \rceil$ steps) a reduced system of size $p-1$ is created.

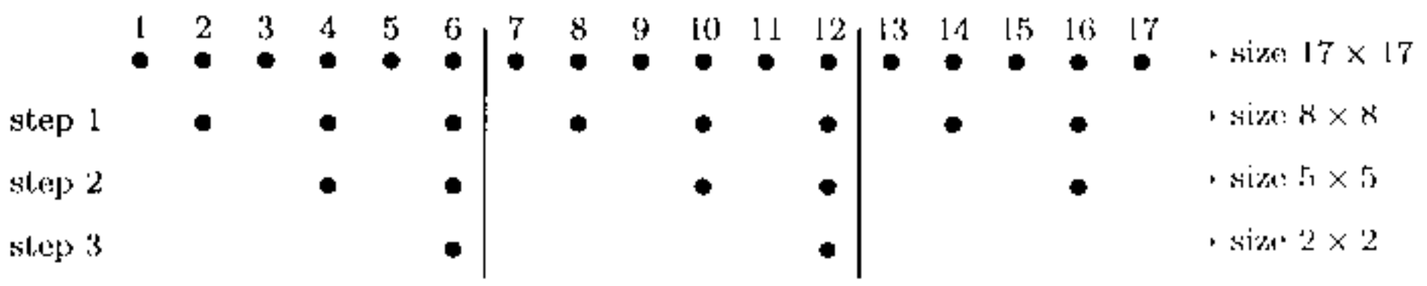

Figure 3. Reduction of a 17×17 matrix on 3 processors. Black points represent block rows involved in the reduced system.

The algorithm to obtain the solution of the problem (1) by means of the cyclic reduction algorithm, for a processor j, is thus:

```
for i = (j-1)k+1, jk
    determine P_i, L_i and U_i such that P_i L_i U_i = D_i
    E_i = D_i^{-1} C_i
    g_i = D_i^{-1} b_i
end
s = 1
while s < k
    for i = (j-1)k+2s, jk, step 2s
        b_i = b_i - E_i b_{i-s}
        E_i = -E_i E_{i-s}
    end
    if i - s < jk
        b_jk = b_jk - E_jk b_{(j-1)k+s}
        E_jk = -E_jk E_{(j-1)k+s}
    end
    s = 2s
end
obtain x_{(j-1)k} and x_jk by solving the reduced system
while s ≥ 1
    for i = (j-1)k+s, jk-1, step 2s
```

$$x_i = g_i - E_i x_{i-s}$$

end

$s = s/2$

end

The solution of the reduced system (7) should be obtained by using the same approach on $p/2$ processors, $p/4$, and so on. Because we are interested in solving large problems (1), the cost of any algorithm for the solution of (7) is negligible. Hence, we will use the same algorithm applied in the previous section with the parallel factorization algorithm.

6. ARITHMETICAL COMPLEXITY AND MEMORY REQUIREMENTS

In this section we introduce and compare the presented solvers from the point of view of computational cost and memory requirement. For simplicity, we assume that blocks in (2) have the same dimension. Hence, the parameters that need to be taken into account when deriving the computational cost and memory requirements of the proposed algorithms are:

m size of each block;

p number of processors involved;

n number of block rows of the problem;

q_i in the quasi-sequential algorithm the number of rows which are scaled in the i^{th} processor while previous processors complete their factorization step.

We will assume that the coefficient matrix may be divided in an optimal way among the processors ($n = kp$ 1), and that q_j $\min((j - 1)q, k)$, for $j = 1, \ldots, p$.

6.1. Arithmetical Complexity

The basic operations that are performed by the algorithms may be identified as (we use BLAS based notation):

$GETRF$ LU factorization with partial pivoting of a block of size m, $((2/3)m^3 - (1/2)m^2 \cdot (1/6)m$ operations);

$TRSM$ solution of an already decomposed linear system of size m, ($2m^2$ m operations);

$GEMM$ matrix-matrix (block-block) multiplication, ($2m^3 \cdot m^2$ operations);

$GEMV$ rank 2 update block-vector multiplication followed by a vector addition, ($2m^2$ operations);

$TRAN(s)$ transmission of s data elements.

Based on the above notation, the sequential solver performs the following operations:

$$[n]\ GETRF + [n]\ TRSM + [n-1]\ GEMV$$

and its computational cost is

$$C_s = \left(\frac{2}{3}m^3 + \frac{7}{2}m^2 - \frac{7}{6}m\right) n \cdot 2m^2.$$

For all four presented algorithms, the only part which is completely parallelizable is the factorization of the blocks on the main diagonal. We must then follow two different paths to obtain the computational cost of the quasi-sequential algorithm and of the remaining three parallel algorithms.

In the quasi-sequential algorithm, the update of the right-hand side blocks remains sequential. The computational cost of this part is thus the sum of the computational costs on all the processors. The solution of the block linear systems is partially parallelized. If we suppose that each

processor j performs the scaling of the main diagonal blocks of the first q_j rows when the previous processors are solving their part of the system, then the operations for this phase overlap and are not counted. This means that the operations performed are

$$[k]\ GETRF + \left[\sum_{j=0}^{p-1}(k \quad jq)_+\right]\ TRSM + [n]\ GEMV + [p-1]\ TRAN(m),$$

where $(x)_+ = x$ if $x \geq 0$ and $(x)_+ = 0$ otherwise, and $q \leq \lfloor (4m-1)/(2m^2 + m - 1)k \rfloor$.

The total cost is thus

$$C_{qs} \quad \left(\frac{2}{3}m^3 \quad \frac{1}{2}m^2 + \frac{1}{6}m\right)\frac{(n+1)}{p} + \sum_{j=0}^{p-1}\left(2m - 1 - \frac{4m \quad 1}{m+1}j\right)_+ \frac{m(n+1)}{p} + 2m^2n \qquad (8)$$

arithmetical operations and

$$T_{qs} \quad (p \quad 1)t(m)$$

transmissions, where $t(m)$ is the cost of transmission of a vector of length m.

If $p > m/2$, the summation does not depend on p, but essentially on $m^2/2$. Then formula (8) simplifies to

$$C_{qs} = \left(\frac{7}{6}m^3 - \frac{1}{2}m^2 + \frac{1}{6}m\right)\frac{(n+1)}{p} + 2m^2n.$$

The number of arithmetical operations consists of two factors: the first depends on m and n but does not depend on p; the second one depends on $1/p$ and decreases when the number of processors increases.

For the reduction algorithm, the parallel factorization and the cyclic reduction algorithms, the operations leading to the creation of the reduced system as well as the appropriate data transmissions are completely parallelizable. The total number of operations performed by each processor will thus essentially depend on two terms that are proportional to $1/p$ and to $\lceil \log_2 p \rceil$.

The reduction algorithm requires the following operations in its initial phase:

$$\lfloor k \rfloor\ GETRF + \lfloor k(n+1) \rfloor\ TRSM + \lfloor k \quad 1 \rfloor\ GEMM + \lfloor 2k - 2 \rfloor\ GEMV + \lfloor 1 \rfloor\ TRAN\left(m^2 + m\right)$$

followed by the solution of the block reduced system of size p that requires

$$\lfloor \lceil \log_2 p \rceil \quad 1 \rfloor\ TRAN\left(m^2 + m\right) + \lfloor 1 \rfloor\ TRAN(m) + \lfloor \lceil \log_2 p \rceil - 1 \rfloor\ GEMM + \lfloor \log_2 p \rfloor\ GEMV.$$

The total number of arithmetical operations is

$$C_{rd} = \left(\frac{14}{3}m^3 + \frac{7}{2}m^2 + \frac{7}{6}m\right)\frac{(n+1)}{p} + \left(2m^3 + m^2\right)\lceil \log_2 p \rceil \quad 4m^3,$$

and the data transmission cost is

$$T_{rd} = \lceil \log_2 p \rceil t\left(m^2 + m\right) + t(n).$$

The parallel factorization and the cyclic reduction algorithms require in their initial parallel phase the following operations:

$$[k]\ GETRF + [k(n+1)]\ TRSM + [k-1]\ GEMM + [2k-2]\ GEMV$$

followed by the solution of the reduced system of size $p - 1$ which is characterized by similar arithmetical complexity function as above. The total cost of these algorithms is almost the same as that of the reduction algorithm. The arithmetical complexity is

$$C_{pf} = C_{cr} = \left(\frac{14}{3}m^3 + \frac{7}{2}m^2 + \frac{7}{6}m\right)\frac{(n+1)}{p} + \left(2m^3 + m^2\right)\lceil \log_2(p-1) \rceil - 4m^3,$$

while the cost of transmissions is

$$T_{pf} = T_{cr} = (\lceil \log_2(p-1) \rceil - 1)\, t\left(m^2 + m\right) + 2t(m).$$

Assuming (as in most common cases of computational practice) that n is much larger than m, that m is large enough so that the m^3 terms dominate lower order terms, and that the costs of data transmission are negligible, we can simplify the above formulas. In Table 1, we present the simplified formulas (the reduction, parallel factorization and cyclic reduction algorithms have the same cost and are labeled "parallel algorithms").

Table 1. Computational costs for the proposed algorithms.

Algorithm	Computational Cost
sequential	$\frac{2}{3}m^3 n$
quasi-sequential	$\frac{7}{6}m^3 n/p + 2m^2 n$
parallel algorithms	$\frac{14}{3}m^3 n/p + 2m^3 \lceil \log_2(p) \rceil$

A number of observations can be made when considering the arithmetic complexity functions of Table 1. When comparing the sequential and the quasi-sequential algorithms, for a large number of processors, the term $(7/6)m^3 n/p$ disappears. Therefore, the limit on the speedup of the quasi-sequential algorithm is $n/3$ (see also Figures 4 and 5).

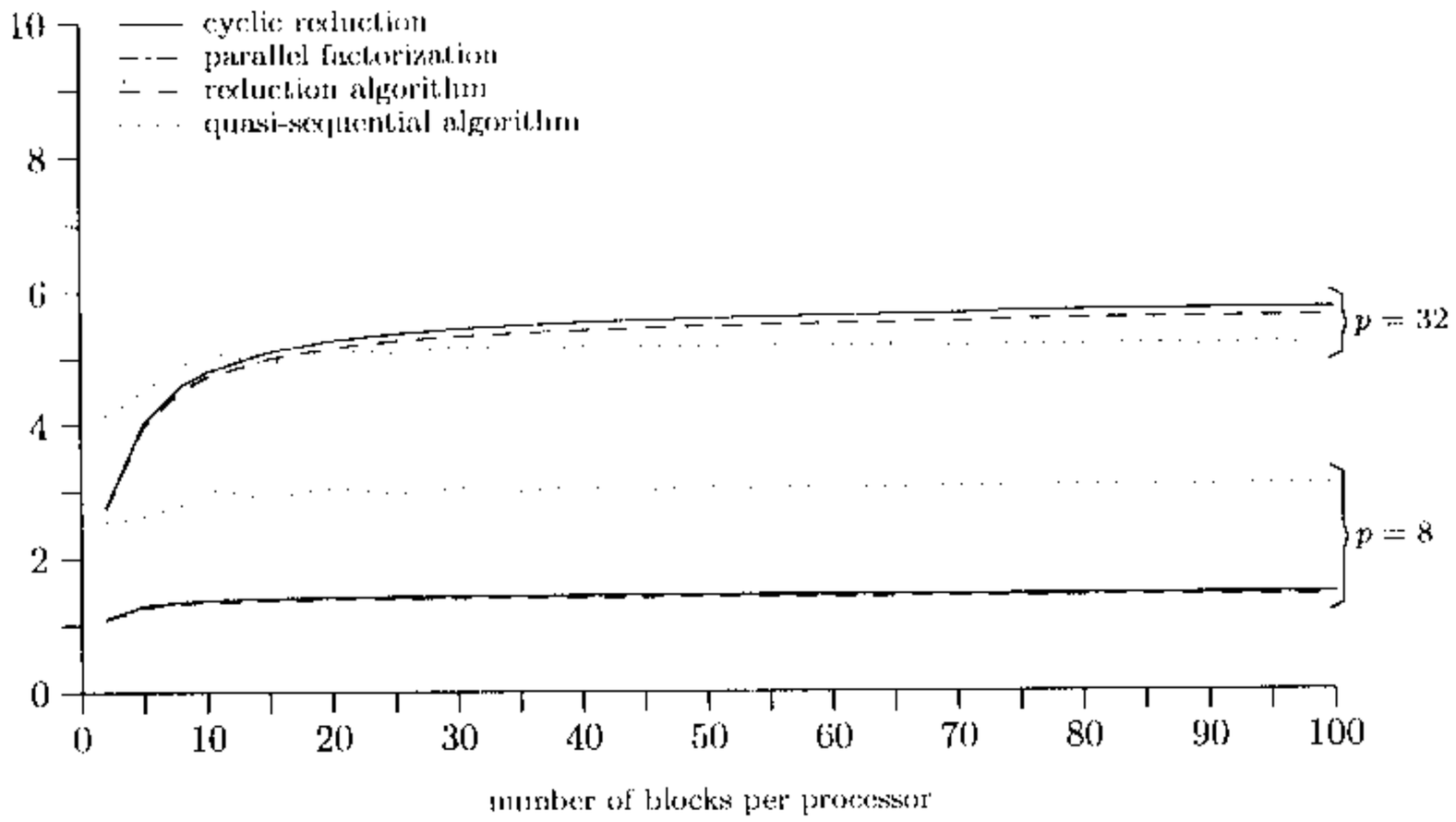

Figure 4. Speedups of the four algorithms for $m = 15$ and $p = 8, 32$.

The arithmetical complexity functions for the parallel algorithms have a structure similar t other *divide-and-conquer* algorithms [8,14] and have similar characteristics. For fixed m and and for large n, further increase in the value of n does not lead to additional speedup. Similarl (excluding the communication cost), for fixed n and p, an increase in m will not produce furthe speedup increase.

Finally, given the values of n, m and p, we can estimate (see Figure 6) when the quasi sequential algorithm should be used instead of either of the parallel algorithms. It can be observe that for a given number of processors p, there exists a value of m starting from which th quasi-sequential algorithm outperforms the parallel algorithms. It can also be observed that i the data transmission costs are taken into consideration (and the cost of one transmission of

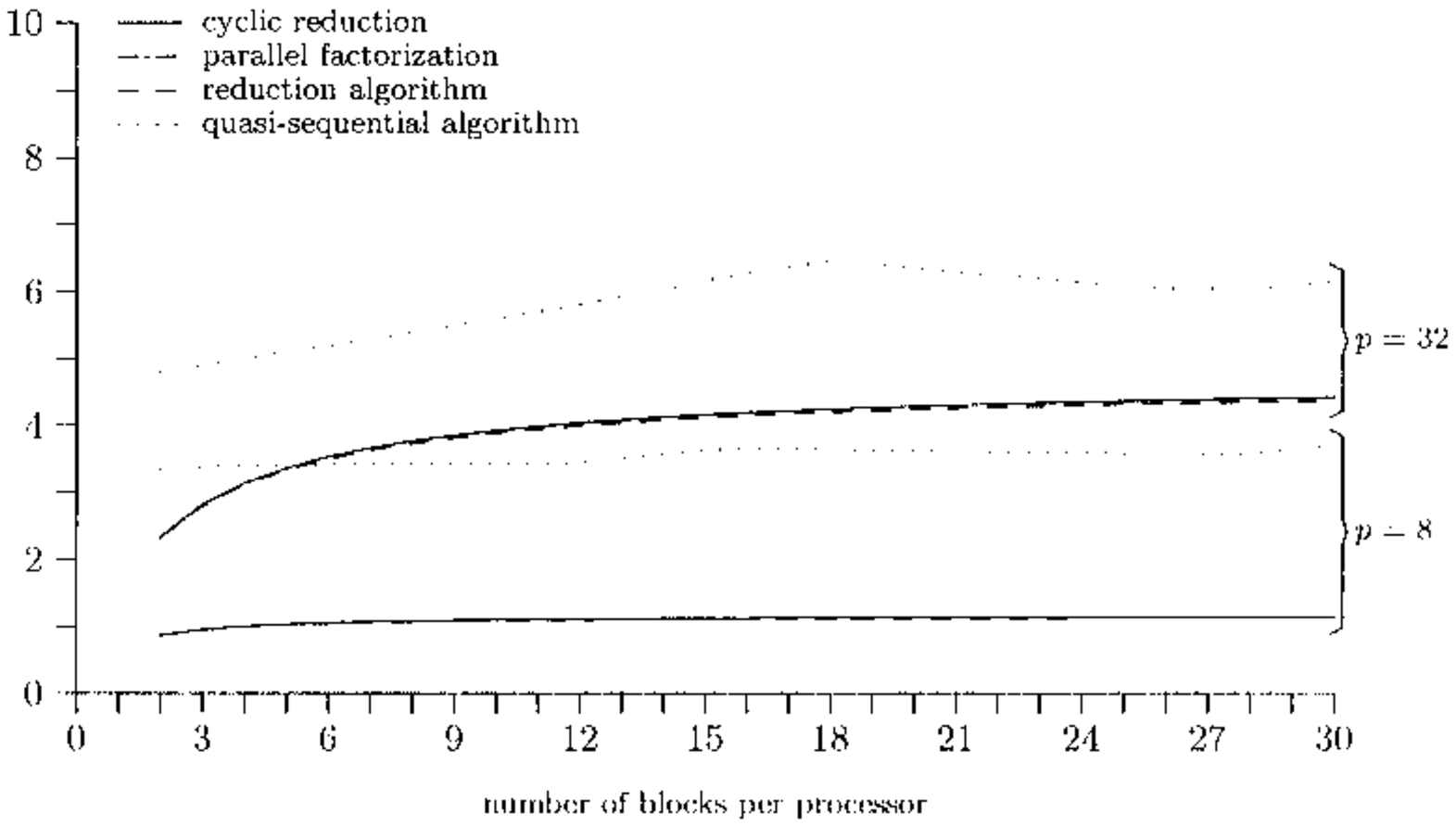

Figure 5. Speedups of the four algorithms for $m = 30$ and $p = 8, 32$.

vector of s elements is almost the same as that of s transmissions of scalars), then the quasi-sequential algorithm requires $O(pn)$ data transmissions whereas the parallel algorithms require $O(\log_2(p)n^2)$ data transmissions. Therefore, for moderate p and large n, the n^2 term will dominate the data transmission cost, giving the quasi-sequential algorithm additional advantage over the parallel algorithms. As the number of processors increases, the situation reverses. Overall the cost of data transmission of the parallel algorithms is smaller than that of the quasi-sequential algorithm for $n < p/\log_2(p)$. These results are also confirmed by our experiments (see Figures 4, 5, and 7–9).

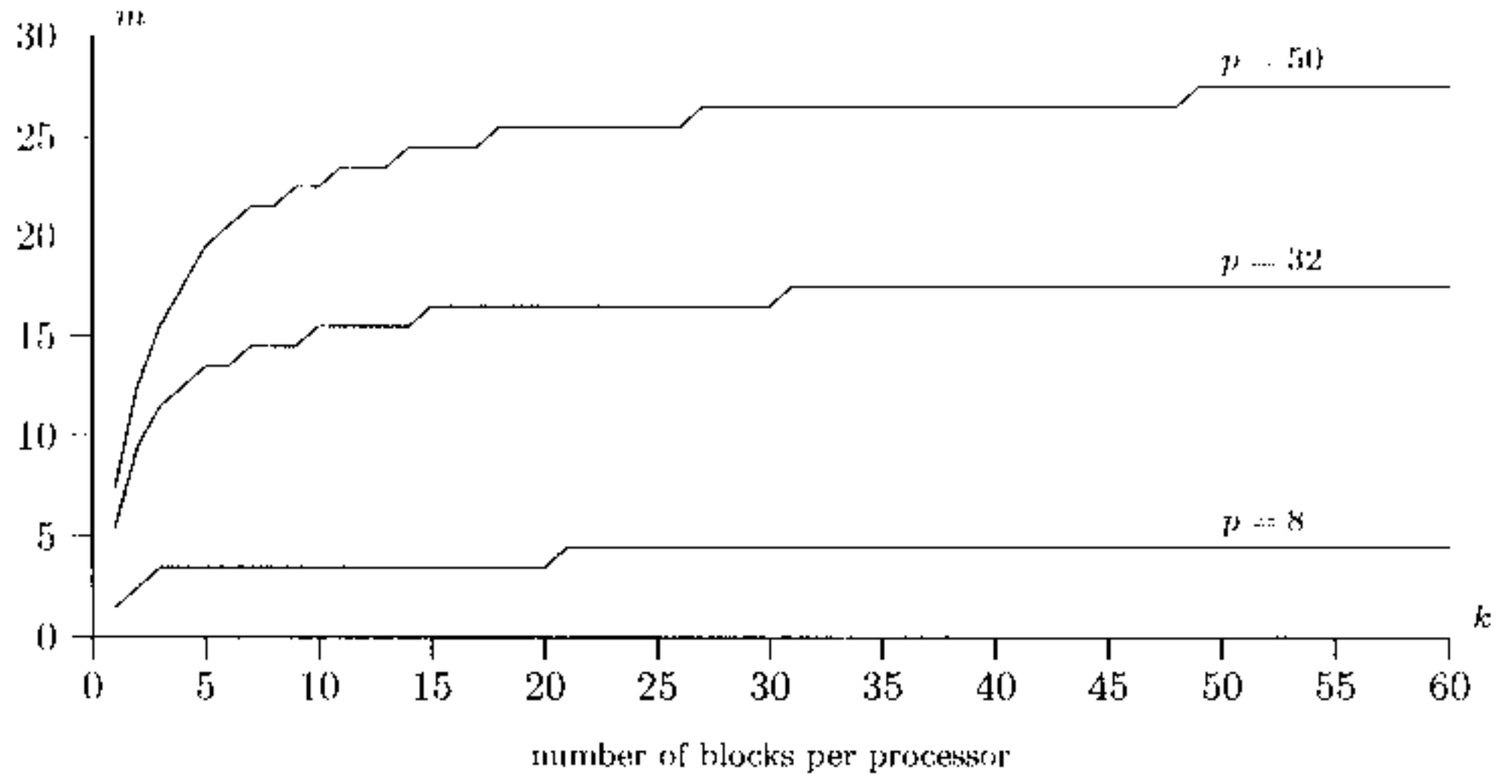

Figure 6. Theoretical comparison between the parallel factorization and the quasi-sequential algorithm. For each line (representing a given number of processors p) points above it are values of k and n for which the quasi-sequential algorithm outperforms the parallel factorization.

6.2. Memory Requirements

We will consider two cases of memory requirements of the proposed algorithms depending on when the solution of the linear system is performed. If the factorization step and the linear system solution step are performed separately, the parallel factorization algorithm requires to store one fill-in vector, and the cyclic reduction algorithm requires to store the off-diagonal elements of the

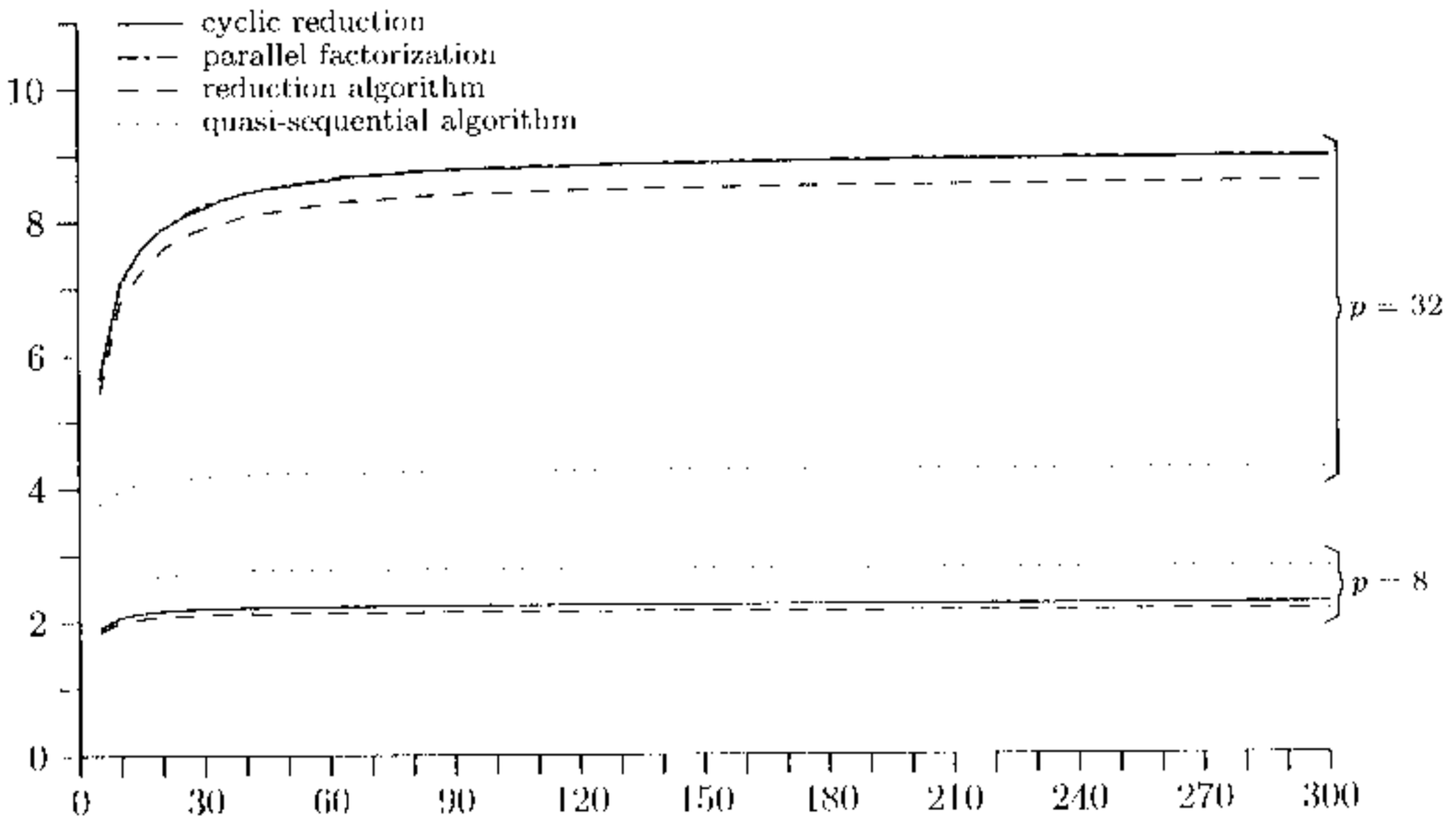

Figure 7. Speedups of the four algorithms for $m = 5$ and $p = 8, 32$.

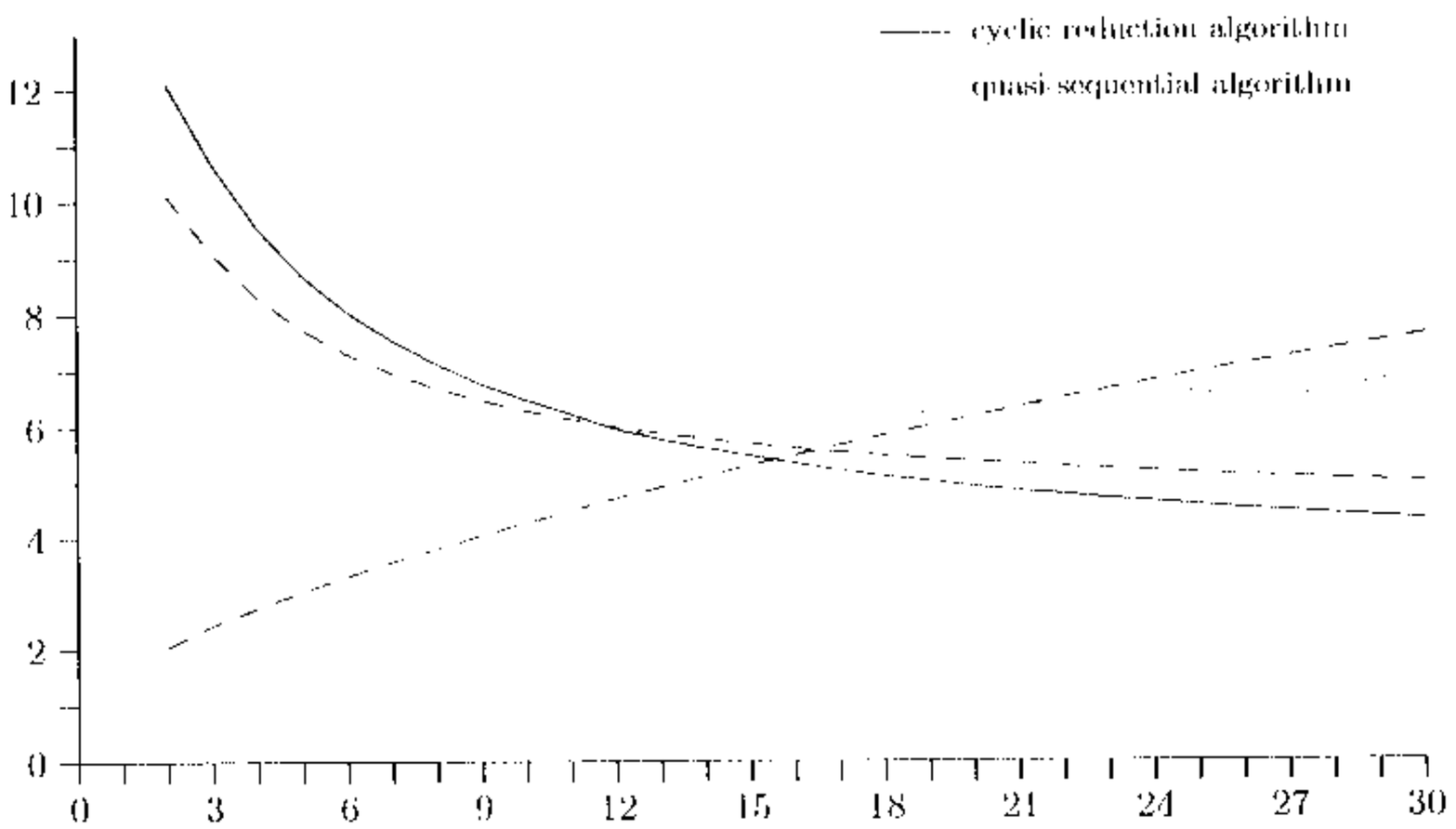

Figure 8. Speedups of the algorithms for $p = 32$ and $k = \lfloor 800/m \rfloor$ variable. The dashed lines represent the theoretical speedups of the two algorithms.

Table 2. Per-processor memory requirements for the proposed algorithms when the coefficient matrix is factored before the solution of the associated linear system.

Algorithm	Workspace
sequential	$2nm^2 + nm$
quasi-sequential	$2kn^2 + (k+1)m$
reduction	$(2k + \lceil \log_2 p \rceil)m^2 + (k+1)m$
parallel factorization	$(3k + \lfloor \log_2(p-1) \rceil - 1)m^2 + (k+1)m$
cyclic reduction	$(3k + \lceil \log_2(p-1) \rceil - 1)m^2 + (k+1)m$

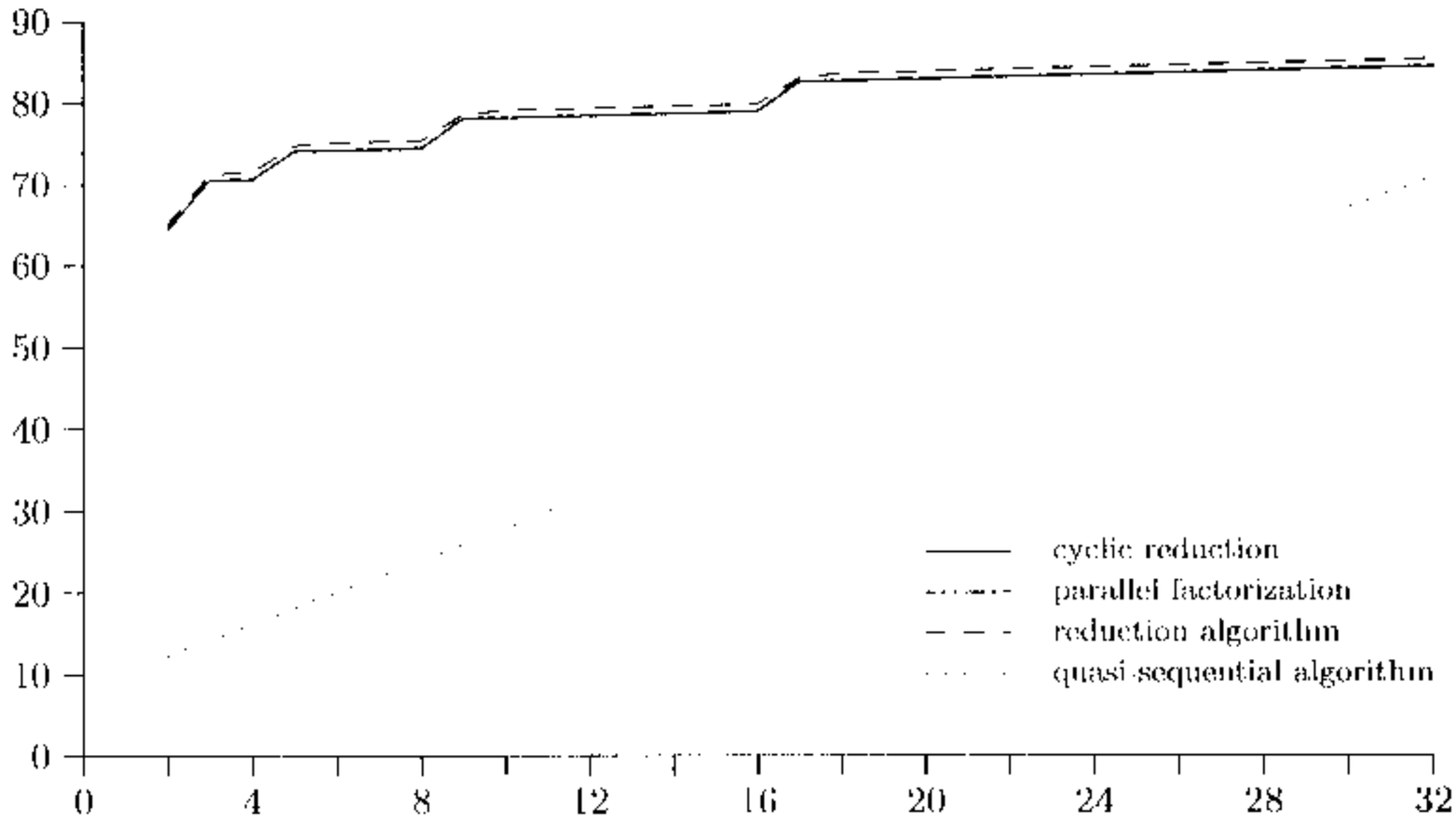

Figure 9. Time (in seconds) of execution for $k = 10$ and $m = 30$ on varying numbers of processors.

submatrices obtained at each step of the reduction. Table 2 presents the per-processor memory requirements of the proposed algorithms. It can be observed that the sequential and quasi-sequential algorithms have precisely the same overall memory requirements. The per-processor memory requirements of the reduction algorithm are slightly higher than those of the quasi-sequential algorithm for a moderate p, and substantially lower than those of the remaining two parallel algorithms for large k. The overall memory requirement of the parallel factorization and cyclic reduction algorithms is larger by km^2 elements than that of the reduction algorithm and by $(k + \lfloor \log_2(p - 1) \rfloor - 1)m^2$ elements than that of the quasi-sequential algorithm.

If the factorization is performed together with the linear system solution, the per-processor memory requirements of the proposed algorithms are summarized in Table 3.

Table 3. Per-processor memory requirements for the proposed algorithms when the solution of the associated linear system and the factorization of the coefficient matrix are performed at the same time.

Algorithm	Workspace
sequential	$2mm^2 + mm$
quasi-sequential	$2km^2 + (k + 1)m$
reduction	$(2k + 1)m^2 + (k + 1)m$
parallel factorization	$2km^2 + (k + 1)m$
cyclic reduction	$2km^2 + (k + 1)m$

It can be observed that in all cases the per-processor memory requirements of all parallel algorithms are the same and that the overall memory requirement of all the algorithms are almost the same.

7. NUMERICAL TESTS

The algorithms presented in previous sections have been coded in Parallel Fortran [15] with the Express communication library [16] on a network of 32 Microway transputers T800–20, each 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 to factorize the main diagonal blocks with only local row pivoting, and DGESL to solve triangular linear systems factored by DGEFA.

As a test problem we have chosen a linear system with $m_i = m$ for all i, and

$$D_i = I - h_i E, \qquad C_i = -I - h_i E$$

arising from the numerical solution of the initial value problem

$$\begin{aligned} \boldsymbol{y}' &= E\boldsymbol{y} + \boldsymbol{b}(t), \qquad t \subset [t_0, T], \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0 \end{aligned} \tag{9}$$

by means of the trapezoidal rule. The obtained block bidiagonal system is in general used as preconditioning when problem (9) is solved by means of Boundary Value Methods (see [4,5]). We have selected the $m \times m$ matrix E with eigenvalues in the negative part of the complex half-plane in order to have a stable solution (for the purpose of our experiments, E was obtained using a random number generator). Moreover, h_i γh_i $_1$, for $\gamma > 1$ fixed.

In Figures 4, 5, and 7, the measured speedups of the cyclic reduction, parallel factorization, reduction and quasi-sequential algorithms are reported for m 5, m 15, m 30, different values of k $(n + 1)/p$, and for different numbers of processors. 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 processor number, $j = 1, \ldots, p$, in such a way that the speedup was the highest (the optimal value of q_j will be architecture dependent). For example, q $k/4.5$ if m 5, $q = k/10$ if m 15 and q 1 for m 30. It should be noted that in this last case the values of k are relatively small (less than 30).

The results of all the experiments accord with the theoretical results of the previous section. Figures 4, 5, and 7 show that if a small number of processors is used, the quasi-sequential algorithm is always the fastest among the presented algorithms. For increasing numbers of processors, parallel algorithms become preferable (but only when the size of each block (m) is not too large). For m 15 and $p = 32$, the quasi-sequential algorithm behaves almost exactly like the parallel algorithms. The increase in m not only reduces the utility of the parallel algorithms but also reduces the obtained speedup; for p 32, the reduction is from the speedup of about 9 for $m = 5$ to speedup of less than 7 for m 30. It should be also observed that the cyclic reduction and parallel factorization algorithms behave exactly the same and always outperform the reduction algorithm.

Figure 8 compares the performance of cyclic reduction and quasi-sequential algorithms for p 32 processors, for a matrix of size $n = 25600$ and for approximately fixed k $[800/m]$ while varying the value of m. It also compares the practical and the theoretical speedup for both algorithms.

Figure 8 shows that for suitable large dimensions of the coefficient matrix A, theoretical speedups are very good approximations to those obtained on the parallel computer. Moreover, the cyclic reduction algorithm outperforms the quasi-sequential algorithm only if $m \leq 16$. Finally, if the optimal algorithm is selected properly for given n, m and p, it is possible to obtain speedup greater than 5.5 on 32 processors.

The last Figure 9 represents the "scalability" of the presented algorithms, that is, the time of execution for a different number of processors with k and m constant. This graph shows that the parallel algorithms scale nicely as the number of processors increases (the jumps in the time are related to the fact that the workload in the solution of the reduced system depends on $\log_2 p$). As predicted, for fixed k and m, the quasi-sequential algorithm does not scale well. It can be observed again that cyclic reduction and factorization algorithms behave similarly, slightly outperforming the reduction algorithm. Moreover, it can be predicted that the crossover point will occur for approximately 40 processors.

8. CONCLUSIONS

We have presented four parallel algorithms for the solution of block bidiagonal linear systems, discussed their implementation details, derived and discussed their arithmetical complexity functions and the memory requirement functions. It was shown that there is no simple answer which of the presented algorithms should be applied. The answer to this question depends on the parameters of the linear system itself (block size and number of blocks) as well as characteristics of the distributed memory computer that the systems are to be solved on (available memory per-processor and the communication overhead). Some guidelines for the selection process have been presented.

REFERENCES

1. I. Gladwell and M. Paprzycki, Parallel solution of almost block diagonal systems on the CRAY Y-MP using level 3 BLAS, *J. of Comput. and Appl. Mathem.* **45**, 181–189 (1993).
2. K. Wright, Parallel treatment of block bidiagonal matrices in the solution of ordinary differential boundary value problems, *J. of Comput. and Appl. Mathem.* **45**, 191–200 (1993).
3. S.J. Wright, Stable parallel algorithms for two-point boundary value problems, *SIAM J. Sci. Stat. Comput.* **13**, 742–764 (1992).
4. P. Amodio and F. Mazzia, Parallel block preconditioning for the solution of boundary value methods, Report n. 12/93 of the Dipartimento di Matematica, Università di Bari, Bari, Italy.
5. P. Amodio and F. Mazzia, Parallel iterative solvers for boundary value methods, Report n. 10/94 of the Dipartimento di Matematica, Università di Bari, Bari, Italy.
6. U.M. Ascher and P.S. Chan, On parallel methods for boundary value ODEs, *Computing* **46**, 1–17 (1991).
7. G.H. Golub and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, (1989).
8. M. Paprzycki and I. Gladwell, Solving almost block diagonal systems on parallel computers, *Parallel Computing* **17**, 133–153 (1991).
9. L. Brugnano, A parallel solver for tridiagonal linear systems for distributed memory parallel computers, *Parallel Computing* **17**, 1017–1023 (1991).
10. P. Amodio and L. Brugnano, Parallel factorizations and parallel solvers for tridiagonal linear systems, *Lin. Alg. Appl.* **172**, 813–823 (1992).
11. P. Amodio, L. Brugnano and T. Politi, Parallel factorizations for tridiagonal matrices, *SIAM J. Numer. Anal.* **30**, 813–823 (1993).
12. D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.* **13**, 484–496 (1976).
13. J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, (1988).
14. M. Paprzycki and P. Stpiczyński, Solving linear recurrence systems on parallel computers, In *Proceedings of the Conference Mardi Gras '94*, Baton Rouge, February 10–12, 1994, Nova Science Publishers, New York (to appear).
15. *Parallel Fortran User Guide*, 3L Ltd, Livingstone, (1988).
16. *Express Fortran User's Guide*, ParaSoft Corp., Pasadena, (1990).