Parallel Computing 17 (1991) 133–153
North-Holland

Solving almost block diagonal systems on parallel computers

Marcin Paprzycki* and Ian Gladwell

Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

Received February 1990
Revised September 1990

Abstract

Paprzycki, M. and I. Gladwell, Solving almost block diagonal systems on parallel computers, Parallel Computing 17 (1991) 133–153.

Finite difference methods for nonlinear boundary value problems (BVP's) in ordinary differential equations involve solving systems of linear equations at an inner iteration. In the typical case of separated boundary conditions, these systems are almost block diagonal (ABD). A new 'tearing' algorithm for the parallel solution of those ABD's is presented here. It is an extension of one proposed by Dongarra and Johnsson [7] for positive definite or strictly diagonally dominant banded systems and it is similar to one proposed by Wright [15] for banded systems. We compare the cost of the new algorithm with costs of other algorithms which might be applied to ABD's. We simulate the use of the algorithm in solving BVP's. The new algorithm is not designed for a specific computer architecture and it is believed that the analysis is sufficiently general to be indicative of performance for most current parallel architectures.

Keywords. Linear algebra; almost block diagonal linear systems; new tearing algorithm; comparison with other algorithms; complexity analysis; simulation results.

0. Introduction

When solving nonlinear boundary value problems (BVP's) in ordinary differential equations (ODE's), often a 'global' method, such as finite differences or spline collocation, is the best choice. Most robust global codes solve the BVP on a sequence of meshes determined using internally computed error estimates. On each mesh the method is used to produce a system of nonlinear algebraic equations which are solved by a Newton method. At each Newton iteration an almost block diagonal (ABD) system (see *Fig. 1*) is solved, possibly for several right hand sides (RHS's) sequentially. All parts of the global algorithm are conceptually simple to parallelize except this ABD solution and, at a much lower level of cost, the error estimation; see [9,10] for a discussion.

We consider the solution of ABDs arising in the finite difference solution of a first order system of BVP's, of any size, with separated boundary conditions (BC's). As a basis for comparison we use a fast sequential ABD solver which we also use to solve the smaller ABD's arising from 'tearing' the original system in a natural way. In Section 1 we describe an algorithm. It extends to ABD systems one due to Dongarra and Johnsson [7] (D&J) for banded

* Current address: *Department of Mathematics and Computer Science, University of Texas of the Permian Basin, Odessa, TX 79762, USA.*

systems. In Section 2 we determine its arithmetic cost and an 'optimal' tearing strategy. We compare this cost with that of the sequential algorithm and a version of the D&J algorithm which assume no interchanges. The latter is an unrealistic approach to solving ABD's arising in BVP's since it always costs more then our fast sequential algorithm. However, it enables us to judge the effect of assuming the ABD is just banded without any interior structure. In Section 3, we consider the interprocessor communication demands of our algorithm and the memory requirements on a distributed memory system. Finally in Section 4, we provide a theoretical simulation where we cost the solution of a 'typical' set of ABD systems such as would arise when solving a difficult nonlinear BVP system. We report a simulation on a Sequent Symetry with 20 processors designed to verify our theoretical predictions.

1. Algorithm description

1.1. ABD solvers

We limit discussion to ABD's of the type arising when finite differences are used to solve BVP's for ODE's. For simplicity, we assume the system results from discretizing a linear first-order system of n ODE's with q separated BC's, $q < n$ at the left end, using the 'box scheme' due to Keller [12]. Such systems arise at each Newton iteration of the discretization of nonlinear BVP's with separated BC's. The results can be extended to ABD systems arising from the collocation method and/or treating higher order equations directly.

As far as we are aware there are no parallel solvers specifically intended for ABD's. The best known sequential codes are the row elimination code SOLVEBLOCK [4] which is used in COLSYS [1] and the alternating row and column elimination code ARCECO (COLROW) described in [14] and later coded in [5]. The advantage of ARCECO over SOLVEBLOCK is that ARCECO does not generate fill-in. We use an implementation available in the NAG library (subroutines F01LHF and F04LHF) [3,13]. It is coded using the level 2 BLAS, [6], and is suitable for use on vector processors. It uses row and column *interchanges* to avoid fill-in (like the unpublished ROWCOL [5]) rather than the row and column *eliminations* of ARCECO. We use a modified version of this code on each processor to solve a subproblem that is ABD.

It is possible to solve ABD's by treating them as banded or block tridiagonal systems. Both these possibilities have significant disadvantages. The former does not allow us to take full advantage of the structure. As will be shown later, use of parallel banded system solvers leads to significant increases in the expected workload in comparison with the ABD approach. The latter method is also problematic. Matrices which arise from finite differences not only require pivoting, but also require codes with good stability properties. Most popular ways of solving block tridiagonal systems cannot be used directly without compromising on stability. See [2] for a fuller discussion.

Our parallel algorithm is based on techniques developed for banded systems and presented in [7,8]. The general idea is to divide the system into smaller segments of similar structure which can be factorized independently and from these to create a reduced system again of the same structure which is factorized on a single processor. Finally the result is calculated in parallel. We will separate the decomposition phase from the solution phase as, in the context of solving BVP's, we will often need to solve for several RHS's sequentially.

1.2. Partition strategy

For banded systems, Dongarra and Johnsson [7] 'tear' the system into a number of approximately equal sized subproblems. Of course, each of these problems is automatically also

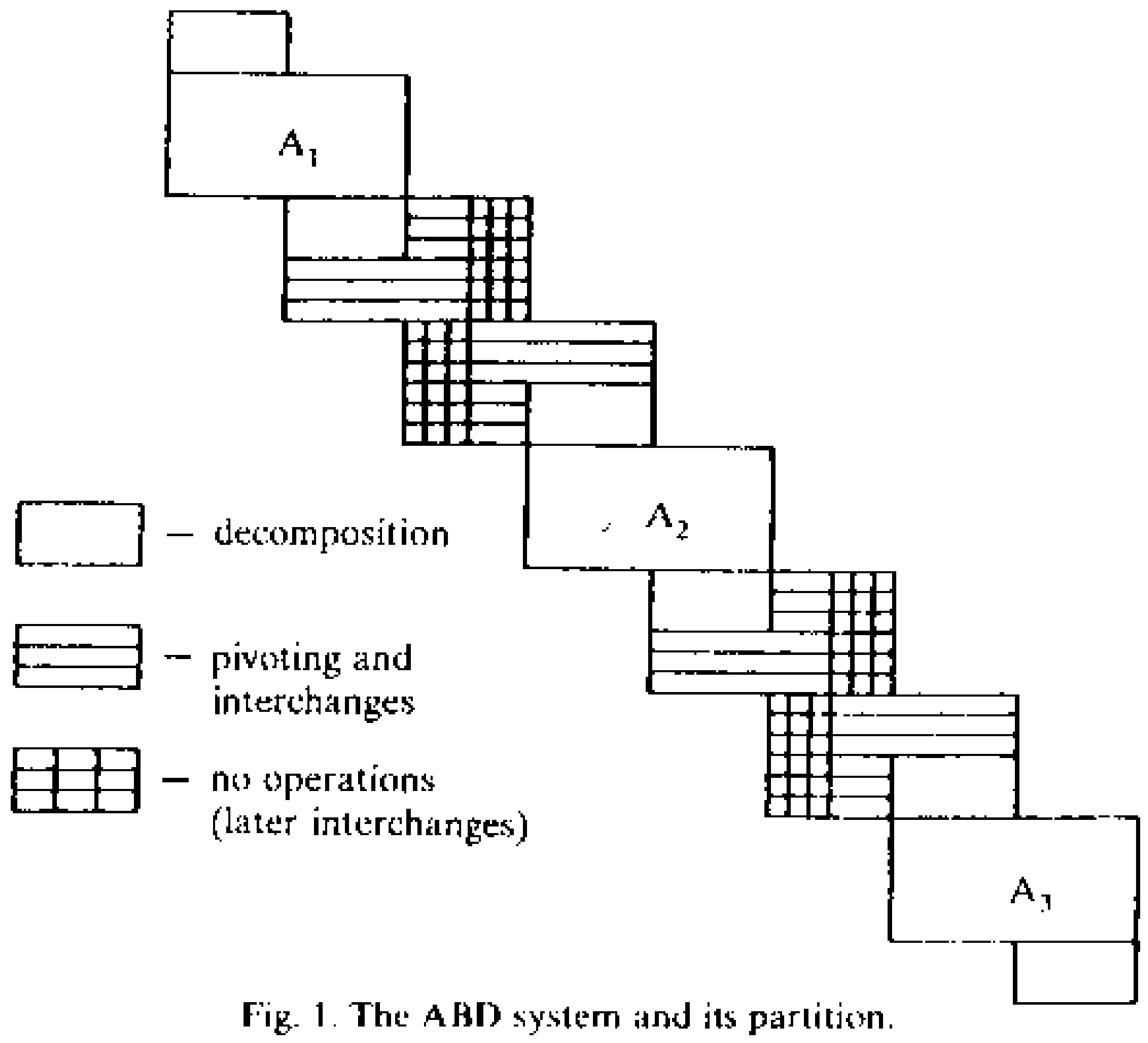
A1
A2
A3
— decomposition
— pivoting and interchanges
— no operations (later interchanges)

Fig. 1. The ABD system and its partition.

banded. The reduction process, similar to that described below, leads naturally to a reduced system which is also banded. The 'inequality' of the ABD structure implies that we must make a more carefull choice when 'tearing' the system.

Consider a typical ABD arising from discretizing a first order system of ODE's (see *Fig. 1*). The partition strategy we adopt is one where each segment has the same ABD structure as the original matrix. An example of such a division (into three segments) of an ABD with nine blocks (seven internal) is presented. This is the maximum number of segments possible for a nine-block ABD. In general the number of blocks in each segment will be greater. The ABD in *Fig. 1* could have arisen from a finite difference method for a BVP using six internal mesh points. Our way of partitioning the system allows us to use existing software. Each of the A_i matrices (segments) is factorized using the most efficient ABD strategy. When the routine F01LHF is used to perform this task, the matrices A_i are decomposed as follows:

$A_i = P_i L_i U_i Q_i$ where *P_i-contains row permutations*
L_i-is lower triangular
U_i-is upper triangular
Q_i-contains column permutations.

However for ABD's arising from finite differences one cannot guarantee 'nice' properties of the segments. It can be shown that when the BVP is nonsingular and the mesh is fine enough the overall ABD matrix from a finite difference discretization is nonsingular. This is not necessarily true for the segments chosen as above and there exist simple examples of nonsingular matrices A with structurally singular segments A_i. Row and column permutations, P and Q, suitable for successful decomposition of A are not available to the processors which decompose the segments. They might not provide a stable ordering in the segments even if they were available.

To avoid this problem a special way of treating first and last blocks in each internal segment has been used. For the first block the first $n-q$ row 'eliminations' are just row interchanges without decomposition. In a similar way for the last block the decomposition stops after $n-q$ row eliminations. However the whole last block provides elements for interchanges. We have observed that this extra freedom in the choice of pivots prevents the occurrence of structural singularities. For the first segment A_1 the special treatment is necessary only for the last block. The situation is different for the last segment A_P. When the last block contains right boundary conditions, it is of full rank. A_P should be copied 'upside down', which is equivalent to decomposing it starting from the last element up. This approach allows us to avoid structural singularity in the same way as for the first segment. Though we have no proof in the general case, in all of the examples with structurally singular segments which we have considered, these additional row interchanges prevent the occurrence of structural singularities. Obviously they cannot prevent numerical singularities.

1.3. The parallel algorithm

Phase I – Decomposition of the system

In Phase I the system is divided into P segments as shown in *Fig. 1*, where P is no larger than the number of available processors. Each segment is decomposed using a modified version of NAG routine F01LHF which we call ABDEXT. This leads to the system in *Fig. 2* (using notation similar to Dongarra and Sameh [8]) where

$$A_i \in R^{M\times M},$$

$$S_i \in R^{q\times n},\qquad \tilde{S}_i = \begin{bmatrix} 0 \\ S_i \end{bmatrix} \in R^{M\times n},\qquad T_i \in R^{(n-q)\times n},\qquad \tilde{T}_i = \begin{bmatrix} T_i \\ 0 \end{bmatrix} \in R^{M\times n},$$

$$M_i \in R^{q\times n},\qquad \tilde{M}_i = \begin{bmatrix} M_i \\ 0 \end{bmatrix} \in R^{n\times n},\qquad N_i \in R^{(n-q)\times n},\qquad \tilde{N}_i = \begin{bmatrix} 0 \\ N_i \end{bmatrix} \in R^{n\times n},$$

$$W_i \in R^{n\times n}.$$

M = size of the segment.

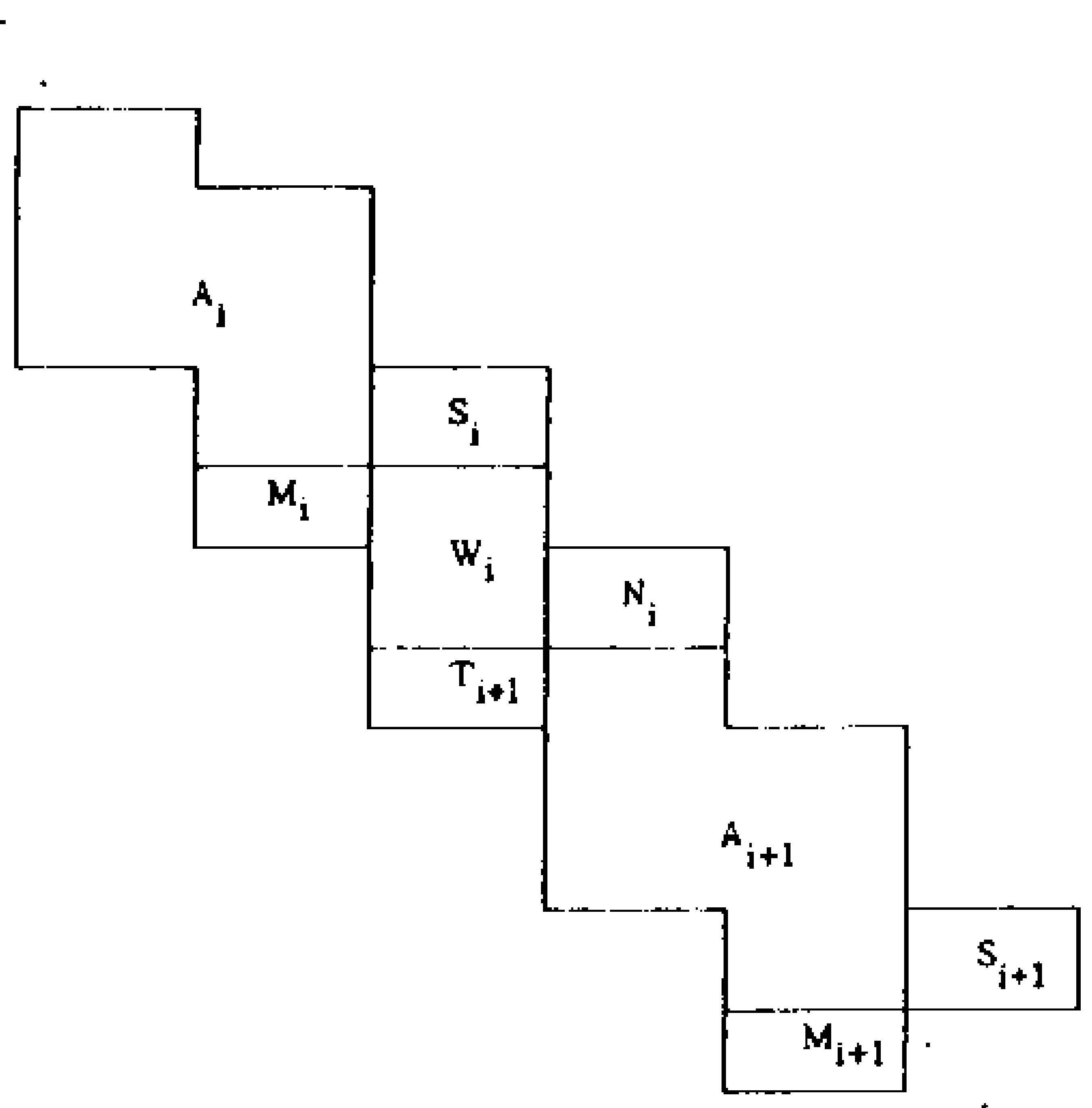

Fig. 2. System after Phase I.

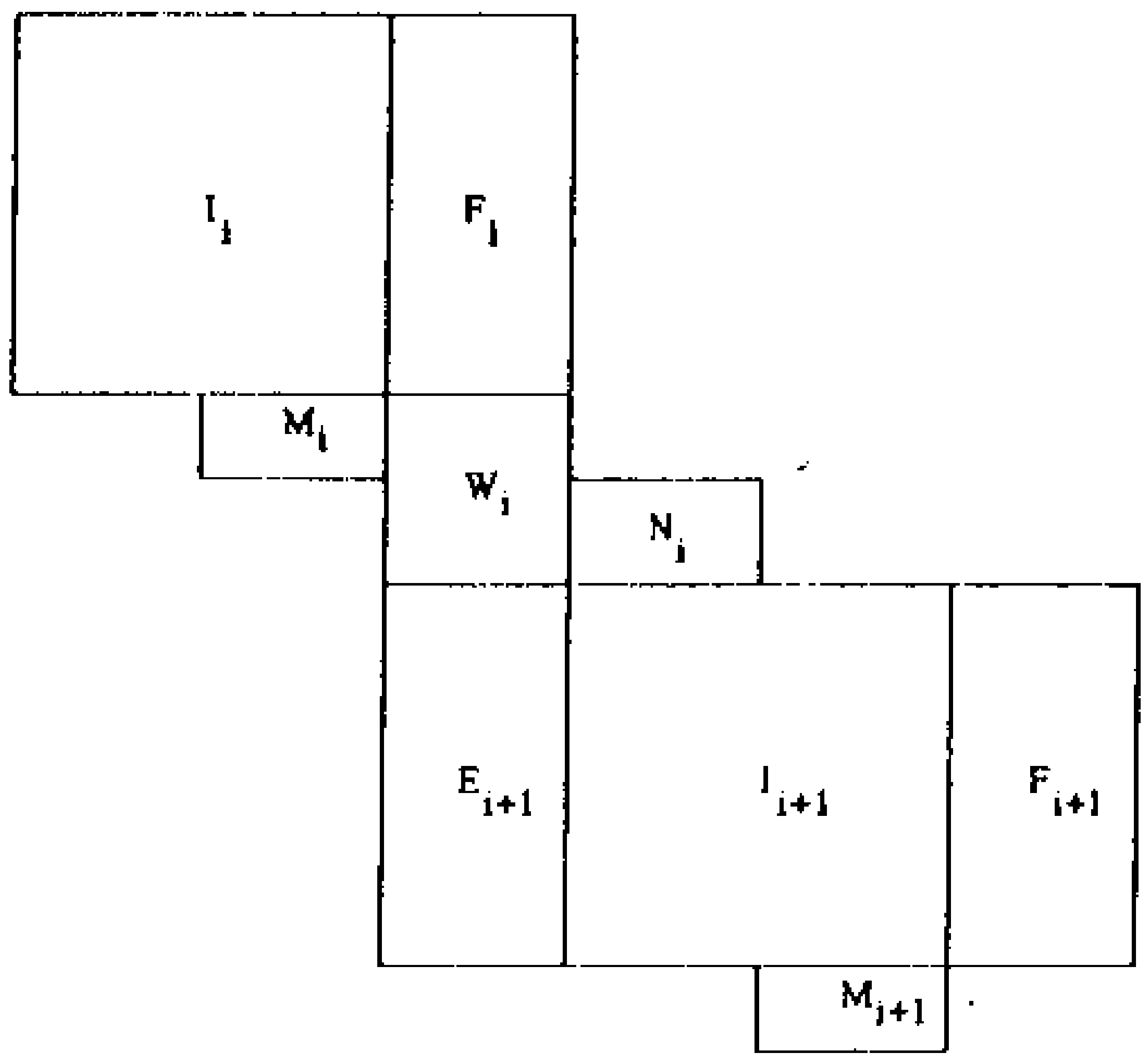

Fig. 3. System after inverse multiplication.

Interchanges in the first and last small blocks of A_i should also occur where appropriate in S_i, M_i, T_i and N_i. Interchanges in M_i and N_i can already be performed by F01LHF by a careful choice of starting pointers to the stack containing the segments. We have extended F01LHF in code ABDEXT to perform row interchanges in S_i and T_i. We need to be able to solve for successive RHS's as occur in an iteration for a nonlinear system. For this, it is necessary to keep the information about the interchanges inside the decomposed matrices A_i and to save the information about those interchanges which occurred outside the decomposed segments. This phase can be performed independently by each processor and no interprocessor communication is necessary.

Phase II Inverse multiplication

The system from Phase I is multiplied implicitly by

$$\mathrm{diag}(A_1^{-1}, I, A_2^{-1}, I, \dots, A_p^{-1}) = \mathrm{INV}$$

to give the system in *Fig. 3*. E_i and F_i are obtained by solving

$$P_i L_i U_i Q_i [E_i, F_i] = [T_i, S_i].$$

We expect E_i and F_i to be full matrices. If our original system had the form $A\underline{x} = \underline{b}$ the new system has the form

$$\tilde{A}\underline{x} = \mathrm{INV}\,\underline{b} \quad \text{where } \tilde{A} = (\mathrm{INV})A.$$

This phase can be performed independently on each processor and no interprocessor communication is necessary.

Phase III – Additional eliminations

Phase III eliminates the M_i's and N_i's using row additions in the appropriate parts of the identity matrices produced in Phase II. When M_i is eliminated the first $n - q$ rows of W_i are

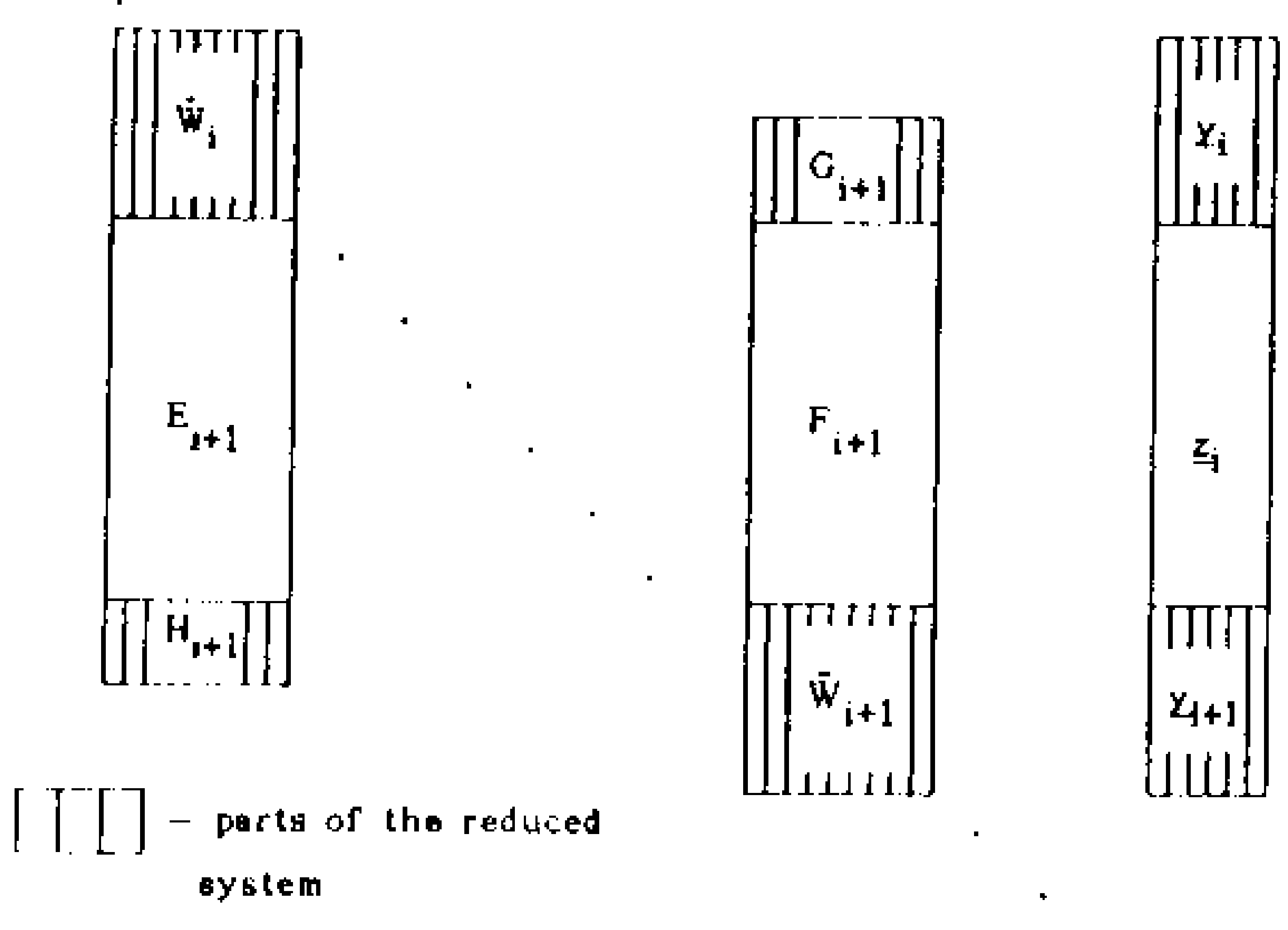

Fig. 4. System after additional eliminations.

(a)
$$\begin{bmatrix} \tilde W_1 & \tilde G_2 & & & \\ \tilde H_2 & \tilde W_2 & \tilde G_3 & & \\ & & \ddots & & \\ & & & \tilde H_{p-1} & \tilde W_{p-1} \end{bmatrix}$$

(b)

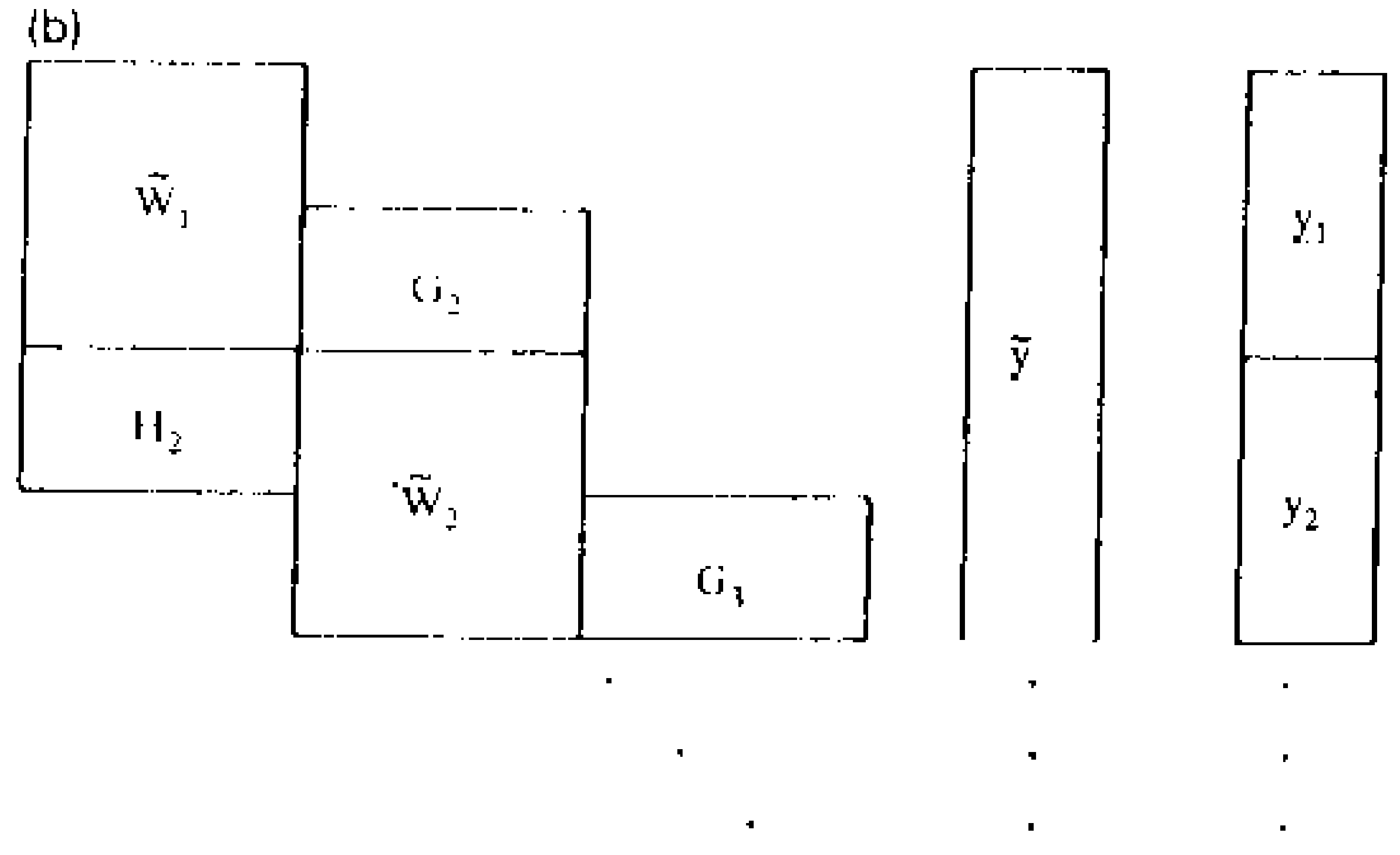

Fig. 5 (a): Reduced system represented in block tridiagonal form; (b) Reduced ABD system and its right hand side.

altered and an additional fill-in $H_i \in R^{(n-q)\times n}$ is generated. When N_i is eliminated the last q rows of W_i are altered and additional fill-in $G_i \in R^{q\times n}$ is generated (see *Fig. 4*). The altered W_i is denoted $\tilde{W}_i$. To perform the appropriate operations on the RHS's, the multipliers must be stored in an additional vector. They cannot be stored in the filled-in parts, because the values there will be necessary for Phase V (solution calculation). In Phase III a new, reduced ABD system is created (see *Fig. 5a* and *b*).

To create the reduced system on a shared memory machine no interprocessor communication is necessary. On a distributed memory computer appropriate parts of the new system must be sent to the processors which will decompose it in Phase IV.

Phase IV – Decomposition of the reduced ABD system

We decompose the reduced system created in Phase III. This system has the same ABD structure as the original, values n and q remaining unchanged. It will have P-2 blocks of size $n \times 2n$ and 2 BC-type blocks. Since we assume that the starting number of blocks k is large and the number of processors P is correspondingly moderate, it seems natural to solve this new system using one processor and the original ABD routine, F01LHF.

In the case when P is large, recursive tearing can be considered. For example, the reduced system may then the too large for the memory of one processor. Even if it is not, it seems possible that a recursive parallel algorithm may be faster than the sequential algorithm, in terms of wall-clock time. It is easy to see that this will only be the case if P, and hence the reduced system, is very large. Since we arrange our original partition into segments so that k/P is large, this scenario is unlikely.

Phase V – Solution of the reduced system

The right hand side $\underline{b}$ is multiplied by the matrix INV, in parallel. Next the multipliers stored in Phase III are used to recalculate parts $\underline{y}_i$ of the RHS (see *Fig. 4*). The $\underline{y}_i$'s are then sent to the processors used in Phase IV as the $\tilde{\text{R}}$HS of the reduced system (see $\tilde{F}$*ig.* 5b). The solution $\underline{\tilde{y}}$ of the reduced system is calculated using routine F04LHF.

Phase VI – Calculation of the solution

The solution of the reduced system effectively decouples the original matrix. After Phase V partial solutions of the original system have been calculated. In Phase VI parts of $\underline{\tilde{y}}$ (corresponding to appropriate $\underline{y}_i$'s in *Fig. 4*) are distributed to the other processors and backsubstitution is performed in each decoupled segment to compute $\underline{\tilde{z}}_i$'s (corresponding to $\underline{z}_i$'s on *Fig. 4*) which together with previously computed $\underline{\tilde{y}}_i$ constitute the complete solution of the problem. This phase can be carried out in parallel. In the case of a shared memory machine no interprocessor communication is necessary, because Phase V is performed in a global shared matrix. On a distributed memory architecture the results from Phase V must first be transmitted to processors which hold the parallel decomposition. A more detailed description of the implementation of this phase (for a banded matrix) is given in [8].

2. Arithmetic operation count

We use estimates from [5] for the cost of ABD solution. The basic unit of cost is one addition and one multiplication. To make the analysis as close as possible to computational reality, the following assumptions were made:
- n (the size of the system of ODE's) is fixed;
- k (the number of internal blocks representing the number of meshpoints in the discretization) is variable. In an adaptive BVP solver k might start small and increase;

- P (the number of processors used) can change. Really the number is fixed but one might not choose to use them all.

We have k internal blocks of size $n \times 2n$ and BC-type blocks of size $q \times n$ and $(n-q) \times n$ respectively. Suppose $k+2$ is divisible by P. Then, we have P segments each containing $[k-2(P-1)]/P$ internal blocks and 2 BC-type blocks. To account of multiple RHS's, the operation count will be calculated independently for decomposition and for solution. Workload for the first and last processors is smaller as only one-sided fill-in is calculated. There is no way to exploit this, except unequal division of blocks between segments. Thus we will count the workload equal for all processors.

2.1. *Decomposition*

Phase I

From [5] the total number of basic operations necessary to decompose a segment containing $[(k+2)/P-2]$ $n \times 2n$ blocks and 2 BC-type blocks is

$$\begin{aligned}&\left[(k+2)/P\right]\left[(5/6)n^3+(1/2)n^2q-(1/2)nq^2-n^2+(1/6)n\right]\\&\quad-(4/3)n^3-n^2q+nq^2+(3/2)n^2-(1/6)n.\end{aligned}$$

For $P=1$ this formula contains all of the costs of decomposition. The following costs are used only when $P>1$.

Phase II

The number of basic operations for back substitution per $n \times 2n$ block for one right hand side is $2n^2$, for the top BC-type block it is n^2-nq, and for the bottom BC-type block it is nq. Hence the number of operations for one right hand side is $n^2(2[(k+2)/P-2]+1)$. There are $2n$ right hand sides per segment, n each for matrices T_i and S_i. Therefore the total number of operations is $2n^3[2(k+2)/P-3]$ per processor.

Phase III

A simple estimate shows that the total number of basic operations for each segment is $2n^3$, except the first and last segments for which there are n^3 operations. The total effective number of operations for this phase is $2n^3P$

Phase IV

Using the formulas of Phase I, the total number of operations to decompose the reduced system with $(P-3)$ $n \times 2n$ blocks and 2 BC-type blocks is

$$\begin{aligned}&P\left[(5/6)n^3+(1/2)n^2q-(1/2)nq^2-n^2-(1/6)n\right]\\&\quad-(4/3)n^3-n^2q+nq^2+(3/2)n^2-(1/6)n.\end{aligned}$$

Hence the effective total number of operations for decomposition of the segmented ABD system is

$$\begin{aligned}&\mathrm{TOTAL}(k,n,q,P)\\&\quad=\left[(k+2)/P\right]\left[(29/6)n^3+(1/2)n^2q-(1/2)nq^2-n^2+(1/6)n\right]\\&\qquad+P\left[(17/6)n^3+(1/2)n^2q-(1/2)nq^2-n^2+(1/6)n\right]\\&\qquad-(26/3)n^3-2n^2q+2nq^2+3n^2-(1/3)n.\end{aligned}$$

There are two specific and commonly occuring cases which can be considered 'extreme' from the point of view of workload. The first is $q=n/2$ (the BC's are equally balanced between left and right) and the second is $q = 1$ (almost all BC's are on the right). For these cases the total work is

$$\begin{aligned}\mathrm{TOTAL}(k, n, n/2, P) = {} & [(k+2)/P][(119/24)n^3 - n^2 + (1/6)n] \\ & + P[(71/24)n^3 - n^2 + (1/6)n] - (55/6)n^3 + 3n^2 \\ & -(1/3)n, \qquad (2.1)\end{aligned}$$

$$\begin{aligned}\mathrm{TOTAL}(k, n, 1, P) = {} & [(k+2)/P][(29/6)n^3 - (1/2)n^2 - (1/3)n] \\ & + P[(17/6)n^3 - (1/2)n^2 - (1/3)n] - (26/3)n^3 + n^2 \\ & + (5/3)n. \qquad (2.2)\end{aligned}$$

The cost with almost all BC's on the left ($q = n - 1$) is the same as (2.2).

For any $n > 2$ and any values of P and k the leading order terms in (2.1) and (2.2) capture at least 90% of the total cost. Therefore we can reasonably add the assumption that n is large enough to keep only the highest order terms, that is terms of $O(n^2)$ can be neglected. The approximate arithmetic costs are then

$$\begin{aligned}\mathrm{TOTAL}(k, n, q, P) = {} & [(k+2)/P](29/6)n^3 + (1/2)n^2q - (1/2)nq^2 \\ & + P[(17/6)n^3 + (1/2)n^2q - (1/2)nq^2] - (26/3)n^3 \\ & - 2n^2q + 2nq^2\end{aligned}$$

which for the 'extreme' distributions of boundary conditions is

$$\mathrm{TOTAL}(k, n, n/2, P) = \{(119/24)(k+2)/P + (71/24)P - (56/6)\}n^3, \qquad (2.3)$$

$$\mathrm{TOTAL}(k, n, 1, P) = \{(29/6)(k+2)/P + (17/6)P - (26/3)\}n^3. \qquad (2.4)$$

The 'optimal' number of processors to minimize the workload in these latter cases is

$$P_{\mathrm{OPT}}(n/2) = \sqrt{(119/71)(k+2)}, \qquad P_{\mathrm{OPT}}(1) = \sqrt{(29/17)(k+2)}$$

In *Table 1*, P_{OPT}, segment size and total workload is given for $k = 58, 118, 238, 478$. This number of internal blocks per segment was chosen to allow balanced usage of all 20 processors on a Sequent Symmetry.

2.2. *Solution phases*

Phase V

The cost of calculating the new RHS (the multiplication INV $\underline{b}$ can be performed in parallel) is $2n^2[(k+2)/P - 3]$ per processor. After this, the cost of solving the reduced system is $2n^2(P - 3)$ on one processor.

Table 1
Optimal number of processors for a given number of blocks. (Segment size measured in number of internal $n \times 2n$ blocks.)

$q = n/2$				$q = 1$			
k	P_{OPT}	Segment size	Total	k	P_{OPT}	Segment size	Total
58	10	4	$51n^3$	58	10	4	$49n^3$
118	14	6	$75n^3$	118	14	6	$73n^3$
238	20	9	$110n^3$	238	20	9	$106n^3$
478	28	14	$159n^3$	478	29	13	$154n^3$

Phase VI

The cost of calculating the solution of the system (which can be performed in parallel) is $2n^2[(k+2)/P - 3]$ per processor.

Hence, the cost of the solution phases for r RHS's is $r[4(k+2)/P + 2P - 18]n^2$. Observe that this is q independent; that is the distribution of BC's is irrelevant.

For the interesting case $n \gg r$, the cost of decomposition is higher than for solution. Therefore the number of processors used should be based on the demands of decomposition. Overestimation of the number of processors leads to a smaller increase of the decomposition workload than underestimation (see Fig. 6 and 7). When k, and and hence P_{OPT} varies, if we cannot vary P accordingly, P should be chosen to overestimate any likely values of P_{OPT}.

2.3. Comparison with other algorithms

If our ABD was solved sequentially (using, say, F01LHF and F04LHF) the total cost of decomposition would be

$$\begin{aligned}\mathrm{TOTAL}'(k, n, q) = k\{(5/6)n^3 + (1/2)n^2q - (1/2)nq^2 - n^2 + (1/6)n\}\\ + (1/3)n^3 - (1/2)n^2 + (1/6)n.\end{aligned}$$

For the 'extreme' cases this is

$$\begin{aligned}&\mathrm{TOTAL}'(k, n, n/2)\\ &\quad = k\left[(23/24)n^3 - n^2 + (1/6)n\right] + (1/3)n^3 - (1/2)n^2 + (1/6)n,\\ &\mathrm{TOTAL}'(k, n, 1)\\ &\quad = k\left[(5/6)n^3 - (1/2)n^2 - (1/3)n\right] + (1/3)n^3 - (1/2)n^2 + (1/6)n.\end{aligned}$$

For $n > 9$ the leading order term capture no less than 90% of the total cost. Therefore for n large, the approximate workload is

$$\begin{aligned}&\mathrm{TOTAL}''(k, n, n/2) = \left[(23/24)k + 1/3\right]n^3,\\ &\mathrm{TOTAL}''(k, n, 1) = \left[(5/6)k + 1/3\right]n^3.\end{aligned}$$

These costs are tabulated in *Table 2* for the same values of k as in *Table 1*. The cost of the solution phase is $r(2k-3)n^2$.

Consider now the D&J parallel algorithm [7, §5] for banded systems. It was designed under the assumption that no interchanges are necessary (as for instance for positive definite or strictly diagonally dominant matrices). Because most ABD's require interchanges for stability we cannot use it directly. We could use the algorithm proposed by Wright [15] which is an extension of the D&J algorithm which allows for interchanges. However, here we cost the D&J algorithm with interchanges, assuming the ABD is included inside a banded system of minimum bandwidth. Below, we refer to this as the modified D&J algorithm. The resulting cost will certainly be no more than for Wright's approach. Using the formulas from [7] the

Table 2
Arithmetic cost of decomposition using the sequential method.

k	$q = n/2$	$q = 1$
58	$56n^3$	$49n^3$
118	$114n^3$	$99n^3$
238	$229n^3$	$199n^3$
478	$459n^3$	$399n^3$

Table 3
Workload of modified D&J algorithm; results in the last column represent workload for $q = n/2$ and $q = 1$ respectively.

k	P_{OPT}	Segment size	Total
58	14	2	$201n3\backslash 476n$
118	19	4	$302n^3\backslash 715n^3$
238	27	7	$444n^3\backslash 1052n^3$
478	38	10	$645n^3\backslash 1528n^3$

Key to Figures 6–8

□–58 blocks
△–478 blocks

——— parallel algorithm
- - - - - - - - serial algorithm
· · · · · · · · · · D&J algorithm

Key to Figures 9–13

——— 58 internal blocks
· · · · 118 internal blocks
· · · · · · 238 internal blocks
- - - - - - - 478 internal blocks

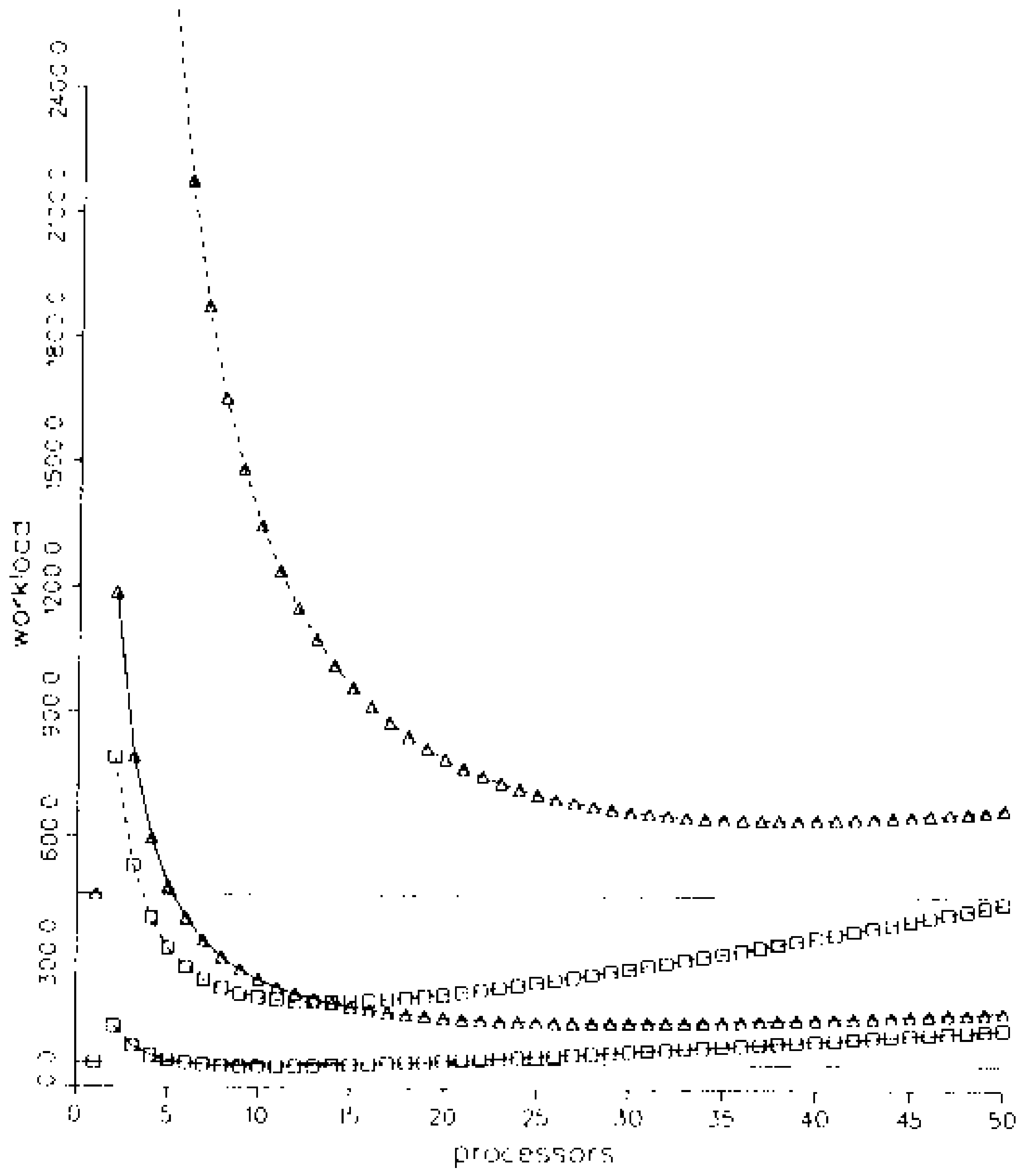

Fig. 6.

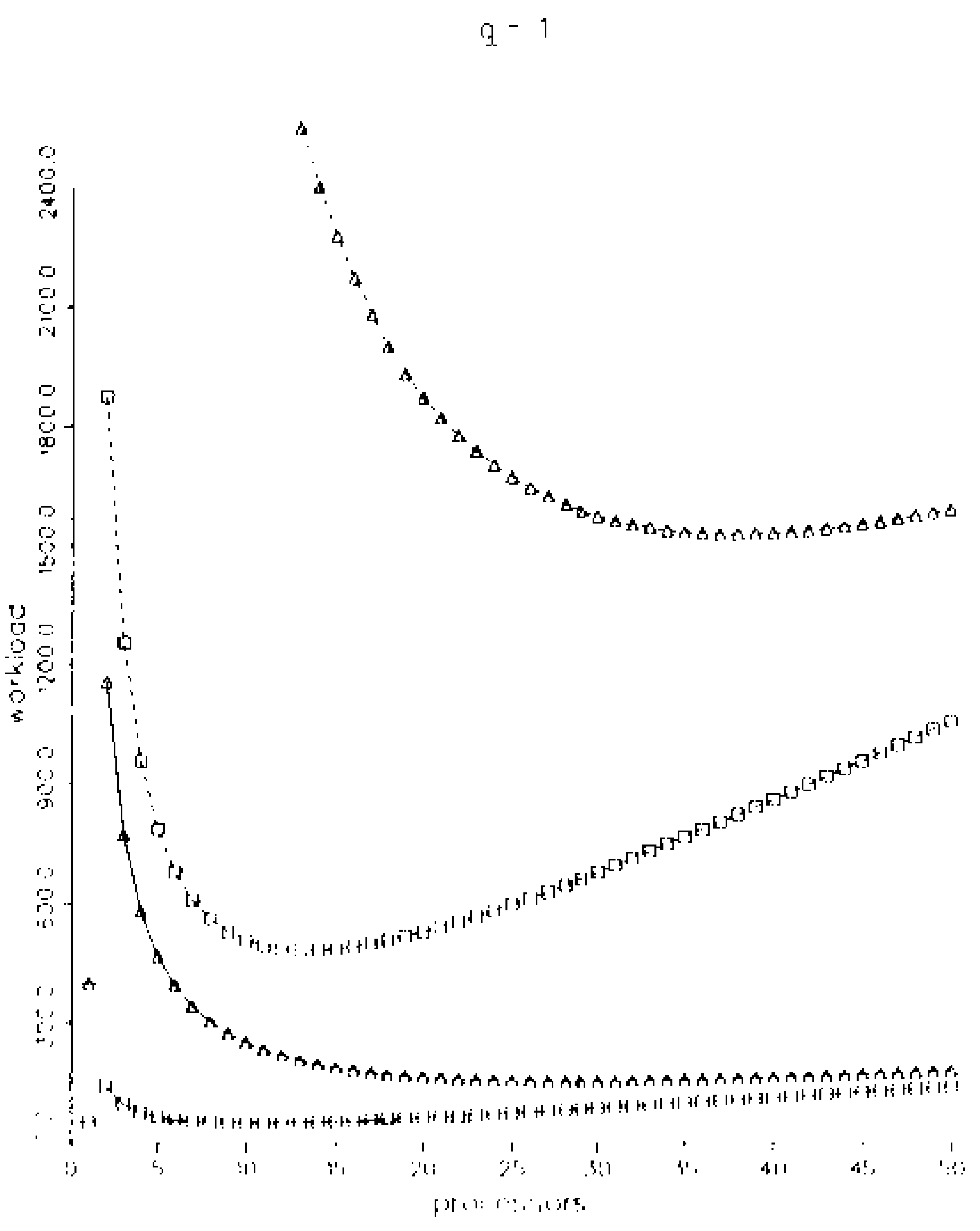

Fig. 7.

number of operations to decompose the system is

$$\mathrm{TOTAL}(k, n, q, P) = (k/P)(8m^3) + P(8/3)m^3 - 11m^3 + (1/3)m$$

where $m = 2n - q$ is the half-bandwidth.

For $n > 1$ the $O(n^3)$ terms capture no less than 90% of the total cost. Under the assumption that n is large enough, in the extreme cases the cost is approximately

$$\mathrm{TOTAL}(k, n, 1, P) = [64(k/P) + (64/3)P - 88]n^3$$

$$\mathrm{TOTAL}(k, n, n/2, P) = [27(k/P) + 9P - (297/8)]n^3.$$

The minimum of both these functions is at $P_{\mathrm{OPT}} = \sqrt{3k}$. The total workload for the modified algorithm using P_{OPT} processors is given in Table 3 using values of k corresponding to *Table 1*. The cost of the solution phase is $rm^2[8(k/P) + 3P - 9]$.

For large n, and $r = 0$, the arithmetic costs of the three algorithms are illustrated in *Figs. 6* and *7* (with $q = n/2$ in *Fig. 6* and $q = 1$ in *Fig. 7*). We have shown the costs for the range $P = [2, 3, \ldots, 50]$ and for $k = 58$ and 478.

Clearly, if more than P_{OPT} processors are available it may be better not to use them all; additional processors will increase the run time instead of decreasing it. This is especially true when the total number of internal blocks is small. It is an even more serious problem for the modified D&J algorithm. However the penalty for using slightly more than P_{OPT} processors is

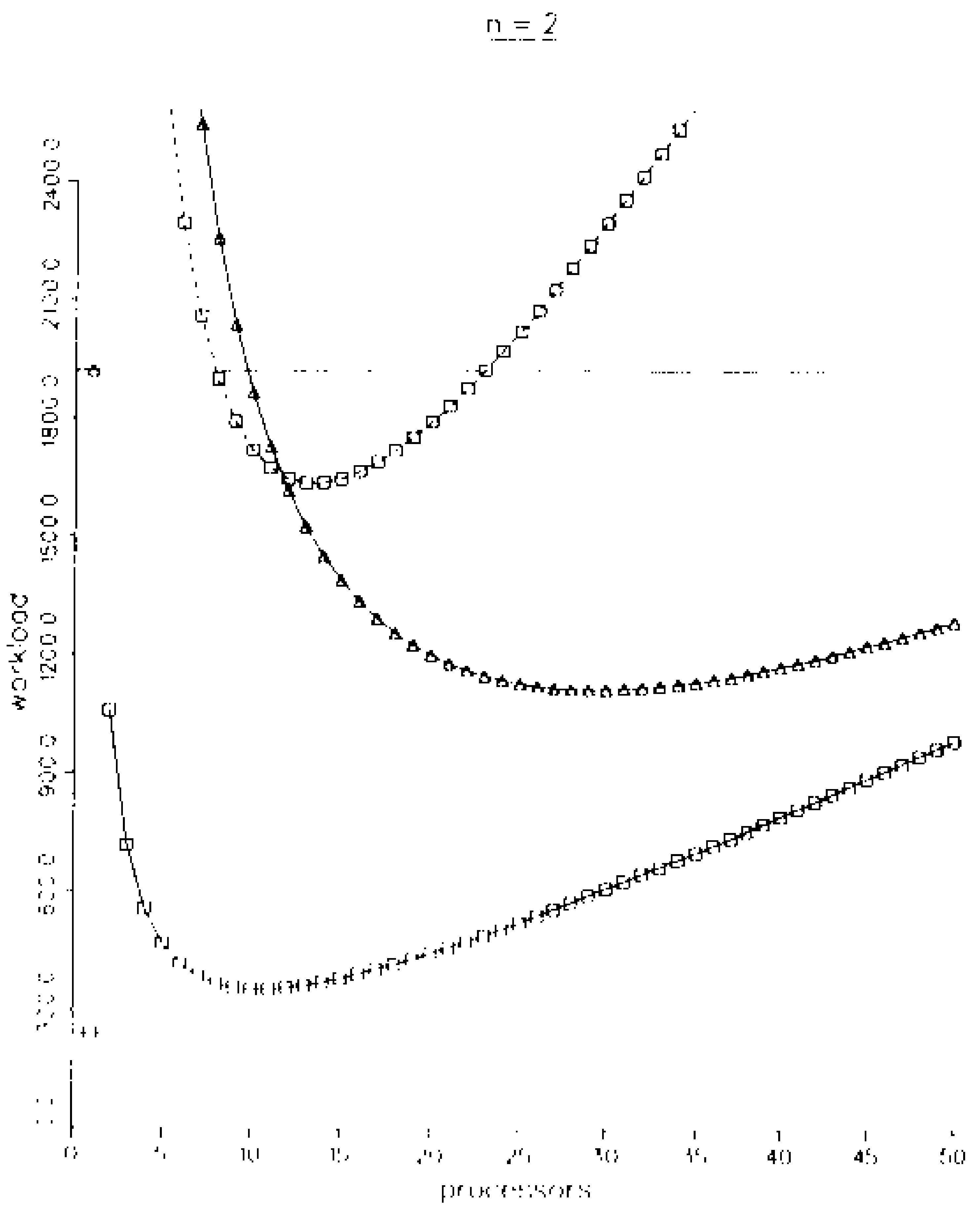

Fig. 8.

not large. When the number of processors used is smaller than P_{OPT} the penalty, if avoidable, can be more severe.

Similar remarks apply for small systems; see *Fig. 8* which presents the workloads for $n = 2$ and $q = 1$. The workload for the modified D&J algorithm when $k = 478$ gave values too large to be captured on the figure.

In *Table 4*, for n large we compare of the costs of the decomposition in the other algorithms to those of our parallel one. We assume the parallel ABD and the D&J algorithms use their own optimal number of processors for each given number of blocks. Indeed, for $P \sim O(\sqrt{k})$ processors the limit as $k \to \infty$ of the ratios of the cost of the sequential algorithm to the parallel algorithm are $.12\sqrt{k+2}$ and $.11\sqrt{k+2}$ for $q = n/2$ and $q = 1$ respectively. This is in quite

Table 4
Ratios of costs of the sequential and the modified D&J algorithms to the parallel algorithm

Sequential approach			Modified D&J algorithm	
k	$q = n/2$	$q = 1$	$q = n/2$	$q = 1$
58	1.10	1.00	3.94	9.71
118	1.52	1.35	4.02	9.79
238	2.08	1.87	4.03	9.92
478	2.89	2.59	4.05	9.92

good agreement with the first two columns of *Table 4*. The results of Table 4 and *Figs. 6–8* demonstrate that the modified D&J algorithm is uncompetitive for BVP systems. We will not consider it further.

3. Additional topics

3.1. Interprocessor communication

Two cases should be distinguished. On a shared memory compute no 'interprocessor' communication is necessary. Phases I, II and III are performed on one global shared memory matrix. Each processor is operating on a set of shared variables and calculates its private pointers when necessary. At the end of Phase III all processors are filling a separate part of the global shared vector which represents the reduced system. This operation also does not require interactions between processors. The start of Phase IV is the first moment which requires the synchronization because, in principle, all of the parts of the reduced system must be in place to decompose it. One could start decomposition as soon as the first two blocks are in place (a frontal approach). This would lead to a very complicated code and we are not convinced that such a price is worth paying because of the expected small size of this system. Similar comments can be made for computing the solution of the reduced system in Phase V. Phase VI, which involves computing the solution of the large system, can be performed independently by all of the processors using the information from the global shared vectors.

The situation is slightly different on a distributed memory architecture. Before the algorithm starts all processors must receive their torn parts of the system. Each of them must also have some free memory space to create the necessary additional vectors. Assuming that the optimal number of the processors will be used and each has enough local memory, Phases I to III are performed independently on all processors. At the start of Phase IV all the necessary information must have been communicated to the processor which will decompose the reduced system. After the solution of the reduced system is computed in Phase V, at the start of Phase VI the partial solution must be communication to the processors which calculate the solution of the original system.

There is considerable interprocessor communication at the end of Phase III and, to a lesser extent, at the start of Phase VI. We have made no attempt to include these costs. Note, however, that while about $2Pn^2$ elements are communicated to a single processor in Phase III, about $(23/24)Pn^3$ arithmetic operations are performed in this phase. There is a similar favorable imbalance in Phase VI. Thus for large n, arithmetic costs are likely to dominate.

3.2. Memory requirements

In the case of a distributed memory configuration the size of memory connected with each processor plays an important role. Each processor needs to be able to keep its segment of the ABD. The memory requirement for the first and last segments are smaller, but there is no simple way to exploit this fact. Therefore memory demand will be calculated for all processors equally. In Phase I the size of the segment is $2(k+2)/Pn^2$. The fill-in generated in Phase III is $2[(k+2)/P-1]n^2$. The multipliers in Phase IV demand $4n^2$ elements. Therefore the minimum memory requirement per processor is $[4(k+2)/P+2]n^2$ elements. Once this memory is 'used' it cannot be 'reused' because the decomposition information is necessary to calculate results for multiple RHS's.

This size of the reduced system in Phase IV is approximately $[2(P-2)+1]n^2$ elements. Therefore the total memory requirement (where it is assumed the reduced system fits on one

Table 5
Number of processors to balance memory requirements

k	P_B
58	12
118	17
238	23
478	32

processor) is $[4k + 4P + 5]n^2$ words on a total of $P + 1$ processors. It is easy to see that if n is large enough and $P \sim O(\sqrt{k})$, then as $k \to \infty$ the *total* memory required for the parallel algorithm is twice that for the sequential algorithm.

Increasing the number of processors used increases the total memory demand, but the memory required per processor decreases. When n is large we may balance memory demands between the requirements of the segments and the requirements of the reduced system. We do this by solving $[4(k + 2)/P + 2]n^2 = [2(P - 2) + 1]n^2$ for P_B. This gives $P_B = (5 + \sqrt{89 + 32k})/4$ processors. See *Table 5* which should be compared with *Table 1*. P_B is slightly larger than P_{OPT}. It is natural to choose the number of processors to maximize the speedup; that is $P = P_{OPT}$. However, as shown in *Figs. 6, 7* and *8*, the penalty for using a slightly larger number of processors to satisfy memory requirements, if necessary, is not severe.

If the memory available per processor is smaller than is required to use P_{OPT} processors then the system should be segmented between a larger number. However, we must be able to keep approximately $4n^2$ words an each processor. Also we must have enough memory to solve the reduced system created in Phase IV of the algorithm. When the number of processors increases the size of the reduced system increases. For all $P > 3$ the size of the reduced system is larger than $4n^2$ words. For large enough n and $P \cdot P_{OPT}$, this implies using tearing to solve the reduced system.

4. Simulation

4.1. *Theoretical problem simulation*

We present our simulation in terms of relevant parameters arising in the solution to realistic problems but our results are for the parallel linear algebra only. Our simulations are intended to reflect the situation of solving large, difficult nonlinear BVP's. Assume the solution is started with 26 meshpoints. (This allows us to divide exactly the number of internal blocks by the optimal number of processors. In all cases for 26 internal blocks the optimal number of processors is 7, with two internal blocks per segment.) Simple doubling of the mesh is used to calculate a series of solutions. Assume we need 5 consecutive mesh doublings. For coarse meshes a large number of iterations is usually necessary. When the number of mesh points increases the number of iterations usually decreases. Assume that for 26 and 52 meshpoints 8 iterations are performed. Similarly for 104 and 208 meshpoints assume 4 iterations are necessary and for 416 and 832 meshpoints 2 iterations suffice. This means solving for 8, 8, 4, 4, 2 and 2 RHS's respectively. Suppose that each step is calculated by the sequential approach and by the parallel algorithm. For each step of the parallel algorithm the optimal number of processors is used. This choice favors the parallel algorithm over using any fixed number of processors for the whole BVP solution. For a small system ($n = 2$), *Table 6* contains the ratios of costs of solving the problem using the sequential approach to solving the same problem as using the parallel algorithm. The results after the (\) represent the situation with a limited

Table 6
Small system simulation

k	P_{OPT}	Relative cost
26	7	2.76
52	10	3.16
104	14	3.25
208	19	4.26
416	28\20	4.56\4.28
832	40\20	6.26\4.96
Total relative cost		4.55\4.17

Table 7
Larger system simulations

k	$q = n/2$			$q = 1$		
	P_{OPT}	Cost $n = 10$	Cost $n = 100$	P_{OPT}	Cost $n = 10$	Cost $n = 100$
26	7	1.53	0.89	7	1.50	0.82
52	10	1.93	1.17	10	1.90	1.08
104	14	2.04	1.50	14	1.97	1.37
208	19	2.75	2.05	19	2.66	1.87
416	26\20	3.20\3.06	2.75\2.64	27\20	3.06\2.91	2.50\2.39
832	38\20	4.44\3.62	3.83\3.16	38\20	4.24\3.45	3.47\2.85
Total		3.19\2.94	2.64\2.44		3.06\2.81	2.40\2.21

number (20) of processors. For a moderate-sized system ($n = 10$) and a large system ($n = 100$), the results are summarized in *Table 7*.

Using the formulas in Section 3.2 for the total memory, we have calculated the minimal memory requirement (per processor and in total) for the simulation described above. *Table 8* summarizes the memory requirements in words/n^2. The approximate factor of two between the memory requirements of the parallel and sequential algorithms is evident.

4.2. *Practical simulation*

To confirm our theoretical predictions we performed a computational simulation. We coded the parallel algorithm using Force directives [11] and tested it on a Sequent Symmetry with 20 processors.

To make a fair comparison the parallel code was tested against the best sequential algorithm, routine F01LHF from the NAG library. The numbers of blocks were chosen to match the

Table 8
Comparison of memory requirements; PP memory requirement per processor, RSS reduced system size, SEQ - memory requirement for the sequential algorithm.

k	P_{OPT}	PP	RSS	Total	SEQ
26	7	18	11	137	53
52	10	24	17	253	105
104	14	33	25	477	209
208	19	47	35	913	417
416	26\20	67\86	49\37	1773\1749	833
832	38\20	90\169	73\37	3485\3413	1665

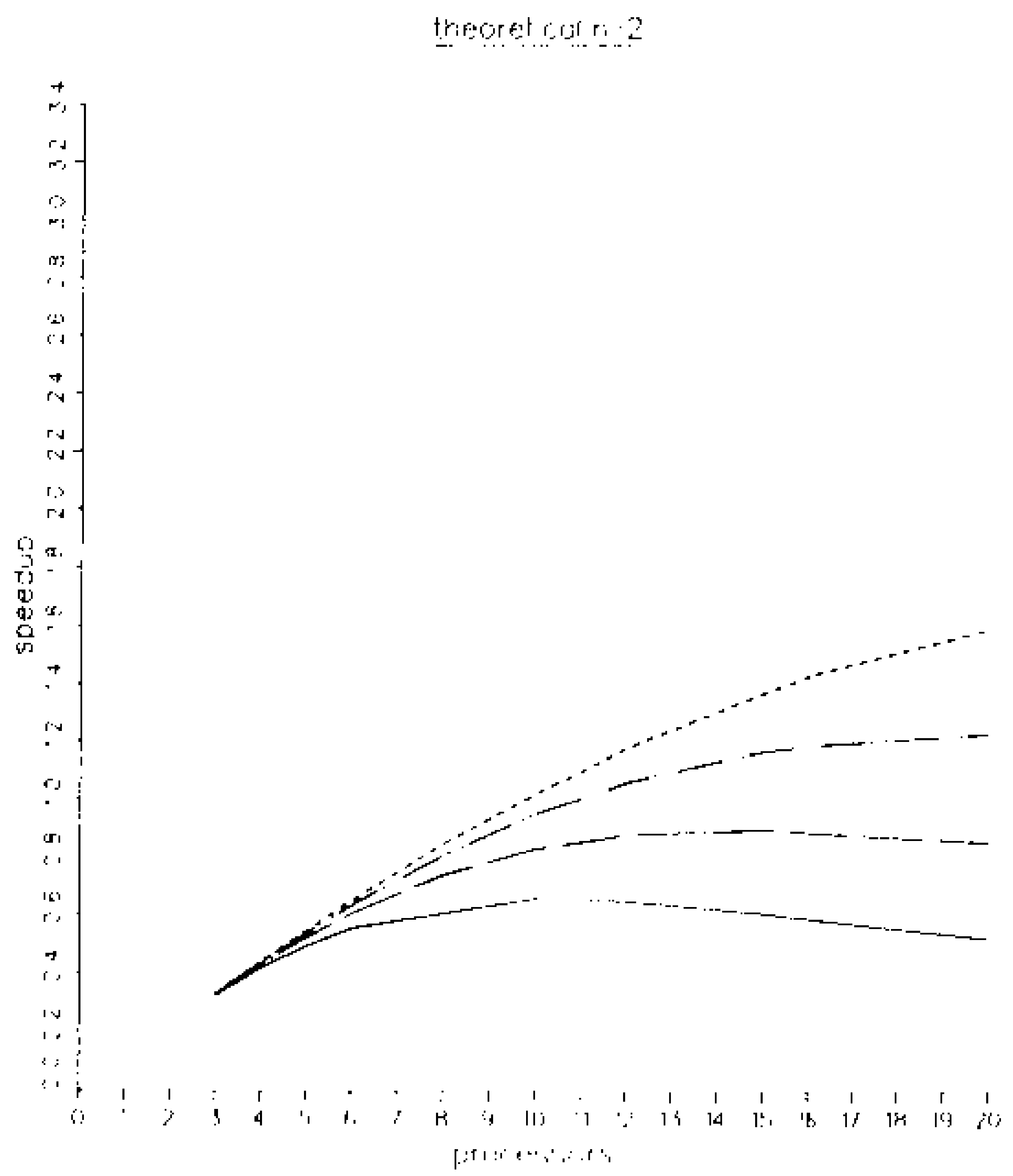

Fig. 9.

theoretical predictions made in Section 2. Tests were performed only for equal divisions of the total number of blocks. This implies that first and last processors waited for the other processors to 'catch up', but it makes the overall picture much clearer (preventing erratic behavior). Comparisons were made for the most expensive part of the process, decomposition of the system, to match results from Section 2.2. Tests were performed for $q = n/2$, which 'extreme' case is more common is computational practice.

Fig. 9 summarizes the theoretical speedups for $n = 2$ with all terms included in the formula, whereas *Fig. 10* presents speedups achieved in the test. *Fig. 11* summarizes the theoretical speedups for $n = 10$ (the largest system with 478 blocks that fits into memory) with all terms included in the formula. *Fig. 12* summarizes the theoretical results when only the $O(n^3)$ terms were included. *Fig. 13* presents speedups achieved in practice.

As predicted earlier, the behavior when $n = 2$ cannot be approximated by $O(n^3)$ terms only (compare *Figs. 9, 10* and *12*). The erratic behavior of the practical test for $n = 2$ can be probably explained by the system overhead (visible especially for small problems). When $n = 10$ test results are very close to the truncated approximation. The influence of the lower order terms is close to negligible. The timings of the sequential code behave as predicted. They double (with some small overhead) when the number of internal blocks is doubled.

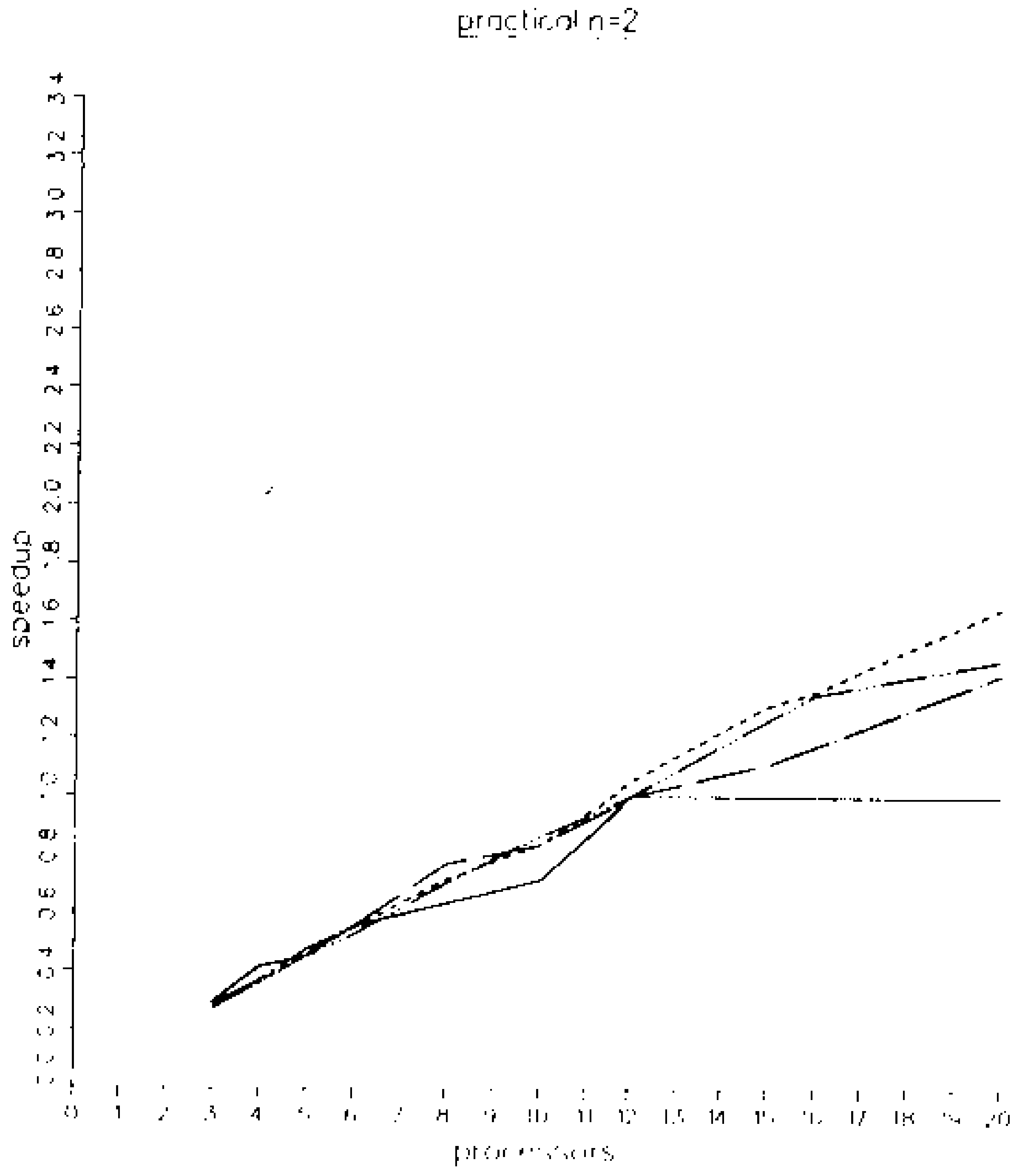
practical n=2
speedup
processors

Fig. 10.

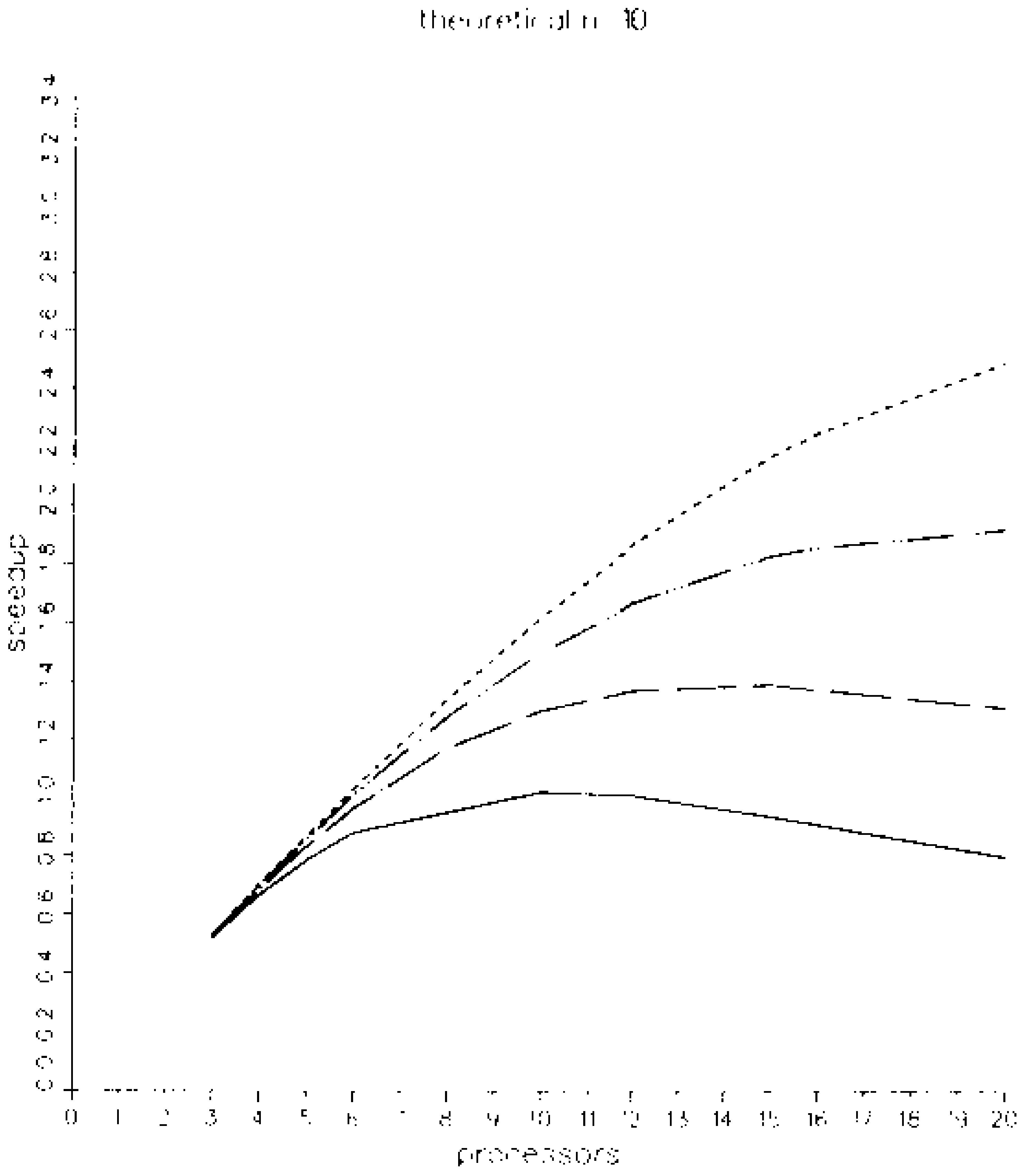
theoretical n=10
speedup
processors

Fig. 11.

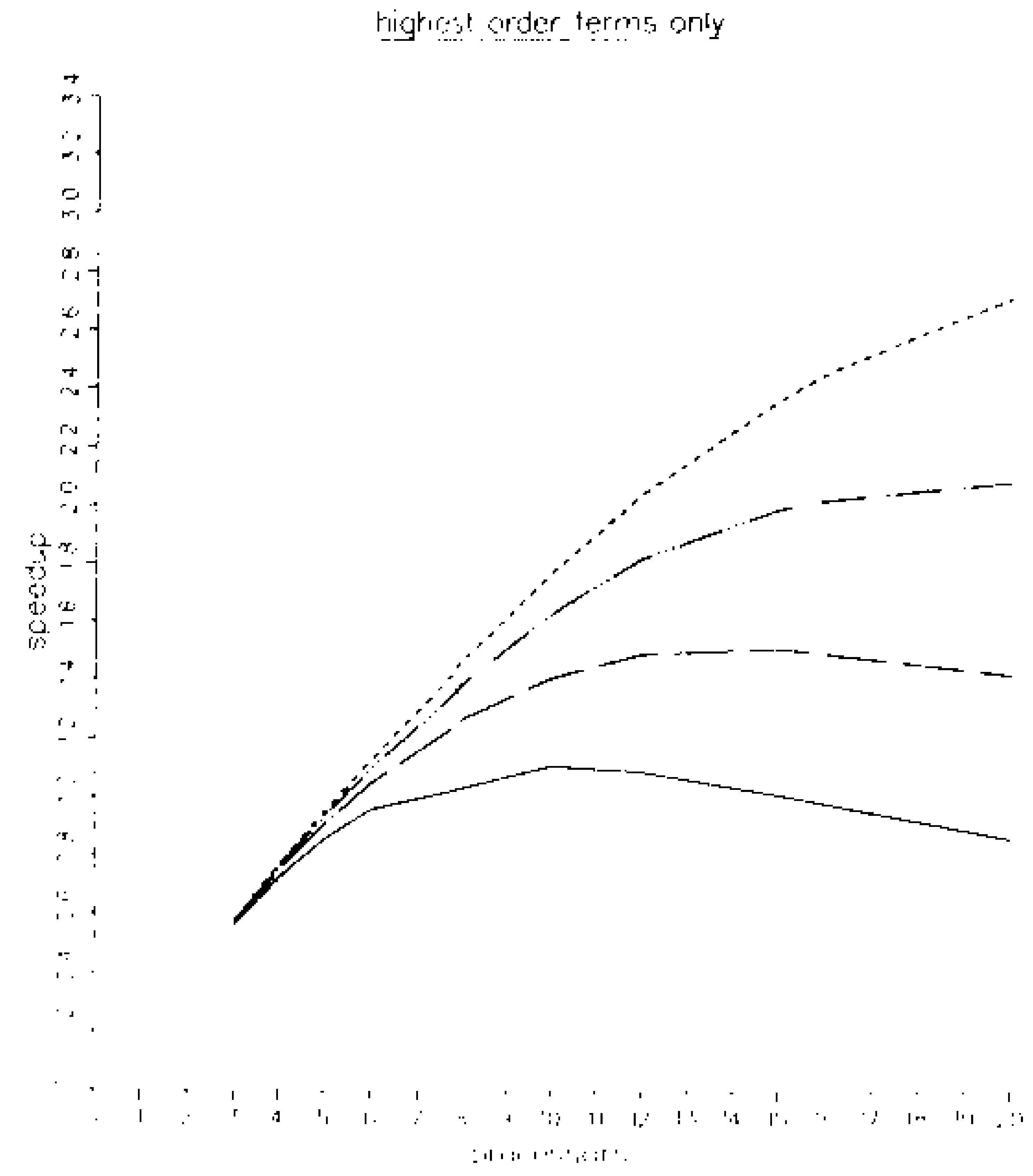
highest order terms only
speedup
processors

Fig. 12.

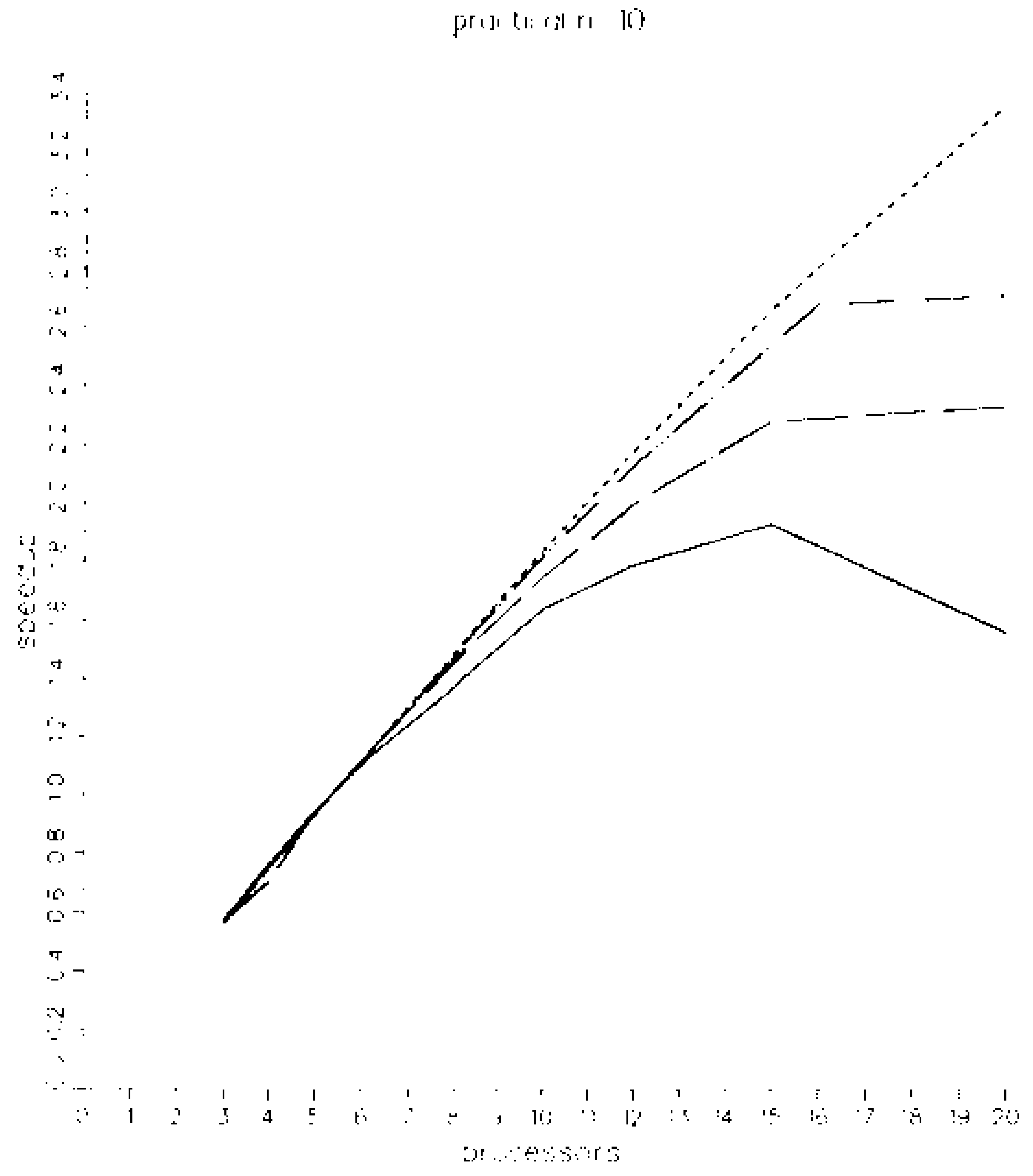
practical m 10
speedup
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
processors

Fig. 13.

For a large number of processors the speedups are slightly better than predicted theoretically. This can probably be explained as the effect of 'cache' memory, which on the Sequent Symmetry is added to each processor. When the number of processors increases the size of segments decreases and the amount of communication with the global memory decreases. That is, we get more than just the arithmetic cost reductions. Even if this effect is machine dependent it has three important consequences. First, it suggests P_{OPT} should be used as a *lower bound* for the number of processors, if feasible. Second, it suggests considering memory balance requirements as an alternative criterion for choosing the number of processors. Third, it implies a distributed memory model may be a good predictor for shared memory cache machines.

Tests were also performed to check the stability of the proposed algorithm. For well-conditioned matrices (regardless of the size of the internal blocks and the number of such blocks) the parallel code produced answers which differed (in Euclidean norm) by a small multiple of machine precision from the results computed by the (stable) sequential code.

5. Conclusions

We have developed a parallel algorithm for Almost Block Diagonal systems. This algorithm is is based on 'tearing' the system into smaller almost block diagonal segments which can be decomposed in parallel. This algorithm maximises the exploitation of the structure of the underlying system. It is shown to be vastly superior to using tearing on the narrowest banded system containing the almost block diagonal structure.

We have presented a realistic simulation of the solution of almost block diagonal systems arising in a finite difference approximation to an ordinary differential equation boundary value problem. Our best theoretical speedups over our sequential algorithm are about the factor of 4 on small systems when using up to 40 processors. For large systems our theoretical speedup is about a factor of 3. This and some limiting observations demonstrate that the tearing algorithm is unlikely to be competitive in terms of speedup in finite difference calculations. However, it may be useful as an alternative to a sequential algorithm, when memory limitations preclude the latter's use. In a later paper we will present more promising results for higher order spline collocation techniques.

Acknowledgement

The authors wish to thank Gertrud Kraut for her assistance with producing the figures.

References

[1] U. Ascher, J. Christiansen and R.D. Russell, Collocation software for boundary value ODEs, *ACM Trans. Math. Soft.* 7 (1981) 209–229.

[2] U. Ascher, R.M.M. Mattheij and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential* (Prentice-Hall, Englewood Cliffs, NJ, 1988).

[3] R.W. Brankin and I. Gladwell, Codes for almost block diagonal systems, *Comput. Math. Appl.* 19 (1990) 1–6.

[4] C. DeBoor and R. Weiss, SOLVEBLOK: A package for solving almost block diagonal linear systems, *ACM Trans. Math. Soft.* 6 (1980) 80–87.

[5] J.C. Diaz, G. Fairweather and P. Keast, FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column eliminations, *ACM Trans. Math. Soft.* 9 (1983) 358–375.

[6] J.J. Dongarra, J. Du Croz, S. Hammarling and R. Hanson, An extended set of Fortran basic linear algebra subprograms, *ACM Trans. Math. Soft.* 14 (1988) 1–17.

[7] J.J. Dongarra and L. Johnsson, Solving banded systems on a parallel processor, *Parallel Comput.* 5 (1987) 219-246.

[8] J.J. Dongarra and A.H. Sameh, On some parallel banded system solvers, *Parallel Comput.* 1 (1984) 223-235.

[9] I. Gladwell and R.I. Hay, Vector- and parallelization of ODE BVP codes, *Parallel Comput.* 12 (1989) 343–350.

[10] R.I. Hay, The impact of vector processors on boundary value codes, Ph.D. Thesis, University of Manchester, England, 1986.

[11] H.F. Jordan, M.S. Benten and N.S. Arenstorf, *Force User's Manual* (University of Colorado, Boulder, Colorado, 1986).

[12] H.B. Keller, *Numerical Solution of Two Point Boundary Value Problems* (SIAM, Philadelphia, 1976).

[13] NAG Fortran Library Manual, Mark 13 (1988), NAG Ltd., Wilkinson House, Jordanhill Road, Oxford, OX2 8DR, U.K.

[14] J.M. Varah, Alternate row and column elimination for solving certain linear systems, *SIAM J. Numer. Anal.* 13 (1976) 71–75.

[15] S.J. Wright, A parallel algorithm for banded linear system, to appear.