Journal of Computational and Applied Mathematics 45 (1993) 181–189
North-Holland

CAM 1284

Parallel solution of almost block diagonal systems on the CRAY Y-MP using level 3 BLAS *

Ian Gladwell ** and Marcin Paprzycki ***
Department of Mathematics, Southern Methodist University, Dallas, TX, United States

Received 18 October 1991
Revised 24 April 1992

Abstract

Gladwell, I. and M. Paprzycki, Parallel solution of almost block diagonal systems on the CRAY Y-MP using level 3 BLAS, Journal of Computational and Applied Mathematics 45 (1993) 181–189.

In a recent publication (1992), the authors showed how efficient a new level 3 BLAS algorithm for almost block diagonal systems could be using just *one* processor of a CRAY Y-MP. Here they compare the corresponding results for up to eight processors using standard CRAY Library parallel implementations of the level 3 BLAS.

Keywords: Parallel linear algebra; boundary value problems; level 3 BLAS.

1. Introduction

The aim of this paper is to discuss experiments on a CRAY Y-MP with an algorithm for the parallel solution of almost block diagonal (ABD) linear systems arising when a two-point boundary value problem (BVP) for ordinary differential equations (ODEs) is discretized along with its boundary conditions (BCs) using box-type difference schemes. There are two general approaches to the solution of this problem. The first is based on "tearing-type" strategies. The idea behind these methods is to divide the system into smaller parts and decompose each of them separately (the price is the fill-in generated). In the next step the smaller (reduced) system, of structure similar to the original one, is set up and decomposed (typically on one

Correspondence to: Prof. I. Gladwell, Department of Mathematics, Southern Methodist University, Dallas, TX 75275-0156, United States.

* This work was supported by a grant of computer time from CRAY Research Inc., which is gratefully acknowledged.

** This author wishes to acknowledge the support of NATO grant CRG 920037.

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

0377-0427/93/$06.00 © 1993 – Elsevier Science Publishers B.V. All rights reserved

processor but possibly recursively). There exist a variety of such methods which differ by the particular division and decomposition strategies (see [1,13,16]). The performance of the tearing algorithm improves when the number of internal blocks k in the ABD increases, whereas it is essentially unaffected when the size of these internal blocks n increases.

When using current BVP algorithms (for example PASVA 3 [8]), k, representing the number of meshpoints in the discretization of the BVP, may be large but will not grow unrestrictedly. At the same time, n, the number of differential equations, may also be large. The arithmetic cost functions for ABD solvers are dominated by terms in k and n^3. (The total cost is affected more by increases in the number of equations than the number of meshpoints.) The approach here is designed to be efficient in the case when n is large.

In [12] an algorithm is proposed based on level 3 BLAS [5]. It is a version of block Gaussian elimination similar to that proposed in [9]. To assure stability and, at the same time, to avoid fill-in, it uses alternating row and column pivoting, similar to that proposed in [15]. In [14] results of experiments on a one-processor CRAY Y-MP are reported. It is shown that, even on one processor, for large enough blocksizes n, code taking full advantage of the hierarchical memory of the CRAY and using highly efficient assembly coded level 3 BLAS kernels can be more efficient than corresponding level 2 BLAS implementations. This paper presents results of experiments performed on a CRAY Y-MP in a multiprocessor environment using parallel BLAS kernels.

2. Algorithm description

2.1. Block Gaussian elimination

The algorithm is described for the ABD system which arises when a two-point BVP is solved using a box-type difference scheme [7]. The size of each internal block is $n \times 2n$ and the BC-type blocks have sizes $q \times n$ and $(n-q) \times n$ respectively. (The BCs are assumed to be separated and q is the number of left BCs.) For systems which arise from collocation methods [2], the algorithm is similar. Figure 1 presents a typical ABD structure with five internal blocks (which corresponds to four internal meshpoints).

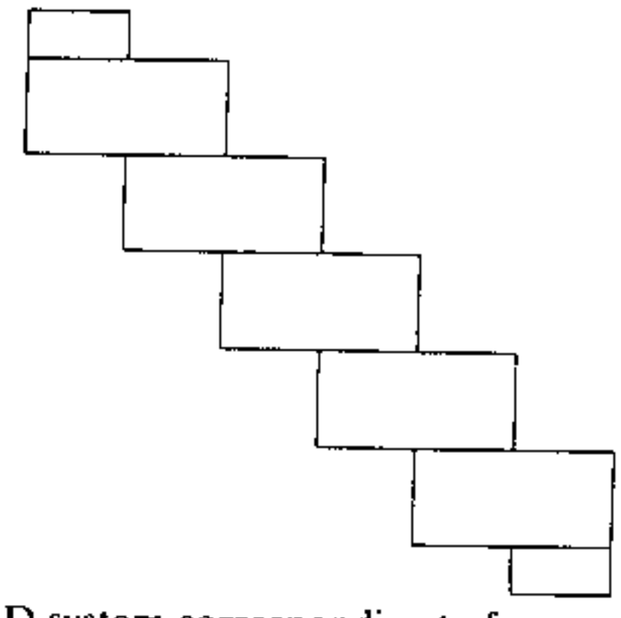

Fig. 1. ABD system corresponding to four meshpoints.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} = \begin{bmatrix} L_{1,1} & & \\ L_{2,1} & L_{2,2} & \\ L_{3,1} & L_{3,2} & L_{3,3} \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ & U_{2,2} & U_{2,3} \\ & & U_{3,3} \end{bmatrix} = LU,$$

$$LU = \begin{bmatrix} L_{1,1}U_{1,1} & L_{1,1}U_{1,2} & L_{1,1}U_{1,3} \\ L_{2,1}U_{1,1} & L_{2,1}U_{1,2} + L_{2,2}U_{2,2} & L_{2,1}U_{1,3} + L_{2,2}U_{2,3} \\ L_{3,1}U_{1,1} & L_{3,1}U_{1,2} + L_{3,2}U_{2,2} & L_{3,1}U_{1,3} + L_{3,2}U_{2,3} + L_{3,3}U_{3,3} \end{bmatrix}$$

Fig. 2. Block LU decompostion.

A short description of the algorithm is given. For further discussion see [14] (details of the implementation are presented in [12]). The algorithm calls the level 3 BLAS routines _TRSM (to solve a triangular system of equations with multiple right-hand sides $B = T^{-1}A$ or $B = AT^{-1}$) and _GEMM (to form a matrix-matrix product $C = \alpha AB + \beta C$). The decomposition step requires both row and column interchanges. Since the level 3 BLAS routine _GERTF [9] provides only row interchanges, it has been superseded. The sizes of the matrices to be decomposed in every step of the algorithm are rather small, so unblocked codes are used for the decomposition. Out of the three possible, column-oriented, versions of unblocked code [4], the SAXPY version was implemented. This version was shown to be most efficient in CRAY's parallel environment [11]. Parallelization is introduced inside the BLAS, its quality depends on the machine implementation.

In the algorithm an extended version of block LU decomposition, shown in Fig. 2, is utilized. Using this extended block structure, a two-phase per step algorithm for the decomposition of the ABD system is defined. At each step (except the last) Phase I decomposes a part of the matrix equivalent to a BC-type block, then Phase II decomposes the part of the internal block which occurs before the next BC-type block enters. Phase I uses column interchanges and decomposes a rectangular block of size $q \times n$. Phase II uses row interchanges and decomposes a rectangular block of size $n \times (n - q)$.

Phase I. Figure 3 shows the block placed inside the system for execution of Phase I. There is the following correspondence between Figs. 2 and 3:

(a) blocks E and F were calculated in previous steps;
(b) $D = [A_{1,1}\ A_{1,2}] \in \mathbb{R}^{q \times n}$;
(c) $A_{1,3} = 0 \in \mathbb{R}^{q \times q}$ lies outside the ABD structure;

(d) $$C = \begin{bmatrix} A_{2,1} \\ A_{3,1} \end{bmatrix} \in \mathbb{R}^{n \times q};$$

(e) $$U = \begin{bmatrix} A_{2,2} \\ A_{3,2} \end{bmatrix} \in \mathbb{R}^{n \times (n-q)};$$

(f) $$Z = \begin{bmatrix} A_{2,3} \\ A_{3,3} \end{bmatrix} \in \mathbb{R}^{n \times q}.$$

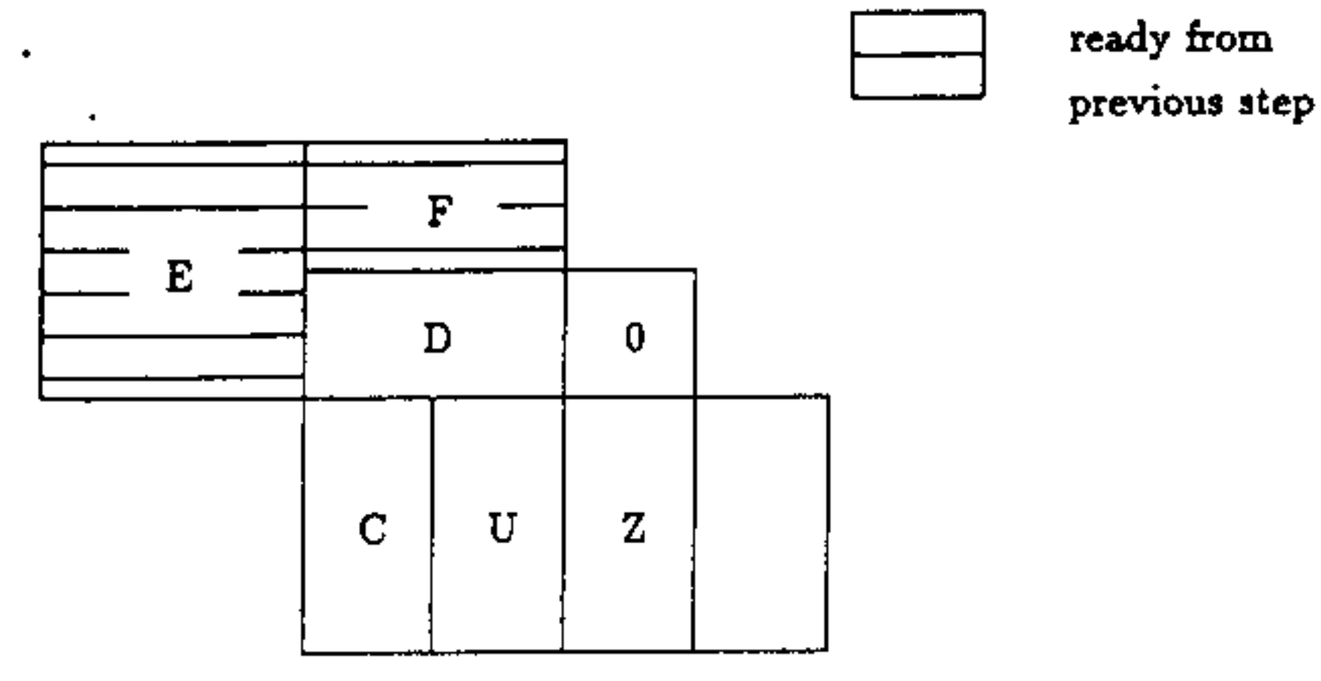

Fig. 3. Block structure in Phase I.

In Phase I,
(1) block D is decomposed; column interchanges are applied internally in D;
(2) these column interchanges are applied blockwise in C and U "below" and in F "above" D;
(3) values of C are calculated;
(4) block U is updated.

Phase II. Figure 4 presents the parts of the system already in place and the location of the block to be decomposed in Phase II. There is the following correspondence between Figs. 2 and 4:

(a) block E was calculated in Phase I;

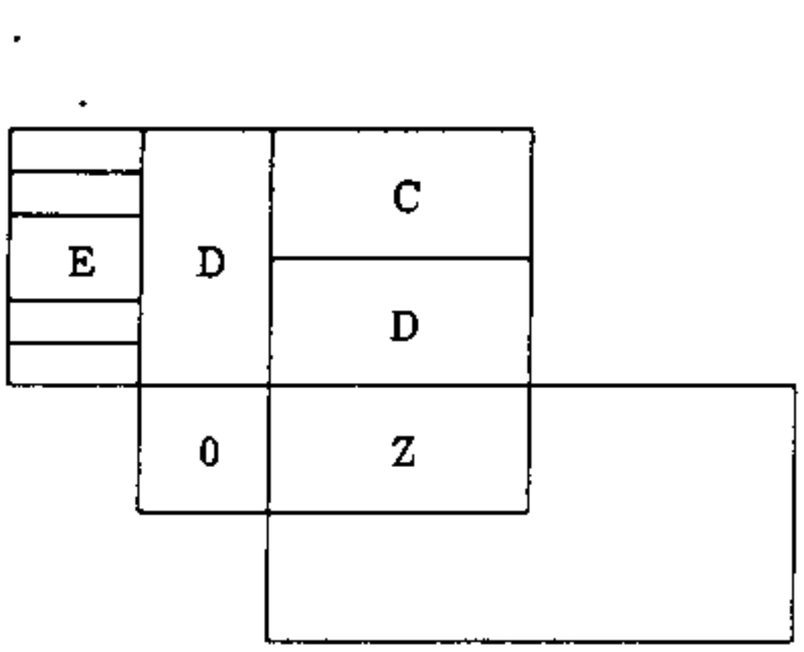

Fig. 4. Block structure in Phase II.

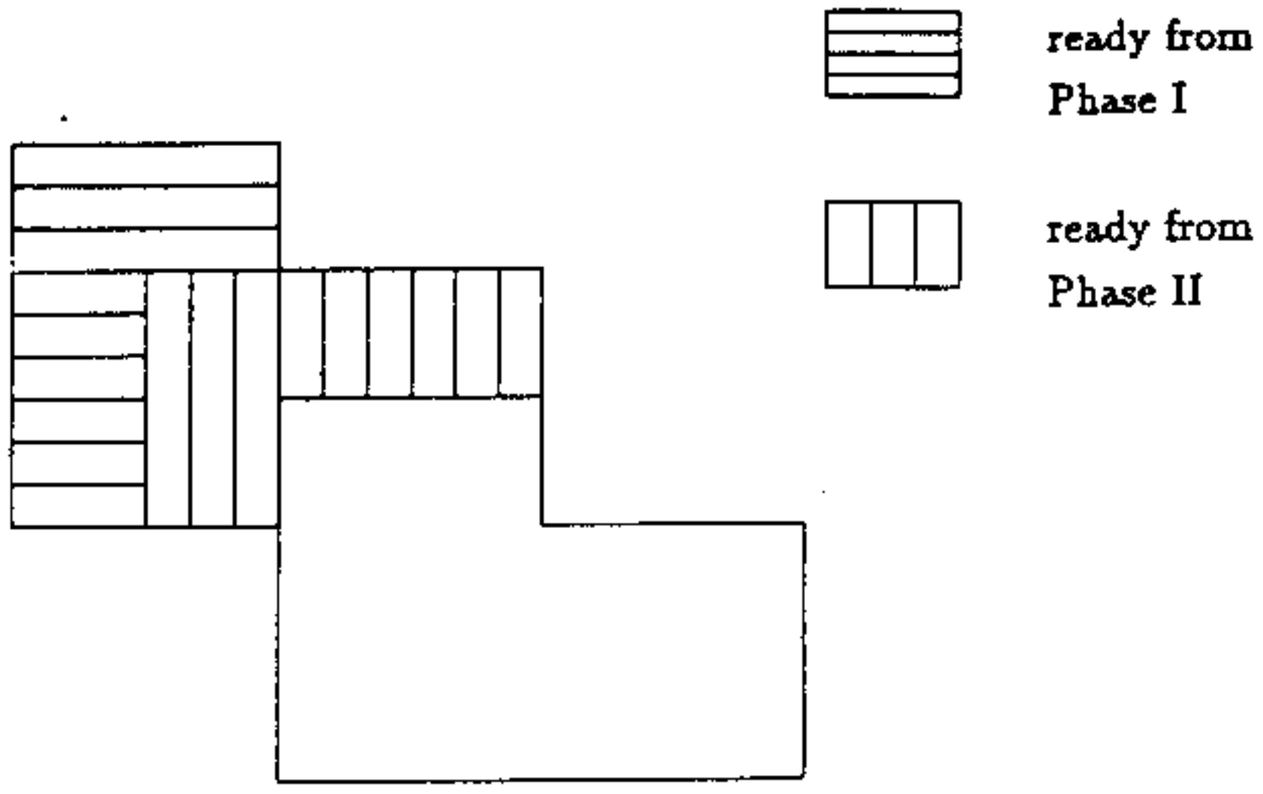

Fig. 5. System after one step of decomposition.

(b) $D = \begin{bmatrix} A_{1,1} \\ A_{2,2} \end{bmatrix} \in \mathbb{R}^{n \times (n-q)}$;

(c) $A_{3,1} = 0 \in \mathbb{R}^{q \times q}$ lies outside the ABD structure;
(d) $C = [A_{1,2}\ A_{1,3}] \in \mathbb{R}^{(n-q) \times n}$;
(e) $U = [A_{2,2}\ A_{2,3}] \in \mathbb{R}^{q \times n}$;
(f) $Z = [A_{3,2}\ A_{3,3}] \in \mathbb{R}^{(n-q) \times n}$.

In Phase II,

(1) block D is decomposed; row interchanges are applied internally in D;
(2) these row interchanges are applied blockwise in C and U on the "right" and in E on the "left";
(3) values of C are calculated;
(4) block U is updated.

This completes Phase II and one step of the algorithm. Figure 5 presents the system after one step of decomposition. Observe that the undecomposed (but updated) part of the ABD has precisely the same structure as the original system.

In the last step both phases are performed on a reduced block. A standard two-block decomposition is used.

2.2. *Block back substitution*

The decomposition described above leads to the system of the form

$$A = PLUQ,$$

where L is unit lower triangular, U is upper triangular and P and Q are row and column permutation matrices respectively. The solution of a system $Ax = b$ will be calculated by implementing the sequence of steps (a) $Pr = b$, (b) $Ls = r$, (c) $Ut = s$ and (d) $Qx = t$.

Steps (a) and (d) permute the right-hand side (RHS) and the final solution of the system respectively (using level 1 BLAS). Steps (b) and (c) represent forward and back substitution. The implementation of these operations uses level 3 BLAS routines. However, only for multiple right-hand sides is this a blocked algorithm. For a single right-hand side it is equivalent to using level 2 BLAS. Given this decomposition, one can also solve

$$A^{\mathrm{T}}x = b,$$

via

$$Q^{\mathrm{T}}U^{\mathrm{T}}L^{\mathrm{T}}P^{\mathrm{T}}x = b,$$

for any b using a similar sequence of steps. This enables estimation of the condition number of A using the algorithm of [6]. This is necessary adjunct since the code, like _GETRF, only exits with an error indication when exact (zero pivot) numerical singularity is encountered. The calculation using A and A^{T}, respectively, in the error estimation can, of course, be carried out in parallel.

3. Numerical results

Experiments were performed on an 8-processor CRAY Y-MP 8/128 at the CRAY Research Inc. facility in Eagan, in single-job stream mode. The system ran Unicos 6.1 and provided parallel assembly coded level 1, 2 and 3 BLAS kernels. Each result presented is an average of several runs. These results should be treated rather as predictors of a general tendency than as absolutely precise runtimes. Timings of particular runs varied from 2% in large cases up to 5% in small cases. The results are compared with those from the decomposition step from the current release (available on the CRAY in Eagan) of the NAG library (level 2 BLAS based) decomposition routine F01LHF [3,17]. The NAG library routine F04LHF (back substitution) does not parallelize, so there are no comparisons of back substitution to report.

Three series of experiments were run for $n = 50$, 200 and 400, respectively, and for $k = 25$. In the decomposition for $n = 50$ no speedup was observed. Figure 6 presents speedups obtained by the new code DGEABD and by F01LHF for $n = 200$ and for the number of "top" BCs ITBC = 100 and for $n = 400$ and ITBC = 200, respectively. Clearly for systems with internal block size 200 and a rather small number of such blocks (25) it is possible to obtain speedups of about 2.5. With internal block size 400 and the number of such blocks unchanged, the speedup increases to about 4.5. For small numbers of processors (below 4) the level 2 BLAS based code outperforms the level 3 BLAS based code. Such behavior can be observed more generally in the case of dense square linear systems, where for small system sizes and for small numbers of processors unblocked codes often outperform blocked ones [11].

Experiments with blocked code were performed in the decomposition step. In [14] it was reported that blocking can lead to performance improvement. However, here no improvement in performance was observed. This can be explained by the quality of the 2 BLAS kernels and by the rather small size of the matrices decomposed at every step.

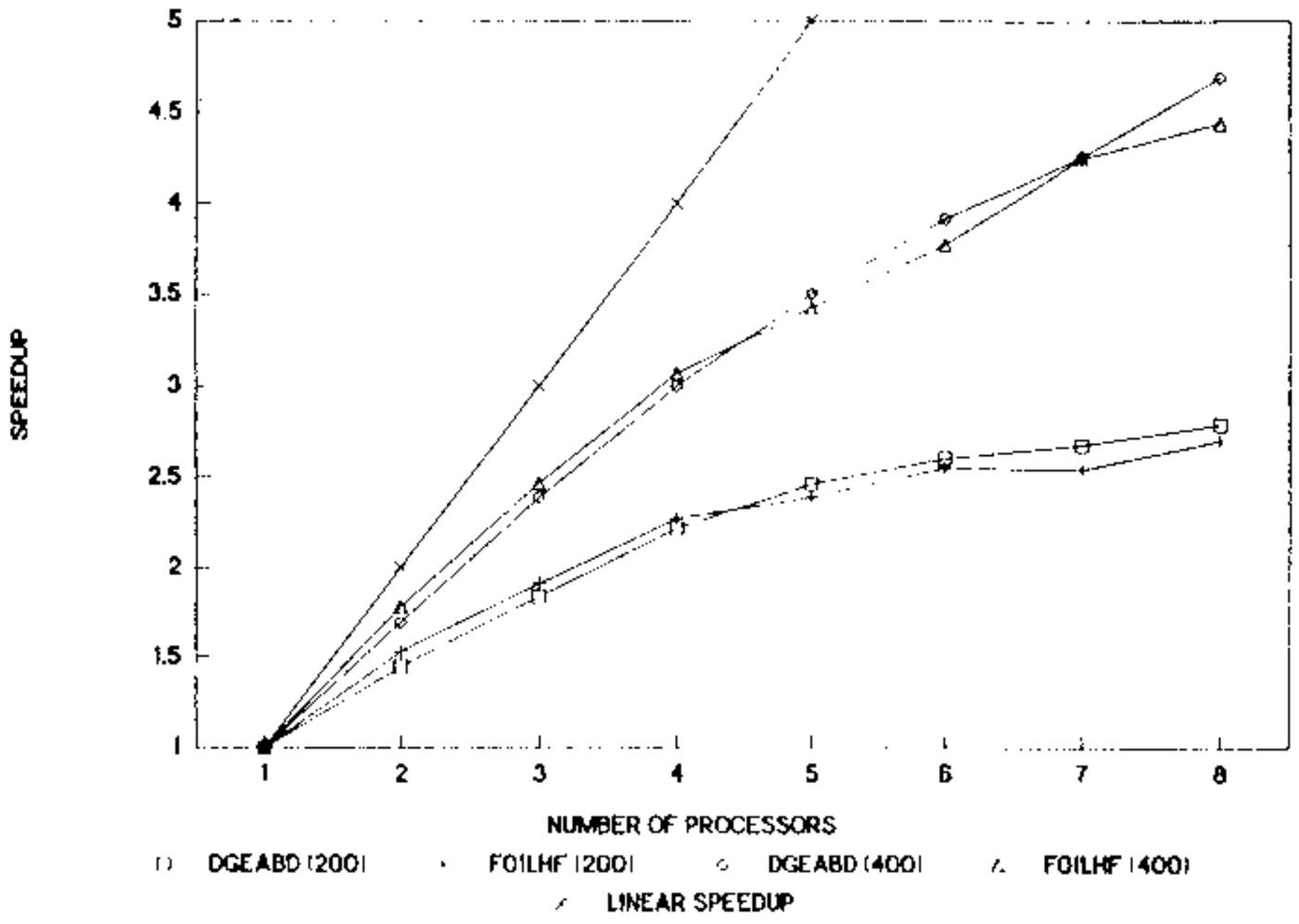

Fig. 6. Performance comparison for decomposition.

The number of "top" BCs was varied. Table 1 summarizes the timings obtained for eight processors when the number of "top" BCs (ITBC) varies between 1 and $n - 1$. In all cases but one, the level 3 BLAS based code outperforms the level 2 BLAS based one. More important, the 8-processor performance improvement is greater for larger internal block sizes.

Another series of experiments were performed to establish the possibility of parallelizing the back substitution step in both standard and transposed modes. For one right-hand side and for all cases ($n = 50$, 200, 400) increasing the number of processors from one to two cut the time by about half, but a further increase in processors did not improve performance. The situation was different for multiple right-hand sides. Figure 7 summarizes the speedups obtained for the equally split boundary conditions; in each case twenty right-hand sides were used and (T) specifies "transposed" back substitution. Results for other values of ITBC were similar.

Even for $n = 50$ a speedup of about 2.5 is observed. Also, for only twenty RHSs the speedup obtained for back substitution is greater than that for decomposition. Note the effect of different divisions of the workload for larger numbers of processors. It is clear that the speedup obtained for seven processors is relatively much greater than for six and eight processors. This

Table 1
Timing variations with size of top block, ITBC

	ITBC	1	34	67	100	133	166	199
$n = 200$	DGE	0.456	0.462	0.503	0.495	0.494	0.476	0.456
	F01	0.447	0.482	0.528	0.551	0.565	0.577	0.588
	ITBC	1	68	134	200	266	332	399
$n = 400$	DGE	2.104	2.173	2.232	2.255	2.234	2.270	2.221
	F01	2.170	2.281	2.351	2.425	2.410	2.345	2.287

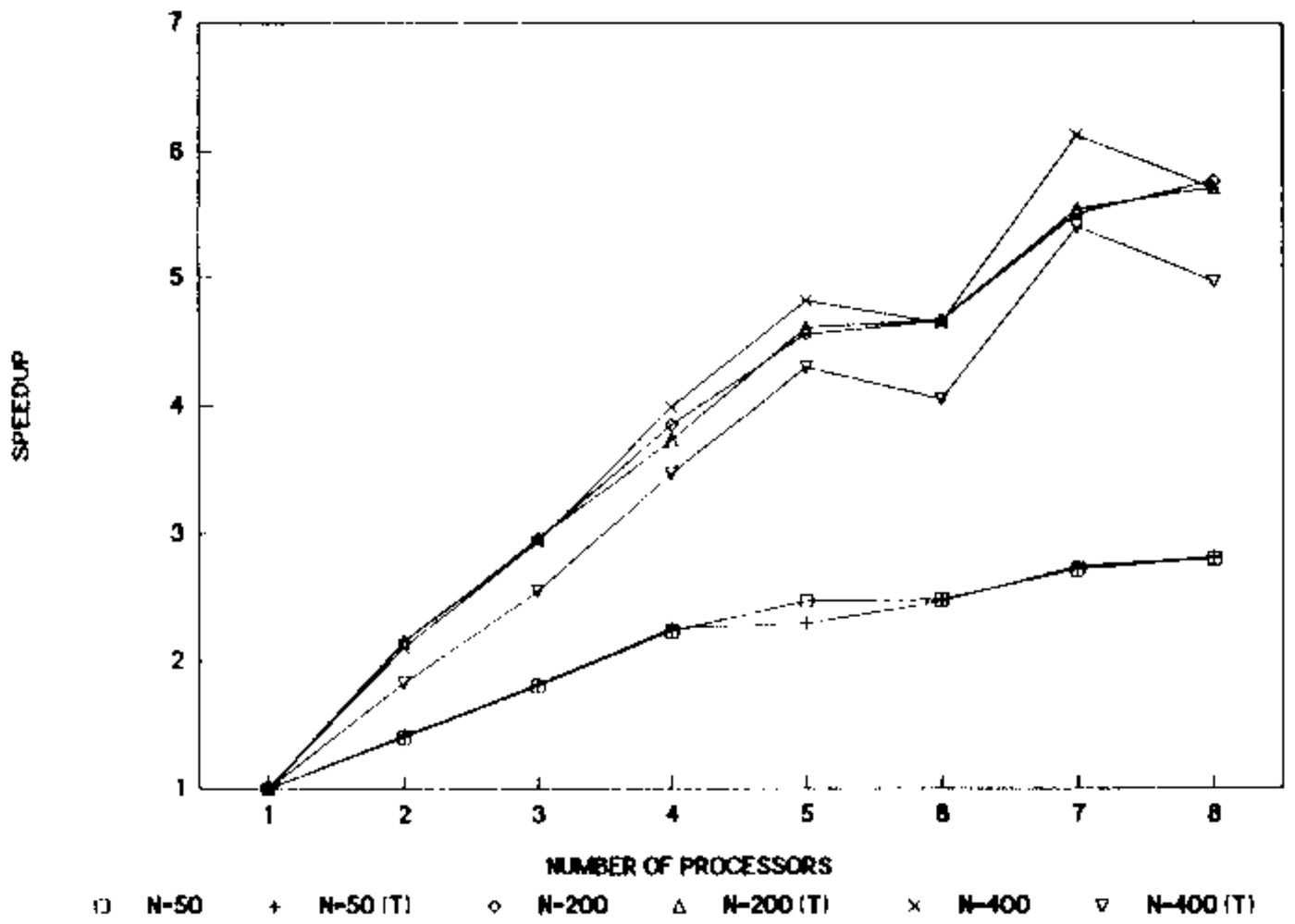

Fig. 7. Speedups for multiple right-hand side back substitution.

can be explained by the CRAY memory organisation's sensitivity to vectors of length divisible by powers of two. Here $n = 400$; hence the vector length is divisible by sixteen which may cause memory conflicts. This may also explain the superlinear speedup observed for two processors. Splitting the work equally can reduce memory conflicts and cause an additional performance increase. This was confirmed by running the code with $n = 401$ where no superlinear speedup was observed.

A suite of codes 3ABDSOL [11] realizing the algorithm described above is available from the second author. It has the functionality of the level 2 BLAS ABD code described in [3] and implemented in the NAG Library [17].

4. Conclusions

A parallel algorithm for almost block diagonal systems with large internal blocks has been developed. This algorithm is based on block linear algebra and is implemented using level 1, 2 and 3 BLAS kernels providing internal parallelization. It maximizes exploitation of the underlying structure of the system and has the same stability properties as other Gaussian elimination algorithms with row and column interchanges for ABDs. The results presented suggest that the proposed algorithm should be a standard solution for problems with very large block sizes.

References

[1] U.M. Ascher and S.Y.P. Chan, On parallel methods for boundary value ODE's, *Computing* **46** (1991) 1–17.
[2] U. Ascher, J. Christiansen and R.D. Russell, Collocation software for boundary value ODEs, *ACM Trans. Math. Software* **7** (1981) 209–229.

[3] R.W. Brankin and I. Gladwell, Codes for almost block diagonal systems, *Comput. Math. Appl.* **19** (1990) 1–6.
[4] M.J. Dayde and I.S. Duff, Level 3 BLAS in LU factorization on Cray 2, ETA-10P and IBM 3090-200/VF, *Internat. J. Supercomput. Appl.* **3** (1989) 40–70.
[5] J. Dongarra, J.J. Du Croz, I. Duff and S. Hammarling, A set of level 3 basic linear algebra subprograms, Report ANL-MCS-P88-1, Argonne National Laboratory, 1988.
[6] N.J. Higham, FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation, *ACM Trans. Math. Software* **14** (1988) 381–396.
[7] H.B. Keller, *Numerical Solution of Two Point Boundary Value Problems* (SIAM, Philadelphia, PA, (1976).
[8] M. Lentini and V. Pereyra, An adaptive finite-difference solver for nonlinear two-point boundary problems with mild boundary layers, *SIAM J. Numer. Anal.* **14** (1977) 91–111.
[9] P. Mayes and G. Radicati, Banded Cholesky factorization using level 3 BLAS, LAPACK working note #12, Report ANL-MCS-TM-134, Argonne National Laboratory, 1988.
[10] M. Paprzycki, Solving dense linear systems of Cray Y-MP multiprocessor using assembly coded BLAS kernels, in preparation.
[11] M. Paprzycki and C. Cyphers, 3ABDSOL — a level 3 BLAS based solver for almost block diagonal systems, SMU Math Softreport 92-3, Dept. Math., Southern Methodist Univ., Dallas, TX, 1992.
[12] M. Paprzycki and I. Gladwell, Solving almost block diagonal systems using level 3 BLAS, SMU Math Report 90-4, Dept. Math., Southern Methodist Univ., Dallas, TX, 1990.
[13] M. Paprzycki and I. Gladwell, Solving almost block diagonal systems on parallel computers, *Parallel Comput.* **17** (2&3) (1991) 133–153.
[14] M. Paprzycki and I. Gladwell, Using level 3 BLAS to solve almost block diagonal systems, in: D. Sorensen et al., Eds., *Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing* (SIAM, Philadelphia, PA, 1992) 52–62.
[15] J.M. Varah, Alternate row and column elimination for solving certain linear systems, *SIAM J. Numer. Anal.* **13** (1976) 71–75.
[16] S.J. Wright, Stable parallel elimination for boundary value ODE's, preprint, ANL-MCS-P229-0491, Argonne National Laboratory, 1991.
[17] The NAG Library Manual, Mark 15, 1991, NAG Ltd., Oxford.