Control and Cybernetics

vol. **25** (1996) No. 3

On the parallel solution of almost block diagonal systems

by

Pierluigi Amodio* and Marcin Paprzycki**

*Dipartimento di Matematica, Università di Bari, Via Orabona 4, I-70125 Bari, Italy, e-mail: `na.pamodio@na_net.ovnl.gov`

**Department of Mathematics, University of Texas of the Permian Basin, Odessa, TX 79762, USA, e-mail: `paprzycki_m@utpb.edu`

Abstract. In this paper we briefly summarize some of the most important algorithms for the parallel solution of Almost Block Diagonal linear systems. Then, we present a parallel algorithm based on the cyclic reduction, which is quite competitive, especially when systems with additional corner blocks are considered. Numerical tests carried out on a distributed memory parallel computer are reported and analyzed.

Keywords: almost block diagonal systems, parallel computers

1. Introduction

Since the 1970's a number of publications studying solution methods for Almost Block Diagonal (ABD) linear systems appeared. These systems arise in various mathematical applications such as Chebyshev spectral decomposition on rectangular domains, orthogonal spline collocation for elliptic problems and various discretizations of boundary value ordinary differential equations (BVP ODE's). We are primarily interested in the latter application. The sequential solution methods can be traced back first to the SOLVEBLOCK package by de Boor and Weiss (1980), and, second, to the alternate row and column elimination algorithm due to Varah (1976), later studied by Diaz, Fairweather and Keast (1983) and implemented using level 3 BLAS primitives by Paprzycki and Gladwell (1991).

There exist a number of approaches to the parallel solution of ABD systems. It was observed that there are two basic parameters that influence the possible solution methods. When the size of each block is large (as in the case of spectral decompositions, generating a relatively small number of large blocks) the level 3 BLAS based approach can be applied (parallelism is introduced inside the BLAS kernels). Gladwell and Paprzycki (1993) have experimented with this approach on shared memory computers and reported satisfactory performance. When a number of blocks is large and their individual sizes are small, tearing-type

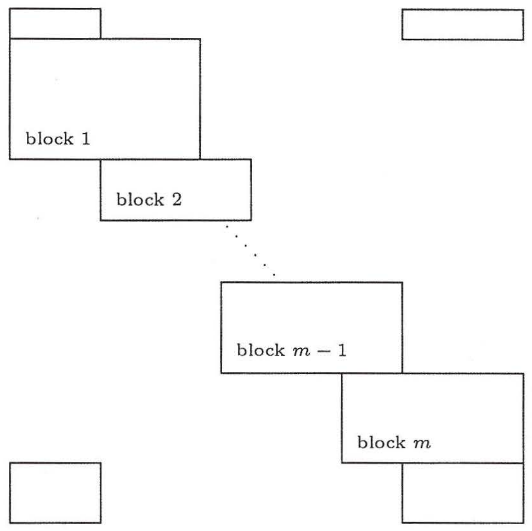

Figure 1. Structure of an ABD matrix with additional corner blocks.

methods can be applied. Ascher and Chan (1991) and Jackson and Pancer (1991) have formed normal equations and suggested applying tearing to the resulting block tridiagonal system. A different tearing-type algorithm has been proposed by Paprzycki and Gladwell (1991), where the tearing process is applied to the ABD system; this approach has been later improved by Amodio and Paprzycki (1996). Yet another tearing method was proposed by K. Wright (1993). He applies tearing to the block bidiagonal system obtained by ignoring the boundary condition blocks, and reintroduces these blocks only in the final step of the solution process. While the method proposed in Paprzycki and Gladwell (1991) and in Amodio and Paprzycki (1996) can be applied on the message passing as well as shared memory computers, the method presented in K. Wright (1993) can be used only on shared memory computers.

All these algorithms deal with the solution of the ABD system arising from the discretization of BVP ODE's with *separated* boundary conditions. In case of *non-separated* boundary conditions additional corner blocks arise (see Figure 1) and no known sequential algorithm exists. S. Wright (1992, 1994) introduced two parallel methods (similar to the approach of K. Wright) that can deal with these additional blocks and can be used on shared memory as well as message passing parallel computers. The aim of this paper is to present a different, cyclic-reduction based approach to the solution of ABD systems with separated as well as non-separated boundary conditions. In Sections 2–3, the proposed algorithm is summarized (in Section 3 a slightly modified version is proposed) while in Section 4, the results of numerical experiments are presented and discussed.

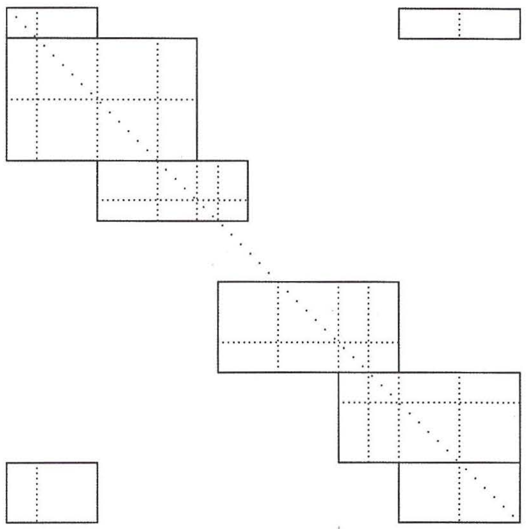

Figure 2. Structure of an ABD matrix with additional corner blocks after the decomposition.

2. The cyclic reduction approach

The cyclic reduction algorithm is one of the most interesting algorithms for the solution of tridiagonal and block tridiagonal linear systems on parallel computers (see, for example, Amodio et al. (1993)). Several implementations have been proposed and have been used to optimize the solution process on different computer architectures. To derive a generalization of the cyclic reduction algorithm for the factorization of ABD matrices let us represent the ABD matrix M in the following form (see Figure 2 and compare with Figure 1):

$$M = \begin{pmatrix} A_{2,0} & D_{2,0} & & & & & & & C_{1,0} & B_{1,0} \\ D_{1,1} & A_{1,1} & B_{2,1} & C_{2,1} & & & & & & \\ C_{1,1} & B_{1,1} & A_{2,1} & D_{2,1} & & & & & & \\ & & D_{1,2} & A_{1,2} & B_{2,2} & C_{2,2} & & & & \\ & & C_{1,2} & B_{1,2} & A_{2,2} & D_{2,2} & & & & \\ & & & & & \ddots & & & & \\ & & & & & D_{1,m} & A_{1,m} & B_{2,m} & C_{2,m} \\ & & & & & C_{1,m} & B_{1,m} & A_{2,m} & D_{2,m} \\ B_{2,m+1} & C_{2,m+1} & & & & & & D_{1,m+1} & A_{1,m+1} \end{pmatrix} \qquad (1)$$

where blocks $A_{i,j}$ are square and any generic block j from Figure 1 (except the corner blocks) is expressed by the following 2×4 block matrix:

$$\begin{pmatrix} D_{1,j} & A_{1,j} & B_{2,j} & C_{2,j} \\ C_{1,j} & B_{1,j} & A_{2,j} & D_{2,j} \end{pmatrix} \qquad (2)$$

Notice that blocks in (1) may have different sizes. At this time the only requirement is that blocks $A_{i,j}$ be square and the size of the other blocks $B_{i,j}, C_{i,j}$ and $D_{i,j}$ be set accordingly (so, for example, $B_{2,i}, D_{1,i+1}$ and $C_{1,i+1}$ have the same number of columns of $A_{2,i}$ and $C_{2,i-1}, D_{2,i-1}$ and $B_{1,i}$ have the same number of columns of $A_{1,i}$). Moreover each block may have some null row or column, or even zero size. For example, if a BVP ODE with separated boundary conditions is being solved, the size of blocks $A_{2,i}$ is typically chosen equal to the number of initial conditions and the size of blocks $A_{1,i}$ equal to the number of final conditions.

To emphasize a block tridiagonal structure, let us now rewrite (1) as:

$$\begin{pmatrix} \Delta_0 & \Lambda_0 & & \Gamma_0 \\ \Gamma_1 & \Delta_1 & \ddots & \\ & \ddots & \ddots & \Lambda_{m-1} \\ \Lambda_m & & \Gamma_m & \Delta_m \end{pmatrix}, \tag{3}$$

where

$$\Gamma_i = \begin{pmatrix} C_{1,i} & B_{1,i} \\ O & O \end{pmatrix}, \quad \Delta_i = \begin{pmatrix} A_{2,i} & D_{2,i} \\ D_{1,i+1} & A_{1,i+1} \end{pmatrix}, \quad \Lambda_i = \begin{pmatrix} O & O \\ B_{2,i+1} & C_{2,i+1} \end{pmatrix}. \tag{4}$$

We may now apply the odd-even cyclic reduction (similar to that proposed in Amodio and Mazzia (1994)) to the matrix (3). In order to preserve the sparsity structure it is required that the first and the last row of (3) are treated as even rows (the first row is considered as row 0, and m must be even). The first step of the algorithm reduces matrix (3) to the following one (which has half the number of blocks of the original system):

$$\begin{pmatrix} \tilde\Delta_0 & \tilde\Lambda_0 & & & \Gamma_0 \\ \tilde\Gamma_2 & \tilde\Delta_2 & \tilde\Lambda_2 & & \\ & \tilde\Gamma_4 & \tilde\Delta_4 & \ddots & \\ & & \ddots & \ddots & \tilde\Lambda_{m-2} \\ \Lambda_m & & & \tilde\Gamma_m & \tilde\Delta_m \end{pmatrix}, \tag{5}$$

where

$$\begin{aligned} &\tilde\Delta_0 = \Delta_0 - \Lambda_0\Delta_1^{-1}\Gamma_1, \qquad \tilde\Delta_m = \Delta_m - \Gamma_m\Delta_{m-1}^{-1}\Lambda_{m-1}, \\ &\tilde\Delta_{2i} = \Delta_{2i} - \Lambda_{2i}\Delta_{2i+1}^{-1}\Gamma_{2i+1} - \Gamma_{2i}\Delta_{2i-1}^{-1}\Lambda_{2i-1}, \quad \text{for } i = 1,\dots,m/2-1, \\ &\tilde\Gamma_{2i} = -\Gamma_{2i}\Delta_{2i-1}^{-1}\Gamma_{2i-1}, \qquad \text{for } i = 1,\dots,m/2, \\ &\tilde\Lambda_{2i} = -\Lambda_{2i}\Delta_{2i+1}^{-1}\Lambda_{2i+1}, \qquad \text{for } i = 0,\dots,m/2-1. \end{aligned} \tag{6}$$

Note that during the factorization of blocks Δ_i row pivoting can be applied. Blocks Λ_i and Γ_i have some null rows, and thus $\Lambda_{2i}\Delta_{2i+1}^{-1}$ and $\Gamma_{2i}\Delta_{2i-1}^{-1}$ have zeroes in the same rows and therefore blocks $\tilde{\Gamma}_{2i}$ and $\tilde{\Lambda}_{2i}$ maintain the same sparsity structure as the corresponding Γ_{2i} and Λ_{2i}. Moreover, observe that in the matrix (5) the blocks Λ_m and Γ_0 as well as rows corresponding to the first and the last row of (1) remain unchanged. This means that the corner blocks (the blocks containing the boundary conditions, if a BVP ODE is solved) are not changed during the reduction process.

The same approach is being repeated and applied to (5) and after $\log_2(m)$ steps (assuming m is a power of 2) a 2×2 block matrix (or a 4×4 block matrix, if expressed in terms of $A_{j,i}$, $B_{j,i}$, $C_{j,i}$ and $D_{j,i}$) is obtained and factorized using Gaussian Elimination with partial pivoting.

3. The stabilized cyclic reduction approach

Since blocks Δ_i in (3) may be ill conditioned or even singular, the algorithm proposed above may be unstable. Moreover, even if for a given ABD matrix we choose the initial decomposition (1) such that each block on the main diagonal of the original matrix (3) is non-singular, it is quite difficult to prove that blocks obtained in the cyclic reduction process remain non-singular. To overcome this problem, we can modify the previous algorithm slightly and thereby ensure stability.

Consider the first step of reduction and let n_i be the number of columns of

$$\begin{pmatrix} B_{2,i} & C_{2,i} \\ A_{2,i} & D_{2,i} \\ D_{1,i+1} & A_{1,i+1} \\ C_{1,i+1} & B_{1,i+1} \end{pmatrix}. \tag{7}$$

This means that block Δ_i in (3) is $n_i \times n_i$. Observe that since the original matrix (1) is nonsingular, it is possible to extract from (7) a nonsingular $n_i \times n_i$ matrix. Then by applying row permutations inside blocks i and $i+1$ (see (1) and (2)), we may derive a new decomposition

$$\begin{pmatrix} \hat{B}_{2,i} & \hat{C}_{2,i} \\ \hat{A}_{2,i} & \hat{D}_{2,i} \\ \hat{D}_{1,i+1} & \hat{A}_{1,i+1} \\ \hat{C}_{1,i+1} & \hat{B}_{1,i+1} \end{pmatrix} \tag{8}$$

where block $\hat{\Delta}_i$ (defined as Δ_i but with blocks $\hat{A}_{2,i}$, $\hat{D}_{2,i}$, $\hat{D}_{1,i+1}$ and $\hat{A}_{1,i+1}$ in (8)) is nonsingular. This can be achieved by applying Gaussian Elimination with partial pivoting to block (7) and then row permutations in order to insert the pivotal elements in the rows of $\hat{A}_{2,i}$ and $\hat{A}_{1,i+1}$. Obviously with this approach blocks $\hat{B}_{2,i}, \hat{C}_{2,i}, \ldots$ have no longer the same size of $B_{2,i}, C_{2,i}, \ldots$, but $\hat{\Delta}_i$ is still $n_i \times n_i$.

To better understand the difference between the two algorithms observe now what happens if they are applied to an ABD matrix with $D_{1,j} = B_{1,j} = I_n$ and $A_{1,j} = C_{1,j} = A_{2,j} = B_{2,j} = C_{2,j} = D_{2,j} = O_n$. This matrix is obviously nonsingular (it is a permutation of the identity matrix) and may also be derived from a simple linear BVP ODE (with $M = \lambda I_{2n}$ in (9) and by selecting an appropriate value of the stepsize h, and an appropriate number of initial conditions).

If we use the cyclic reduction, then $\Delta_i = \begin{pmatrix} O_n & O_n \\ I_n & O_n \end{pmatrix}$ is singular and the algorithm stops at the first step of reduction.

If we use the stabilized algorithm, then Gaussian Elimination with partial pivoting is applied to blocks of the form $\begin{pmatrix} O_{2n} \\ I_{2n} \end{pmatrix}$. The permutations will result in $\hat{A}_{2,i}$, $\hat{D}_{2,i}$, $\hat{C}_{1,i+1}$, and $\hat{B}_{1,i+1}$ having zero rows, $\hat{A}_{1,i+1}$ and $\hat{C}_{2,i}$ being $2n \times 2n$ and $\hat{D}_{1,i+1}$ and $\hat{B}_{2,i}$ being $2n \times 0$. Then $\hat{\Delta}_i = I_{2n}$ is nonsingular and the algorithm calculates the solution.

Summarizing, the main (substantial) difference between the two algorithms is that in cyclic reduction the size of the blocks $A_{i,j}, B_{i,j}, \ldots$, is a priori fixed while in the stabilized cyclic reduction it is variable (it changes at each step of reduction) and depends on the factorization process (the pivoting elements during the factorization). The main advantages of the stabilized algorithm is that the factorization always exists and is stable. Moreover, the computational cost is the same as that of the cyclic reduction algorithm of the previous section, even if the execution time will be greater because additional permutations are performed. At the same time our numerical experiments suggest that the possibility for an ABD matrix to have a singular block Δ_i is very small (see below and Amodio and Paprzycki 1995) so the stabilized algorithm needs to be used only if the standard algorithm fails.

4. Numerical tests

Numerical tests were performed on a distributed memory parallel computer MicroWay Multiputer with 32 processors. Each processor is a T800 transputer and has a local memory of 1 Mbyte. The sequential tests were performed on a single transputer with 16 Mbytes of local memory. The implementation details can be found in Amodio and Paprzycki (1995). We have tested the two sequential algorithms:

- SOLVEBLOCK routine by De Boor and Weiss (1980);
- SGEABD routine by Cyphers, Paprzycki and Gladwell (1992);

and two parallel algorithms introduced in the previous sections:

- ABDCR cyclic reduction;

Table 1. Sequential execution times for the Problems 1–5 and $m = 32$

	Probl. 1	Probl. 2	Probl. 3	Probl. 4	Probl. 5
SOLVEBLOCK	399	722	2270	7819	13912
SGEABD	828	1267	2749	6557	11016
ABDCR	775	1495	4031	11453	19537
SABDCR	963	1904	4585	12857	21833

- SABDCR stabilized version of the cyclic reduction.

All the considered test problems arise from the numerical solution (by using finite differences) of a general boundary value problem:

$$\begin{aligned} &y' = My + q(t) \qquad t \in [a,b], \qquad y, q \in \Re^n, M \in \Re^{n\times n} \\ &B_a y(a) + B_b y(b) = d \in \Re^n. \end{aligned} \tag{9}$$

where M is a random $n \times n$ matrix, for $n = 2, 3, 5, 8, 10$. The following five test problems have been experimented with:

- Problem 1: system of two first-order BVP's – internal blocks of size 2×4;
- Problem 2: system of three first-order BVP's – internal blocks of size 3×6;
- Problem 3: system of five first-order BVP's – internal blocks of size 5×10;
- Problem 4: system of eight first-order BVP's – internal blocks of size 8×16;
- Problem 5: system of ten first-order BVP's – internal blocks of size 10×20.

In all cases $q(t)$ has been selected in such a way to have all the components of the solution behave as e^t. In order to compare the performance of the proposed algorithms with that of SOLVEBLOCK and SGEABD, separated boundary conditions have been selected (both these codes are designed for ABD matrices without additional corner blocks). Then, in a separate experiment, we have applied the new algorithms to the same BVP's with non-separated boundary conditions. The timings of the parallel solutions of problems with separated as well as non-separated BC's were exactly the same, in agreement with the arithmetical complexity functions of the proposed algorithms derived in Amodio and Paprzycki (1995).

Tables 1–2 contain the sequential execution times in ticks (1 tick = $64 \cdot 10^{-6}$ seconds) of the proposed solvers for $m = 32$ and 256 internal blocks. It can be observed that for both values of m SOLVEBLOCK outperforms the remaining solvers for the first three problems while time ratio between SOLVEBLOCK and SGEABD decreases as the size of the internal blocks increases. For Problems 4 and 5 the situation reverses and SGEABD outperforms SOLVEBLOCK. This can be explained by the fact that the BLAS kernels used were not specially tuned for the Transputers. In such a case, for small blocks, level 1 and 2 BLAS based SOLVEBLOCK can outperform level 3 BLAS based SGEABD. Interestingly,

Table 2. Sequential execution times for the Problems 1–5 and $m = 256$

	Probl. 1	Probl. 2	Probl. 3	Probl. 4	Probl. 5
SOLVEBLOCK	3142	5699	17977	62125	110497
SGEABD	6456	9944	21617	51915	87639
ABDCR	6124	11811	31709	91170	155211
SABDCR	7634	15184	36416	103559	174586

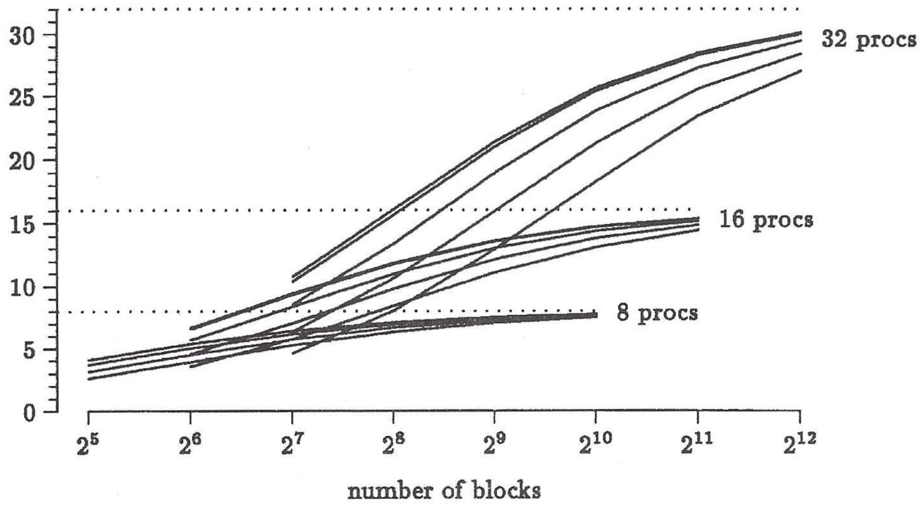

Figure 3. Speedup of the ABDCR solver.

the ABDCR routine outperforms SGEABD for the Problem 1, but as the sizes of the individual blocks increase SGEABD becomes faster. The time ratio of ABDCR to SOLVEBLOCK decreases for larger block sizes. This shows once more the advantage of level 3 BLAS kernels which were used to implement ABDCR. In all cases the stabilized SABDCR routine is the slowest.

Figure 3 represents the speedup (ratio between the sequential and the parallel execution time) of the ABDCR algorithm for $p = 8, 16$ and 32 processors for the increasing value of m. For each number of processors the five lines represent speedup of the five test problems. Figure 4 contains similar results for the stabilized algorithm SABDCR. In both cases the best speedups were obtained for the Problem 5, the worst for Problem 1. It should be observed that when compared internally both algorithms perform well. For large n an almost linear speedup is observed.

Tables 3 and 4 address the scalability of the proposed algorithms. The execution times of ABDCR and SABDCR for all three problems are presented

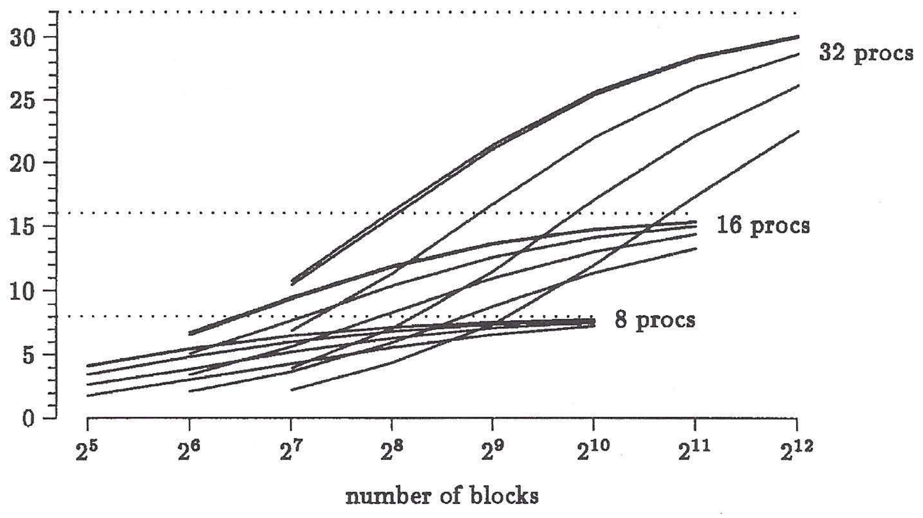

Figure 4. Speedup of the SABDCR solver.

Table 3. Scalability of the ABDCR solver ($m = 32p$).

	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$
Problem 1	775	824	883	967	1104	1338
Problem 2	1495	1567	1654	1768	1946	2218
Problem 3	4031	4173	4345	4560	4864	5319
Problem 4	11453	11848	12281	12787	13443	14348
Problem 5	19537	20210	20934	21779	22782	24181

in ticks for $p = 1, 2, 4, 8, 16$ and 32 processors while the size of the problem (number of blocks m) increases as the number of processors increases and is equal to $32 * p$ (32 blocks per processor).

It can be observed that the algorithms do not scale too well. The best scalability (for both algorithms) has been observed for Problem 5. This result should be viewed together with the fact that the best speedup has been also observed for Problem 5. It can be explained by the fact that Problem 5 is characterized by the largest sizes of the blocks thus leading to the best calculation-to-communication ratio. The ABDCR algorithm is not only faster and has better speedup, but has also better scalability than its stabilized version.

Figure 5 presents time ratio of the best sequential algorithm for a given problem (SOLVEBLOCK for Problems 1–3 and SGEABD for Problems 4–5) to the faster of the parallel algorithms (ABDCR) for the increasing number of

Table 4. Scalability of the SABDCR solver ($m = 32p$).

	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Problem 1	963	1042	1152	1373	1731	2547
Problem 2	1904	2016	2167	2410	2771	3557
Problem 3	4585	4771	4992	5330	5783	6613
Problem 4	12857	13206	13871	14449	15244	16292
Problem 5	21833	22563	233745	24351	25558	27205

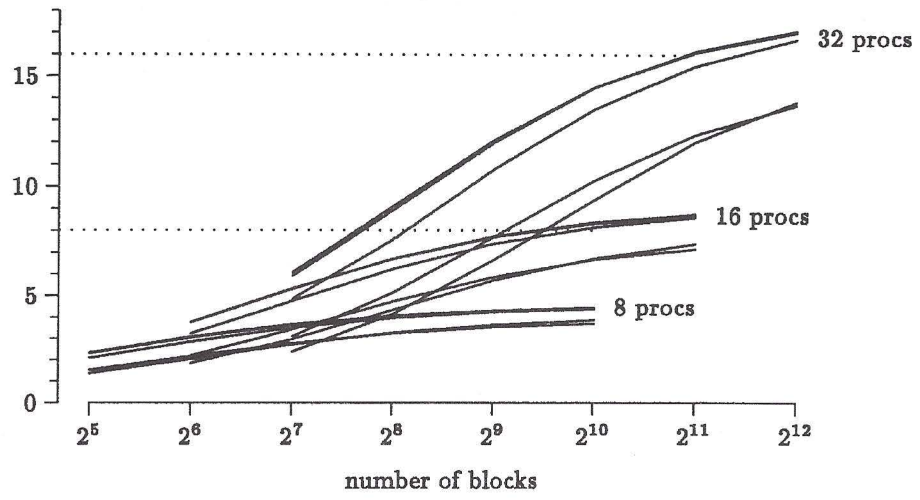

Figure 5. Time ratio of the best considered scalar solver (SOLVEBLOCK for the first three problems and SGEABD for the remaining two) and the ABDCR solver.

blocks m. This data represents the absolute speedup of the proposed algorithm. It can be observed that this time the performance gain is not as large as shown in Figures 3–4. As previously, Problem 5 is characterized by the best speedup and for large m reaches efficiency of 53%.

Finally, it should be pointed out that, for all problems reported here, as well for a number of additional tests that we have performed (some results have been reported in Amodio and Paprzycki 1995), the relative error with respect to the exact solution obtained by both ABDCR and SABDCR was the same and equal to the error given by the sequential algorithms.

5. Conclusion

Two versions of cyclic reduction algorithm for the solution of Almost Block Diagonal Systems have been presented. The experimental results suggest that the standard cyclic reduction algorithm has good numerical properties and performs well on a distributed memory computer. The stabilized algorithm does not perform as well and its usage should be limited to the cases when it is known that the problem may have a badly conditioned matrix (or when the non-stabilized algorithm fails).

Acknowledgements

The authors would like to express their gratitude to the anonymous referee for the helpful comments that improved the paper, and to Katarzyna Paprzycka for help with English.

References

Amodio, P., Brugnano, L. and Politi, T. (1993) Parallel factorizations for tridiagonal matrices. *SIAM J. Numer. Anal.* **30**, 813–823.

Amodio, P. and Mazzia, F. (1994) Backward error analysis of cyclic reduction for the solution of tridiagonal systems. *Math. Comp.* **62**, 601–617.

Amodio, P. and Paprzycki, M. (1995) A cyclic reduction approach to the numerical solution of boundary value ODE's. Report n. 19/95 of the Dipartimento di Matematica, Università di Bari, Bari, Italy.

Amodio, P. and Paprzycki, M. (1996) Parallel solution of almost block diagonal systems on a hypercube. *Linear Algebra Applic.* (in press).

Ascher, U.M. and Chan, S.Y.P. (1991) On the parallel methods for boundary value ODE's. *Computing* **46**, 1–17.

Cyphers, C., Paprzycki, M. and Gladwell, I. (1992) A level 3 BLAS based solver for almost block diagonal systems. SMU Software Report, 92-3.

de Boor, C. and Weiss, R. (1980) SOLVEBLOCK: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Software* **6**, 80–87.

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

Gladwell, I. and Paprzycki, M. (1993) Parallel solution of almost block diagonal systems on the CRAY Y-MP using level 3 BLAS. *J. Comput. Appl. Math.* **45**, 181–189.

Jackson, K.R. and Pancer, R.N. (1991) The parallel solution of ABD systems arising in numerical methods for BVP's for ODE's. Technical report

n. 255/91 of the Department of Computer Science, University of Toronto, Toronto, Ontario, Cananda.

KREISS, H.O., NICHOLS, K. AND BROWN, D. (1986) Numerical methods for stiff two-point boundary value problems. *SIAM J. Numer. Anal.* **23**, 325–368.

PAPRZYCKI, M. AND GLADWELL, I. (1991) Solving almost block diagonal systems on parallel computers. *Parallel Comput.* **17**, 133–153.

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

WRIGHT, K. (1993) Parallel treatment of block-bidiagonal matrices in the solution of ordinary differential boundary value problems. *J. Comput. Appl. Math.* **45**, 191–200.

WRIGHT, S. (1992) Stable parallel algorithms for two-point boundary value problems. *SIAM J. Sci. Statist. Comput.* **13**, 742–764.

WRIGHT, S. (1994) Stable parallel elimination for boundary value ODE's. *Numer. Math.* **67**, 521–536.