MULTIBODY DYNAMICS 2007, ECCOMAS Thematic Conference
C.L. Bottasso, P. Masarati, L. Trainelli (eds.)
Milano, Italy, 25–28 June 2007

AN ALGORITHM FOR THE SOLUTION OF BORDERED ABD LINEAR SYSTEMS ARISING FROM BOUNDARY VALUE PROBLEMS

Pierluigi Amodio[⋆], Ian Gladwell[†], and Giuseppe Romanazzi[‡]

[⋆]Dipartimento di Matematica
Università di Bari, I-70125 Bari, Italy
e-mail: amodio@dm.uniba.it
web page: http://www.dm.uniba.it/~amodio

[†]Department of Mathematics
Southern Methodist University, Dallas, TX 75275, USA
e-mail: gladwell@seas.smu.edu
web page: http://faculty.smu.edu/igladwel/

[‡]School of Computing
University of Leeds, Leeds LS2 9JT, UK
e-mail: roman@comp.leeds.ac.uk

Keywords: Boundary Value Problems, Linear systems solution, Bordered Almost Block Diagonal matrices, Cyclic Reduction.

Abstract. *We consider linear systems with coefficient matrices of Bordered ABD structure. They arise in the discretization of BVPs for ordinary and partial differential equations with non-separated boundary conditions. We report on tests (and comparisons with the well-known code* COLROW*) of the Fortran 90 cyclic reduction algorithm* BABDCR*, inserted in a modified version of the BVP code* MIRKDC*. Also, a distributed parallel version of* BABDCR *is implemented and tested for possible use replacing the BABD code* RSCALE *in the parallel version of* MIRKDC*.*

1 INTRODUCTION

Almost Block Diagonal (ABD) and Bordered ABD (BABD) [1] linear systems arise in discretizing Boundary Value Problems (BVPs) with, respectively, separated and non-separated boundary conditions, for both ordinary and partial differential equations. The coefficient matrices associated with ABD linear systems are characterized as follows: the nonzero elements are grouped in block rows, there is no intersection between the nonzero columns of two non-consecutive block rows; and, the main diagonal entries always lie inside the nonzero blocks. BABD matrices must also satisfy: the first (or the last) block row has an additional block in the right-upper (left-lower) corner whose columns only intersect the nonzero columns of the last (first) block row. In Figs. 1 and 2, we show the ABD and BABD structures arising most frequently in BVP solvers. All the nonzero blocks in the ABD and BABD structures respectively are considered dense.

$$\begin{pmatrix} B_{top} & & & & \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \\ & & & & B_{bot} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} d_a \\ f_1 \\ f_2 \\ \vdots \\ f_N \\ d_b \end{pmatrix}$$

Figure 1: ABD linear system: $S_i, R_i \in \mathbb{R}^{m\times m}$, $B_{top} \in \mathbb{R}^{m_0\times m}$, $B_{bot} \in \mathbb{R}^{(m-m_0)\times m}$, $f_i, x_i \in \mathbb{R}^m$, $d_a \in \mathbb{R}^{m_0}$, $d_b \in \mathbb{R}^{m-m_0}$.

$$\begin{pmatrix} B_a & & & & B_b \\ S_0 & R_1 & & & \\ & S_1 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_{N-1} & R_N \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} d \\ f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

Figure 2: BABD linear system: $S_i, R_i, B_a, B_b \in \mathbb{R}^{m\times m}$, $x_i, f_i, d \in \mathbb{R}^m$.

1.1 Solvers for ABD systems

For a historical view of solving ABD systems see [1, 11]. The first ABD code was *SOLVEBLOK* [8]; as for standard LU factorization applied to banded systems, it uses Gaussian elimination with partial pivoting to ensure stability and so requires fill-in. Varah's alternate row and column stable elimination [18] exploits the structure of the ABD matrices to avoid this fill-in. The packages *COLROW* and *ARCECO* in [9] are based on a modified version of Varah's procedure. Numerical experiments reported in [9] demonstrate the effectiveness of *COLROW* and *ARCECO*, and their superiority over *SOLVEBLOK* on systems arising from BVPs. Modifications have been proposed to these packages to deal with more specific structures of the coefficient matrix (see [1] and the references therein).

1.2 ABD and BABD solvers in BVODE packages

Most modern nonlinear BVP packages employ ABD packages. The BVP code *COLSYS* [4] uses *SOLVEBLOK* to solve ABD linear systems arising from using orthogonal spline collocation (OSC) at Gauss points with B-spline bases. *COLNEW* [5] uses a modified version of *SOLVEBLOK* to solve ABD linear systems arising from using OSC at Gauss points with monomial spline bases. The Mono Implicit Runge Kutta (MIRK) code with defect control *MIRKDC* [10] uses *COLROW* as a solver for ABD systems. Modified versions of *COLROW* are used in *TWPBVP* [7], a deferred correction code, in *COLMOD* [17], a modified version of *COLNEW*, and in *ACDC* [6], which uses automatic continuation and OSC at Lobatto points to solve singularly perturbed BVODEs. The code *PMIRKDC*, a parallel version of *MIRKDC*, uses the BABD package *RSCALE* which has a shared memory parallel implementation.

We propose *BABDCR* as an alternative to *COLROW* for the solution of BABD systems, and to *RSCALE* in a parallel environment. In Section 3, we discuss a modified version of *MIRKDC* which is effective for solving BVPs with non-separated boundary conditions. Numerical tests comparing *BABDCR* with *COLROW* and *RSCALE* are presented in Section 4. In Section 5, we present numerical results from a distributed memory implementation of *BABDCR*.

2 THE *BABDCR* PACKAGE

BABDCR [3, 2] solves BABD systems as in Fig. 2. The algorithm cyclically reduces the BABD matrix to derive systems of lower dimension with the same BABD structure. Suppose that the matrix in Fig. 2 is nonsingular; we reduce each pair of block row equations, for $i = 2j-1,\ j = 1, 2, \ldots, \lfloor N/2 \rfloor$,

$$\begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} \begin{pmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{pmatrix} = \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix} \tag{1}$$

into one block row equation

$$S'_{i-1} x_{i-1} + R'_{i+1} x_{i+1} = f'_{i+1} \tag{2}$$

involving only the unknowns x_{i-1} and x_{i+1}. Since the columns overlapped by the $2m \times m$ matrix $\begin{pmatrix} R_i \\ S_i \end{pmatrix}$ are linearly independent, we use a LU factorization with partial pivoting

$$P_i \begin{pmatrix} R_i \\ S_i \end{pmatrix} = \begin{pmatrix} L_i \\ T_i \end{pmatrix} U_i = \begin{pmatrix} L_i & \\ T_i & I \end{pmatrix} \begin{pmatrix} U_i \\ 0 \end{pmatrix} \tag{3}$$

where we premultiply (1) by the permutation matrix P_i and the inverse of the lower triangular matrix in (3) to obtain

$$\begin{aligned} \begin{pmatrix} L_i & \\ T_i & I \end{pmatrix}^{-1} P_i \begin{pmatrix} S_{i-1} & R_i & \\ & S_i & R_{i+1} \end{pmatrix} &\equiv \begin{pmatrix} \tilde{S}_{i-1} & U_i & \tilde{R}_{i+1} \\ S'_{i-1} & & R'_{i+1} \end{pmatrix}, \\ \begin{pmatrix} L_i & \\ T_i & I \end{pmatrix}^{-1} P_i \begin{pmatrix} f_i \\ f_{i+1} \end{pmatrix} &\equiv \begin{pmatrix} \tilde{f}_i \\ f'_{i+1} \end{pmatrix}, \end{aligned} \tag{4}$$

where S'_{i-1}, R'_{i+1} and f'_{i+1} are the blocks of the reduced equation (2). After k steps of reduction, the coefficient matrix obtained is of size $\lceil N/s \rceil + 1$ where $s = 2^k$, and can be further reduced

by combining two successive block row equations (for each $i = (2j-1)s+1$, with $j = 1, 2, \ldots, \lfloor \lceil N/s \rceil / 2 \rfloor$)

$$\begin{pmatrix} S_{i-s}^{(k)} & R_i^{(k)} & \\ & S_i^{(k)} & R_{i+s}^{(k)} \end{pmatrix} \begin{pmatrix} x_{i-s} \\ x_i \\ x_{i+s} \end{pmatrix} = \begin{pmatrix} f_i^{(k)} \\ f_{i+s}^{(k)} \end{pmatrix}$$

to give the block row equation $S_{i-s}^{(k+1)} x_{i-s} + R_{i+s}^{(k+1)} x_{i+s} = f_{i+s}^{(k+1)}$. The reduction ends after $p = \lceil \log_2 N \rceil$ steps with the 2×2 block linear system

$$\begin{pmatrix} B_a & B_b \\ S_0^{(p)} & R_N^{(p)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_N \end{pmatrix} = \begin{pmatrix} d \\ f_N^{(p)} \end{pmatrix}. \tag{5}$$

The algorithm then proceeds by solving (5) and using a back-substitution phase to compute the unknowns $x_1, \ldots, x_{N-1}$.

The null blocks in equation (4) permit us to reduce the memory requirement and the computational cost. In the reduction, after $k+1$ steps, we save the matrices $S_{i-s}^{(k+1)}$ and $R_{i+s}^{(k+1)}$ in place of $S_{i-s}^{(k)}$ and $R_{i+s}^{(k)}$, the product $T_i L_i^{-1}$ in place of $S_i^{(k)}$ and the vector $f_{i+s}^{(k+1)}$ in place of $f_{i+s}^{(k)}$, respectively. In the back-substitution phase, we save the first m elements of $P_i^{(k)} \begin{pmatrix} f_i^{(k)} \\ f_{i+s}^{(k)} \end{pmatrix}$ in place of $f_i^{(k)}$. The additional memory required has size $m \times m$ for each of the $N-1$ reductions corresponding to the first m rows of $P_i^{(k)} \begin{pmatrix} S_{i-s}^{(k)} \\ R_{i+s}^{(k)} \end{pmatrix}$. To leading order in powers of m and N the computational cost of the factorization is $\frac{14}{3} m^3 N$ flops and of the back-substitution is $6m^2 N$.

3 BVPS AND THE *MIRKDC/PMIRKDC* CODES

MIRKDC [10] solves BVPs

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)), \qquad t \in [a, b] \tag{6}$$

where $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$, with separated BCs

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \begin{pmatrix} \mathbf{g}_0(\mathbf{y}(a)) \\ \mathbf{g}_1(\mathbf{y}(b)) \end{pmatrix} = 0. \tag{7}$$

It uses Mono-Implicit Runge Kutta (MIRK) formulas to discretize (6) on a subdivision $\{t_i\}_{i=0}^N$ of $[a, b]$. A continuous solution approximation is obtained by means of a Continuous MIRK (CMIRK) scheme, and is used to provide defect control and mesh selection capabilities. The MIRK scheme applied to the BVP system (6)-(7) on N subintervals, yields the nonlinear system

$$\mathbf{\Phi}(\mathbf{Y}) = (\mathbf{\Phi}_0(\mathbf{Y})^T, \ldots, \mathbf{\Phi}_N(\mathbf{Y})^T)^T = 0, \quad \text{where } \mathbf{\Phi}_i : \mathbb{R}^{m(N+1)} \longrightarrow \mathbb{R}^m$$

and $\mathbf{Y} = (\mathbf{y}_0^T, \ldots, \mathbf{y}_N^T)^T$, $\mathbf{y}_j \in \mathbb{R}^m$. This is solved using a modified version of the Newton iteration described by $\mathbf{Y}^{(q+1)} = \mathbf{Y}^{(q)} + \Delta \mathbf{Y}^{(q)}$, for $q = 0, 1, \ldots$, where

$$\left[\frac{\partial \mathbf{\Phi}(\mathbf{Y}^{(q)})}{\partial \mathbf{Y}} \right] \Delta \mathbf{Y}^{(q)} = -\mathbf{\Phi}(\mathbf{Y}^{(q)}) \tag{8}$$

given $\mathbf{Y}^{(0)}$. For separated boundary conditions (7), the ABD linear system arising in (8) is as shown in Fig. 1 with

$$S_i = -I - h_i K_{i,i}, \qquad R_i = I - h_i K_{i+1,i},$$

where the dense block $K_{i,j}$ depends on $\mathbf{f}$ and on the Runge-Kutta formulas chosen.

MIRKDC uses *COLROW* to solve systems of the form (8). The computational cost of this ABD solver, to leading order in m and m_0 (the number of boundary conditions at $x = a$), is $\left(\frac{5}{3}m^3 + mm_0^2\right) N$ flops for the factorization and $4m^2N$ for the solution. This means that its computational cost varies from approximately $\frac{5}{3}m^3N$ and $\frac{8}{3}m^3N$ (when $m_0 \approx 0$ and $m_0 \approx m$, respectively), and is $\frac{23}{12}m^3N$ in the most commonly occurring case where $m_0 \approx \frac{m}{2}$. *COLROW* does not require fill-in vectors.

Since linear algebra is the most expensive (in term of execution times) part of *MIRKDC* and since *COLROW* is a sequential algorithm, the parallel version (written for a shared memory computer) *PMIRKDC* [14] uses the parallel linear system solver *RSCALE* . This package solves BABD systems and has a computational cost of approximately $\frac{20}{3}m^3N$ and $6m^2N$ for the factorization and the solution, respectively. Moreover, it requires fill-in vectors of length $m^2(N + 1)$.

Our sequential variant, *MIRKDC_NS*, solves BVPs (6) with non-separated boundary conditions

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = 0. \tag{9}$$

We stress that (6), (9) may also be solved directly by *MIRKDC* by recasting the problem in the form

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t)), \quad \mathbf{z}' = 0, \qquad t \in [a, b], \tag{10}$$

with separated boundary conditions

$$\mathbf{y}(a) - \mathbf{z}(a) = 0, \qquad \mathbf{g}(\mathbf{z}(b), \mathbf{y}(b)) = 0. \tag{11}$$

If the function g in (9) comprises k nonseparated and $m-k$ separated boundary conditions with m_0 boundary conditions at $x = a$, then the size of the equivalent BVP is $(m + k)$ with $(m_0 + k)$ boundary conditions at $x = a$.

MIRKDC_NS is equivalent to *MIRKDC*. The only essential difference is that the linear systems (8) have a BABD structure as in Fig. 2 and are solved using *BABDCR*. We made the following further modifications in *MIRKDC*:

- The permutation array of length $m(N + 1)$ becomes one of length $2mN$;
- The fill-in described in Section 2, adds $m^2(N-1)$ locations in the array containing blocks S_i, R_i of the current BABD Jacobian in (8);
- The arrays containing blocks B_a, B_b of the Jacobian in (8), in Fig. 2, are of dimension $m \times m$ instead of $m_0 \times m$, $(m - m_0) \times m$, respectively;
- In *BABDCR* the right hand side is overwritten by the solution (*COLROW* does not overwrite the solution). Therefore, before calling *BABDCR* , the right hand side must already occupy the locations where the solution will be placed.

4 COMPARING THE LINEAR SYSTEM SOLVERS

We compare *BABDCR* with *COLROW* and *RSCALE* on the ABD and BABD linear systems generated by the codes *MIRKDC* and *MIRKDC_NS* respectively. In our experiments, we use a fourth order method, based on the optimal 4-th order, 3-stage MIRK scheme, see [10].

Consider the ABD linear systems generated by *MIRKDC* applied to a linear BVP

$$\mathbf{y}' = M\mathbf{y}(t), \quad M \in \mathbb{R}^{m\times m} \tag{12}$$

with linear boundary conditions

$$B_{top}\mathbf{y}(a) = \mathbf{d}_a \in \mathbb{R}^{m_0}, \quad B_{bot}\mathbf{y}(b) = \mathbf{d}_b \in \mathbb{R}^{m-m_0}$$

First, we fix $m = 20$ and $m_0 = 10$ then $m = 40$ and $m_0 = 20$. B_{top}, B_{bot}, both of full rank $m_0 = m - m_0$, and M are randomly generated full matrices. The matrix M is obtained by computing $M = Q\Lambda Q^T$ where Q is an orthogonal matrix arising from the QR factorization of a random matrix and Λ is a diagonal matrix with $m/2$ positive and $m/2$ negative values of moderate size such that the resulting BVP is well conditioned. *BABDCR* and *RSCALE* use

$$B_a = \begin{pmatrix} B_{top} \\ 0 \end{pmatrix}, B_b = \begin{pmatrix} 0 \\ B_{bot} \end{pmatrix} \in \mathbb{R}^{m\times m}, \quad \mathbf{d} = \begin{pmatrix} \mathbf{d}_a \\ \mathbf{d}_b \end{pmatrix} \in \mathbb{R}^m,$$

to obtain a BABD system as in Fig. 2.

Overall timings and errors, given in Tables 1 and 2, show that *COLROW* is more than twice as fast as *BABDCR* and more than four times as fast as *RSCALE*. The errors for *COLROW* and *BABDCR* are similar, but *RSCALE* is less accurate. These computational costs are close to the predictions of theory. Since $m_0 = m/2$, *COLROW* should be approximately 2.4 and 3.5 times faster then *BABDCR* and *RSCALE*, respectively.

	time			error		
	N=256	N=512	N=1024	N=256	N=512	N=1024
BABDCR	3.71e-02	7.12e-02	0.152	1.65e-13	2.88e-13	3.46e-13
COLROW	1.46e-02	2.83e-02	6.34e-02	1.55e-13	2.05e-13	7.08e-13
RSCALE	6.83e-02	0.136	0.274	2.54e-12	6.02e-12	3.01e-11

Table 1: ABD systems generated by *MIRKDC* applied to a linear BVP of size $m = 20$.

	time			error		
	N=256	N=512	N=1024	N=256	N=512	N=1024
BABDCR	0.213	0.461	0.924	6.51e-13	9.20e-13	1.64e-12
COLROW	8.69e-02	0.201	0.402	7.02e-13	5.36e-13	8.07e-13
RSCALE	0.481	0.972	1.950	1.66e-11	3.20e-11	2.74e-10

Table 2: ABD systems generated by *MIRKDC* applied to a linear BVP of size $m = 40$.

For BABD linear systems, we apply *MIRKDC_NS* to a linear BVP (12) with non-separated boundary conditions

$$B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{d} \in \mathbb{R}^m.$$

For $m = 20$ we investigate two cases:

(a) M is a well-conditioned matrix with eigenvalues -102, -10, -7, -4, -3, -2.5, -1.3, -1, -0.5, -0.4, 0.2, 0.3, 1, 1, 2, 2.5, 3, 4, 11, 25;

(b) M is an ill-conditioned matrix with eigenvalues -9, -3.5, -3, -2, -2, -1.5, -1.5, -1.25, -0.5, -1e-08, 0.25, 0.5, 0.5 , 1, 3, 4, 5, 7, 8, 1e+08.

For $m = 40$ we just consider a well-conditioned case with M having eigenvalues -102, -90, -75, -53, -51, -42.5, -38, -30, -27, -14, -13, -12.5, -9.3, -8, -5.5, -4, -2.5, -1.3, -1, -0.5, 0.2, 0.3, 1, 1, 2, 2.5, 3, 4, 11, 15, 20.5, 25, 28, 33, 40, 45, 46, 50.5, 54, 57

To use *COLROW*, we convert the BABD system to an ABD linear system with blocks of double the size, see Fig. 3. In the case of k nonseparated boundary conditions and $m_0 = \lfloor (m-k)/2 \rfloor$ separated boundary conditions at $x = a$, we observe that *BABDCR* is faster than *COLROW* when $k > \lceil m/3 \rceil$. Since, in our example, the number of nonseparated boundary conditions is $k = m$, the computational cost of *COLROW* is approximately $\frac{46}{3}m^3N$ for the factorization and $16m^2N$ for the solution. So, theoretically *BABDCR* is more than three times as fast as *COLROW*.

$$\begin{pmatrix} -I & I & & & & \\ S_0 & 0 & R_1 & & & \\ & -I & 0 & I & & \\ & & \ddots & & \ddots & \\ & & & S_{N-1} & 0 & R_N \\ & & & & -I & 0 \\ & & & & & B_b \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{z}_0 \\ \mathbf{x}_1 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{x}_N \\ \mathbf{z}_N \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{f}_1 \\ 0 \\ \vdots \\ \mathbf{f}_N \\ 0 \\ \mathbf{d} \end{pmatrix}$$

Figure 3: ABD system of doubled size (corresponding to the BABD system in Fig. 2). $\mathbf{z}_i$, $i = 0, \ldots, N$ are the new unknowns, $\mathbf{z}_N = \mathbf{z}_{N-1} = \ldots = \mathbf{z}_0 = \mathbf{x}_0$.

In Tables 3-4 we compare the errors and timings of the three linear solvers. From the results in Table 3, for the cases (a) and (b) *BABDCR* is approximately 3 times as fast as *COLROW* and more than 1.5 times as fast as *RSCALE*. Timings associated with *COLROW* include converting the linear system from BABD to ABD structure in Fig. 3 (which is between 10% and 15% of the total time taken by *COLROW*). The errors associated with *BABDCR* and *COLROW* are similar, and *RSCALE* is the least accurate algorithm. The errors of the three methods for case (b), are given in Table 4. These errors are large, because the BVP is ill-conditioned. Observe that *BABDCR* and *COLROW* are still significantly more accurate than *RSCALE*. Table 5 for $m = 40$ in comparison with Table 3 for $m = 20$ shows that the relative costs of the three algorithms increase consistently cubically with m.

	N=256	N=512	N=1024
BABDCR	3.61e-02	7.03e-02	0.151
COLROW	0.102	0.224	0.464
RSCALE	6.83e-02	0.136	0.287

Table 3: Timings for the solution of BABD systems generated by *MIRKDC_NS* applied to a linear BVP with $m = 20$. (For *COLROW* the timings without the conversion phase are $0.0859, 0.190, 0.394$ for N$= 256, 512, 1024$ respectively.)

	case (a)			case (b)		
	N=256	N=512	N=1024	N=256	N=512	N=1024
BABDCR	7.62e-13	1.22e-12	1.19e-12	6.28e-04	2.22e-04	8.00e-05
COLROW	7.53e-13	4.10e-13	1.25e-12	2.80e-04	2.16e-04	9.25e-05
RSCALE	5.17e-12	2.90e-11	6.02e-11	1.90e-02	8.55e-03	1.08e-02

Table 4: Errors for the solution of BABD systems generated by *MIRKDC_NS* with $m = 20$.

	N=256	N=512	N=1024
BABDCR	0.211	0.449	0.924
COLROW	0.740	1.481	2.971
RSCALE	0.481	0.967	1.949

Table 5: Timings for the solution of BABD systems generated by *MIRKDC_NS* applied to a linear BVP with $m = 40$. (For *COLROW* the timings without the conversion phase are $0.684, 1.370, 2.746$ for N$= 256, 512, 1024$ respectively.)

To emphasize the advantages of *BABDCR*, Tables 4-4 give statistics for one call to *MIRKDC_NS* applied to the BVP (6), (9) and to *MIRKDC* applied to the equivalent BVP of double the size (10) with separated boundary conditions (11). The columns represent the size, N, of the computed mesh (initially set equal to 256) and the number (with the timings) of the factorizations performed and linear system solutions on that mesh. Then, in the last two rows, we give the total time for the linear algebra calls and for the remaining operations.

Both codes were applied to the problem with M having eigenvalues as in case (a) using the MIRK/CMIRK scheme of order 4 [10]. From Tables 4 and 4, starting from the same mesh, the same number of factorizations and linear system solutions are required. Observe that *BABDCR* in *MIRKDC_NS* saves more than one half of the linear algebra time in comparison with *COLROW*. Since the error given by the two codes is similar and the time required outside the linear algebra calls remains constant, we expect that, in general, for a well-conditioned BVP with nonseparated boundary conditions, the two codes will perform the same number of factorizations and linear system solutions and that *MIRKDC_NS* will run faster than *MIRKDC*.

MESH	♯FACTs	time	♯SOLVEs	time
256	1	0.32e-01	2	0.98e-02
224	1	0.25e-01	2	0.78e-02
246	1	0.29e-01	2	0.78e-02
Total:	3	0.87e-01	6	0.25e-01
Total monitored Linear Algebra time:				0.11 secs.
Total monitored non Linear Algebra time:				0.12 secs.

Table 6: *MIRKDC_NS* (using *BABDCR*) applied to the linear problem in case (a) with an initial mesh of 256 points and error tolerance 1e-07.

5 PARALLEL IMPLEMENTATION OF *BABDCR*

In the previous section we observed the superiority in timings of *MIRKDC_NS* (using *BABDCR*) over *MIRKDC* (using *COLROW*) when BVPs with nonseparated boundary conditions are

MESH	♯FACTs	time	♯SOLVEs	time
256	1	0.11	2	0.16e-01
224	1	0.75e-01	2	0.12e-01
246	1	0.82e-01	2	0.14e-01
Total:	3	0.27	6	0.41e-01
Total monitored Linear Algebra time:				0.31 secs.
Total monitored non Linear Algebra time:				0.12 secs.

Table 7: *MIRKDC* (using *COLROW*) applied to the linear problem of doubled size in case (a) with an initial mesh of 256 points and error tolerance 1e-07.

solved. Here, we show that a parallel implementation of *BABDCR* also pays off in this context.

To this aim, we consider a straightforward distributed memory parallel implementation of *BABDCR* that uses MPI procedures [16] to implement the clearly defined parallel structures visible in the CR algorithm. We preliminarily observe that all the reductions from (1) to (2) may be performed in parallel if the two adjacent block rows are stored in the same processor. Therefore, following the idea in *PMIRKDC* [14], we suppose that the original BVP has been discretized in order so that each processor is only involved with a subinterval of $[a, b]$. This implies that, before calling the factorization procedure, each processor has been assigned some consecutive block rows of the original matrix and the boundary conditions. In the factorization step, the first reductions are computed in parallel on all the p processors without any communication. When each processor has one block row of the reduced matrix, than it is necessary to perform a send/receive communication before each additional reduction step. This means that, if $N \gg p$, the computational cost of the factorization is reduced by a factor p and requires $\lceil \log_2 p \rceil$ communications of blocks of size $m \times 2m$. We proceed analogously for the linear system solution, where each vector which is sent/received has length m. Finally, each processor only computes its own part of the solution. See [15] for more details and [2] for the original proposal.

We use a cluster of 32 processors (2.4 GHz Intel Xeon, 4Mb L2 cache) with the Intel Fortran compiler 8.1 for Linux. In Table 8 we show the speedup of the algorithm. Here, the speedup is defined as the ratio between the execution time (time of factorization plus time of solution) of the serial version of the algorithm and the execution time of the parallel version run on the specified number of processors. (The time for the serial version on one processor and for the parallel version run on one processor are essentially the same.) Note that the cyclic reduction algorithm has improved speed-up as the dimension m increases. Particularly, for 32 processors we need large values of m to observe almost linear speedup. In fact, since the factorization of an $m \times m$ matrix is the principal cost in each reduction, as m increases the arithmetic cost $O(m^3)$ prevails over the time spent in the communication $O(m^2)$.

The slightly better than linear speedup (observed in many cases for $m > 4$) is at least in part due to a cache effect. In the serial version ($p = 1$) all the data is stored on a single processor, and the cache (4Mbytes) is always full hence requiring frequent data transfers from main memory. So, memory access takes longer than for the parallel runs with $p > 1$ processors where the data is shared over the processors, each with its own 4Mbytes cache memory, resulting in needing fewer accesses to main memory.

Direct comparisons with *PMIRKDC*[14] using *RSCALE* are not possible since *PMIRKDC* is designed specifically for shared memory architectures and a distributed memory version would require redesign of parts of the algorithm, not simply a change of implementation language.

	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$
$m=4$	1.983	3.398	5.474	6.426	6.978
$m=16$	2.052	4.109	7.913	15.116	24.965
$m=64$	2.069	4.141	7.940	15.263	25.458
$m=256$	2.169	4.395	8.536	16.316	29.519

Table 8: Speedup for the parallel *BABDCR* algorithm with $N = 1024$.

6 CONCLUSIONS

COLROW has long been considered a fast code for solving ABD linear systems and is widely used in packages for the solution of BVPs with separated boundary conditions. In these packages, the solution of problems with non-separated boundary conditions can only be achieved by modifying the original problem to give a new problem of double the size with separated boundary conditions. Starting from the BVP package *MIRKDC*, we have shown that *BABDCR* runs faster than *COLROW* and with approximately the same accuracy when the original problem has non-separated boundary conditions. In addition, when it is possible to use a distributed parallel computer (with a moderate number of processors) the implementation of *BABDCR* within a version of *MIRKDC* leads to close to linear speedup.

REFERENCES

[1] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut and M. Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications, *Numer. Linear Algebra Appl.*, **7**, no. 5, 275–317, 2000.

[2] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM J. Sci. Comput.*, **18**, no. 1, 56–68, 1997.

[3] P. Amodio and G. Romanazzi. BABDCR: a Fortran 90 package for the solution of Bordered ABD systems. *ACM Trans. Math. Software*, **32**, no. 4, 597–608, 2006.

[4] U.M. Ascher, J. Christiansen and R.D. Russell. Algorithm 569: COLSYS: Collocation Software for Boundary-Value ODEs. *ACM Trans. Math. Software*, **7**, no. 2, 223–229, 1981.

[5] G. Bader and U. Ascher. A new basis implementation for a mixed order boundary value ODE solver. *SIAM J. Sci. Statist. Comput.*, **8**, no. 4, 483–500, 1987.

[6] J.R. Cash, G. Moore and R.W. Wright. An automatic continuation strategy for the solution of singularly perturbed nonlinear boundary value problems. *ACM Trans. Math. Software*, **27**, no. 2, 245–266, 2001.

[7] J.R. Cash and R.W. Wright. A deferred correction method for nonlinear two-point boundary value problems: Implementations and numerical evaluation. *SIAM J. Sci. Statist. Comput.*, **12**, no. 4, 971–989, 1991.

[8] C. De Boor and R. Weiss. SOLVEBLOK: A package for solving almost block diagonal linear systems. *ACM Trans. Math. Software* **6** no. 1 (1980) 80–87.

[9] J.C. Diaz, G. Fairweather and P. Keast. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Software*, **9**, no. 3, 358–375, 1983.

[10] W.H. Enright and P.H. Muir. Runge-Kutta software with defect control for boundary value ODEs. *SIAM J. Sci. Comput.*, **17**, no. 2, 479–497, 1996.

[11] G. Fairweather and I. Gladwell. Algorithms for almost block diagonal linear systems. *SIAM Rev.*, **46**, no. 1, 49–58, 2004.

[12] B. Garrett and I. Gladwell. Solving bordered almost block diagonal systems stably and efficiently. *J. Comput. Methods Sci. Engrg.*, **1**, 75–98, 2001.

[13] K.R. Jackson and R.N. Pancer. The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs. Techinical Report n. 255/91, Computer Science Department, University of Toronto, 1992.

[14] P.H. Muir, R.N. Pancer and K.R. Jackson. PMIRKDC: a parallel mono-implicit Runge-Kutta code with defect control for boundary value ODEs. *Parallel Comput.*, **29**, 711–741, 2003.

[15] G. Romanazzi. Numerical Solution of Bordered Almost Block Diagonal linear systems arising from BVPs. Ph.D. Thesis, Università di Bari, 2006.

[16] M. Snir, S.W. Otto, S. Huss-Lederman, D. Walker and J. Dongarra. *MPI: The Complete Reference.* The MIT Press, Cambridge, Massachusetts, 1996.

[17] R.W. Wright, J. Cash and G. Moore. Mesh selection for stiff two-point boundary value problems. *Numer. Algorithms*, **7**, 205–224, 1994.

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