The Parallel Solution of ABD Systems Arising in Numerical Methods for BVPs for ODEs

K. R. Jackson and R. N. Pancer

Technical Report No. 255/91

May 1992.

Computer Science Department
University of Toronto
Toronto, Ontario, Canada M5S 1A4

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.

Abstract

Many commonly used numerical methods for the solution of Boundary Value Problems (BVPs) for Ordinary Differential Equations (ODEs) contain significant components that can be parallelized easily, and recent investigations have shown that substantial speedups are attainable on machines with a modest number of processors. However, a stage in most of these methods is the numerical solution of an *Almost Block Diagonal* (ABD) system of linear algebraic equations. Several authors have proposed parallel algorithms for solving such systems; unfortunately most are potentially unstable or offer only limited parallelism. As a result, solving ABD systems has proven to be a bottleneck in the overall execution time of parallel BVP codes.

In recent papers, Wright presents new stable parallel algorithms for solving ABD systems ([Wrig 90b] and [Wrig 91a]). Each algorithm attains the theoretically optimal speedup for the problem if enough processors are available, and each can be adapted for use on architectures with fewer than the optimal number of processors. The algorithms in §3 of this paper were discovered independently and are similar to those in [Wrig 90b] and [Wrig 91a]. Extensions to the basic algorithm are described which make better use of available processors during the decomposition stage in order to increase parallelism in the back-solve stage.

A new algorithm based on *eigenvalue rescaling* is presented in §4 which can be up to twice as fast as the fastest algorithm in §3. Numerical experiments show that this new algorithm does not exhibit the instability inherent in some earlier schemes. In addition, some insight supporting the numerical evidence is given.

1 Introduction

The linear systems considered in this paper frequently arise in the numerical solution of Boundary Value Problems (BVPs) for Ordinary Differential Equations (ODEs) of the form

$$y'(t) = f(t, y(t)), \quad t \in [a, b], \tag{1}$$

$$g(y(a), y(b)) = 0, \tag{2}$$

where $y, f, g \in \mathcal{R}^n$. If the boundary conditions are separable, (2) can be written as

$$g(y(a), y(b)) = \begin{bmatrix} g_a(y(a)) \\ g_b(y(b)) \end{bmatrix} = 0, \tag{3}$$

where $g_a \in \mathcal{R}^{n_a}$ and $g_b \in \mathcal{R}^{n_b}$ with $n_a + n_b = n$.

When solving this problem numerically, a mesh is first defined which subdivides the problem interval:

$$a = t_0 < t_1 < \cdots < t_M = b.$$

Many techniques then proceed to compute a discrete approximation of the form

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix} \in \mathcal{R}^{(M+1)n}$$

where $y_i \approx y(t_i)$, $y_i \in \mathcal{R}^n$, $i = 0, \ldots, M$. The vector Y is obtained by solving a discrete system $\Phi(Y) = 0$ that depends both on the system of ODEs and the underlying numerical method used to discretize the continuous problem. Although the residual function Φ may be written in many forms, it usually has one component only per subinterval, and often each component depends only on the unknowns local to its subinterval. If, in addition, the boundary conditions are separable, $\Phi(Y) = 0$ can be written as

$$\Phi(Y) = \begin{bmatrix} g_a(y_0) \\ \phi_0(y_0, y_1) \\ \phi_1(y_1, y_2) \\ \vdots \\ \phi_{M-1}(y_{M-1}, y_M) \\ g_b(y_M) \end{bmatrix} = 0, \tag{4}$$

where $\phi_i \in \mathcal{R}^n, i = 0 \ldots M-1$, and $\Phi \in \mathcal{R}^{(M+1)n}$. A variant of Newton's method is often used to solve (4), yielding an iteration of the form

$$\mathcal{J}^{(q)} \left[Y^{(q+1)} - Y^{(q)} \right] = -\Phi(Y^{(q)}), \quad q = 0, 1, \ldots \tag{5}$$

where $\mathcal{J}^{(q)} \approx \partial\Phi(Y^{(q)})/\partial Y$, the Jacobian of Φ.

During each iteration, $\mathcal{J}^{(q)}$ may be recomputed and the linear system (5) is solved. Because of the structure of Φ shown in (4), the Jacobian of Φ has a "staircase" or Almost Block Diagonal (ABD) form

$$\frac{\partial \Phi(Y)}{\partial Y} = \begin{bmatrix} \mathcal{B}_a & & & & & \\ S_0 & T_0 & & & & \\ & S_1 & T_1 & & & \\ & & S_2 & T_2 & & \\ & & & \ddots & \ddots & \\ & & & & S_{M-1} & T_{M-1} \\ & & & & & \mathcal{B}_b \end{bmatrix} \in \mathcal{R}^{(M+1)n \times (M+1)n} \tag{6}$$

where

$$S_i = \frac{\partial \phi_i(y_i, y_{i+1})}{\partial y_i} \in \mathcal{R}^{n \times n}, \quad T_i = \frac{\partial \phi_i(y_i, y_{i+1})}{\partial y_{i+1}} \in \mathcal{R}^{n \times n}, \quad i = 0, 1, \ldots, M-1,$$

and

$$\mathcal{B}_a = \frac{\partial g_a(y_0)}{\partial y_0} \in \mathcal{R}^{n_a \times n}, \quad \mathcal{B}_b = \frac{\partial g_b(y_M)}{\partial y_M} \in \mathcal{R}^{n_b \times n}.$$

When (5) is converging at an acceptable rate, $\mathcal{J}^{(q)}$ may be held fixed for several iterations. Therefore, if the factors of $\mathcal{J}^{(q)}$ are stored, several linear systems in (5) may be solved at a reduced cost with only a forward elimination and back-solve on subsequent right-hand sides $-\Phi(Y^{(k)})$, $k = q+1, q+2, \ldots$.

There are several other issues that need be addressed in the numerical solution of BVPs, including selecting an appropriate mesh, choosing a specific formula for the residual function, and determining a convergence criteria for (5). See [Asch 88] for details.

Much parallelism is inherent in the approach outlined above. For example, the residual components of $\Phi(Y^{(q)})$ can be evaluated independently, and the block-pairs $[S_i, T_i]$ of $\mathcal{J}^{(q)}$ can be constructed independently each time $\mathcal{J}^{(q)}$ is re-evaluated. In [Benn 90], a parallel version of COLNEW [Bade 87] is implemented and the speed-up achievable by parallelizing the Jacobian set-up phase is investigated. A high percentage (60-80%) of the total execution time in COLNEW is spent during set-up where large blocks are "condensed" to form (6). Condensing in parallel, therefore, is very effective and in fact shifts the most time-consuming phase of the code: run-time profiling in [Benn 90] shows that the factorization and solution of the condensed ABD system becomes the bottleneck in the parallel implementation. In codes that do not use condensation the resulting ABD system can be much larger, and its factorization and solution an even bigger bottleneck. In COLSYS [Asch 81], for example, the ABD factorization often accounts for more than 50% of the total execution time [Muir 91]. These statistics emphasize the importance of developing a parallel ABD solver.

Efficient sequential codes for solving ABD systems have been available for several years—two examples are SOLVEBLOK [deBo 80] and COLROW [Diaz 83]. Both codes perform the factorization in $O(Mn^3)$ time. SOLVEBLOK eliminates by rows and COLROW uses a variation of alternate row and column elimination to avoid fill-in. Stability is achieved by a pivoting strategy which controls element growth during the factorization. The pivoting results in an implicit decoupling of the underlying increasing and decreasing fundamental solution modes, an essential process for the stable solution of a BVP ([Asch 88, Chapter 6]).

The greatest potential for speed-up in the parallel solution of ABDs exists across the blocks (i.e. in M), since typically $M \gg n$ in a BVP. Parallelism within blocks is complementary, and could be exploited also if n is large, but this possibility is not pursued here. Surpressing the dependence on n, therefore, the theoretically best parallel complexity is $O(\log M)$ block-steps. Most parallel algorithms proposed recently, however, either do not achieve the optimal speed-up or suffer from poor stability properties. Wright and Pereyra [Wrig 90a] present a block factorization algorithm which is essentially equivalent to *compactification*—an algorithm known to be potentially unstable ([Asch 88, page 153]). They propose using the parallel algorithm initially, and when instability is detected, switching to a more stable sequential method. Paprzycki and Gladwell [Papr 89] describe a "tearing" algorithm in which the original ABD is partitioned into several smaller matrices, each of which is ABD. These represent sub-BVPs which can be solved independently, after which the solutions are combined. Unfortunately, although it is always possible to select intermediate boundary conditions and construct the smaller ABDs, there is no guarantee that the sub-problems will be well-conditioned. In addition, the authors found that even when a problem could be solved stably with this method, the speed-up achieved was less than expected. Lentini [Lent 89] suggests performing Gaussian elimination with pivoting simultaneously from both ends of the matrix. This leads to a stable algorithm utilizing two processors, but it cannot be generalized for greater parallelism. Finally, Ascher and Chan [Asch 89] propose solving the ABD system by first forming the normal equations. The resulting system is symmetric, positive-definite, block-tridiagonal and can be solved stably in $O(\log M)$ block-steps using block cyclic reduction [Hell 76]. The drawback here, of course, is that by forming the normal equations explicitly the 2-norm condition number of the original system is squared.

Wright was the first to present *stable* parallel algorithms that attain the theoretically optimal speed-up for this problem. In [Wrig 90b] a "structured orthogonal factorization" is described which—when imbedded in a cyclic reduction algorithm—solves the ABD system stably in $O(\log M)$ block-steps with as few as $M/\log M$ processors. Stability is assured since the algorithm is essentially a QR-factorization applied to a row and column permuted version of the original ABD. In [Wrig 91a] the local orthogonal transformations used in [Wrig 90b] are replaced by Gauss transformations with row partial-pivoting resulting in a substantial speed-up even though the number of block-steps is the same. The algorithm is equivalent to Gaussian elimination with restriced row partial-pivoting applied to the original ABD, and is stable for a wide range of problems, although rapid error growth can arise in some cases [Wrig 91b].

The algorithms described in §3 of this paper were discovered independently and are similar to those of Wright. Each is based on applying a variation of block cyclic reduction *directly* to (6) (i.e. the normal equations are not formed). In §2 the basic (and unstable) block cyclic reduction algorithm is briefly outlined. In §3 the block multiplications used in the unmodified algorithm are replaced by *Stable Local Factorization* (*SLF*) transformations. These transformations give rise to stable parallel algorithms for solving ABD systems—a thorough stability analysis is given in [Wrig 90b] and [Wrig 91a]. In addition, the algorithms are extended to increase parallelism in the back-solve stage by making better use of available processors during the decomposition stage. In §4 a new technique based on *eigenvalue rescaling* is presented in which the ABD is first transformed so that cyclic reduction as described in §2 can be performed stably. Although the number of block-steps is the same as that of the algorithms in §3, rescaling can be up to twice as fast as the fastest SLF-based algorithm because of reduced operation counts within each block-step. The storage requirements and operations counts of the various algorithms are summarized in §5. Numerical results are presented in §6.

None of the algorithms presented in this paper require that the boundary conditions are separable, so a more general form of (6) will be assumed. Augmented with its right-hand side, this linear system has the form

$$[\mathcal{J}_1|\Phi] = \left[\begin{array}{cccccc|c} \mathcal{B}_a & & & & & \mathcal{B}_b & g \\ S_0 & T_0 & & & & & \phi_0 \\ & S_1 & T_1 & & & & \phi_1 \\ & & S_2 & T_2 & & & \phi_2 \\ & & & \ddots & \ddots & & \vdots \\ & & & & S_{M-1} & T_{M-1} & \phi_{M-1} \end{array}\right] \tag{7}$$

where $\mathcal{B}_a, \mathcal{B}_b \in \mathcal{R}^{n\times n}$. In addition, the algorithms in §4 are slightly more efficient in terms of both storage and speed when $T_i = I$, $i = 0, \ldots, M-1$:

$$[\mathcal{J}_2|\Phi] = \left[\begin{array}{cccccc|c} \mathcal{B}_a & & & & & \mathcal{B}_b & g \\ V_0 & I & & & & & \phi_0 \\ & V_1 & I & & & & \phi_1 \\ & & V_2 & I & & & \phi_2 \\ & & & \ddots & \ddots & & \vdots \\ & & & & V_{M-1} & I & \phi_{M-1} \end{array}\right] \tag{8}$$

This second form of the Jacobian arises naturally, for example, in multiple shooting and codes based on implicit Runge-Kutta formulas (see [Asch 88]). If (8) does not arise naturally from the discretization, it can be computed by multiplying the i-th block-row of (7) through by T_i^{-1} (providing T_i^{-1} exists and is not too badly conditioned)—a process which is obviously highly parallel.

2 Block Cyclic Reduction

Cyclic reduction was originally proposed as a stable sequential algorithm for solving symmetric positive-definite tridiagonal linear systems ([Hock 65], [Hock 70], [Golu 83]). It does, however, have considerable inherent parallelism and can easily be adapted to solve ABD systems. To this end, consider the second form of the Jacobian (8). Assuming $M/2$ processors are available initially, the first step or "sweep" of the reduction assigns each pair of block-rows i and $i+1$ to a processor, multiplies row i by V_{i+1} and subtracts it from row $i+1$, giving

$$\left[\begin{array}{cccccccc|c}
\mathcal{B}_a & & & & & & & \mathcal{B}_b & g \\
V_0 & I & & & & & & & \phi_0 \\
-V_1V_0 & & I & & & & & & \phi_1 - V_1\phi_0 \\
& & V_2 & I & & & & & \phi_2 \\
& & -V_3V_2 & & I & & & & \phi_3 - V_3\phi_2 \\
& & & & \ddots & \ddots & & & \vdots \\
& & & & & V_{M-2} & I & & \phi_{M-2} \\
& & & & & -V_{M-1}V_{M-2} & & I & \phi_{M-1} - V_{M-1}\phi_{M-2}
\end{array}\right] \tag{9}$$

This effectively uncouples the odd-indexed unknowns from the system. Once $y_{2i}, i = 0 \ldots M/2$, have been computed, $y_{2i+1}, i = 0 \ldots M/2-2$, can be obtained from (9) by

$$y_{2i+1} = \phi_{2i} - V_{2i}y_{2i}, \quad i = 0, \ldots, M/2-2. \tag{10}$$

The even-indexed unknowns are computed during the second sweep by solving a reduced or compacted ABD system constructed from the odd rows and columns of (9):

$$\left[\begin{array}{ccccc|c}
\mathcal{B}_a & & & & \mathcal{B}_b & g \\
-V_1V_0 & I & & & & \phi_1 - V_1\phi_0 \\
& -V_3V_2 & I & & & \phi_3 - V_3\phi_2 \\
& & \ddots & \ddots & & \vdots \\
& & & -V_{M-1}V_{M-2} & I & \phi_{M-1} - V_{M-1}\phi_{M-2}
\end{array}\right] \tag{11}$$

The algorithm proceeds recursively for $\log M$ sweeps—each sweep utilizing half the processors of the previous one—resulting in a final $2n \times 2n$ compacted system. This small system is solved by a stable method, such as Gaussian elimination, to obtain y_0 and y_M, and then $\log M$ back-solve sweeps of the form (10) are used to recover the remaining unknowns. During the back-solve, the number of active processors doubles with each sweep.

Other variants of this algorithm exist. The need for $\log M$ back-solve sweeps can be eliminated by continuously using $M/2$ processors during the reduction. This results in a system of the form

$$\left[\begin{array}{ccccccc|c}
\mathcal{B}_a & & & & & & \mathcal{B}_b & g \\
V_0 & I & & & & & & \tilde{\phi}_0 \\
-V_1V_0 & & I & & & & & \tilde{\phi}_1 \\
V_2V_1V_0 & & & I & & & & \tilde{\phi}_2 \\
-V_3V_2V_1V_0 & & & & I & & & \tilde{\phi}_3 \\
\vdots & & & & & \ddots & & \vdots \\
(-1)^{M-1}\prod_{i=0}^{M-1} V_{M-1-i} & & & & & & I & \tilde{\phi}_{M-1}
\end{array}\right] \tag{12}$$

where $\tilde{\phi}_0 = \phi_0, \tilde{\phi}_i = \phi_i - V_i\tilde{\phi}_{i-1}, i = 1 \ldots M-1$. Then y_0 and y_M are computed by solving

$$\left[\begin{array}{cc}
\mathcal{B}_a & \mathcal{B}_b \\
(-1)^{M-1}\prod_{i=0}^{M-1} V_{M-1-i} & I
\end{array}\right]\left[\begin{array}{c} y_0 \\ y_M \end{array}\right] = \left[\begin{array}{c} g \\ \tilde{\phi}_{M-1} \end{array}\right] \tag{13}$$

the same $2n \times 2n$ compacted system that arises in the first version. The back-solve for (12) is completely parallel: the remaining unknowns can be computed in 1 step on M processors or 2 steps on $M/2$ processors.

If only $P < M/2$ processors are available, each processor can be assigned $M/2P$ pairs of block-rows to process in the first sweep, with the number of pairs of block-rows decreasing by a factor of 2 with each sweep until there are more processors than pairs of block-rows. Alternatively, the system can first be partitioned into P blocks of M/P block-rows. Each partition is assigned to a single processor, where it is reduced using a sequential algorithm. This results in a compacted system of order P that can be solved using cyclic reduction as shown above. Using this technique, $O(\log M)$ complexity is attainable with as few as $P = M/\log M$ processors. See [Asch 89] for more details.

Unfortunately, none of these algorithms is appropriate for solving ABD systems arising from BVPs. As pointed out in [Asch 89], they are equivalent to compactification—an algorithm that is known to be unstable because it fails to decouple the fast increasing and decreasing fundamental solution modes. In the following sections, stable variants of cyclic reduction are proposed for solving ABD systems.

3 Stable Local Factorization

The cyclic reduction algorithm described in §2 uses the elementary transformation

$$\begin{bmatrix} I & 0 \\ -V_{i+1} & I \end{bmatrix} \times \left[\begin{array}{ccccc|c} \cdots & V_i & I & & \cdots & \phi_i \\ \cdots & & V_{i+1} & I & \cdots & \phi_{i+1} \end{array}\right] = \left[\begin{array}{cccc|c} \cdots & V_i & I & \cdots & \phi_i \\ \cdots & -V_{i+1}V_i & I & \cdots & \phi_{i+1} - V_{i+1}\phi_i \end{array}\right] \tag{14}$$

The algorithm is potentially unstable if, for example, some eigenvalues of V_i are > 1 in magnitude, which is typical of ABD matrices arising from the discretization of BVPs—the larger eigenvalues correspond to increasing fundamental solution modes. As the reduction progresses, $\|\prod V_i\|$ can grow rapidly resulting in the loss of all significant digits to machine round-off.

In this section two alternative transformations are presented which can be adapted for use in a variation of the cyclic reduction algorithm. These *Stable Local Factorization* (*SLF*) transformations give rise to stable parallel algorithms for solving ABD systems. Each avoids the instability inherent in (14) by controlling the growth of elements during the reduction. One uses orthogonal factorization, the other Gaussian elimination with row partial-pivoting. Their stability is discussed in [Wrig 90b] and [Wrig 91a].

Since SLF-transformations do not exploit the identity matrix, the first form of the ABD system (7) is used throughout this section; the i-th slice appears as

$$\left[\begin{array}{ccccc|c} \cdots & S_i & T_i & & \cdots & \phi_i \\ \cdots & & S_{i+1} & T_{i+1} & \cdots & \phi_{i+1} \end{array}\right]$$

The notation $S_i^{(k)}$ is used to indicate that the matrix S_i has been transformed during the k-th step of an algorithm.

3.1 Orthogonal Factorization

This transformation requires the QR-factorization of part of the i-th slice:

$$\begin{bmatrix} T_i \\ S_{i+1} \end{bmatrix} = Q \begin{bmatrix} R_i \\ 0 \end{bmatrix}$$

where $Q \in \mathcal{R}^{2n\times 2n}$ is orthogonal and $R_i \in \mathcal{R}^{n\times n}$ is upper-triangular. Conceptually, once Q is obtained the i-th slice is multiplied through by Q^T giving

$$\left[\begin{array}{cccccc|c} \cdots & S_i^{(k)} & R_i^{(k)} & T_i^{(k)} & & \cdots & \phi_i^{(k)} \\ \cdots & S_{i+1}^{(k)} & & T_{i+1}^{(k)} & & \cdots & \phi_{i+1}^{(k)} \end{array}\right] \longleftarrow Q^T \times \left[\begin{array}{ccccc|c} \cdots & S_i & T_i & & \cdots & \phi_i \\ \cdots & & S_{i+1} & T_{i+1} & \cdots & \phi_{i+1} \end{array}\right] \quad (15)$$

However, in practice, the QR-factorization is computed in place with Householder reflections which are stored in the space formerly occupied by the elements below $R_i^{(k)}$. Thus, in the context of a modified Newton iteration (5), it is possible to perform several iterations (i.e. transform several subsequent right-hand sides) without recomputing the QR-factorization.

The SLF-QR transformation is structurally different than (14); in particular, there is fill-in at $T_i^{(k)}$. Therefore, the algorithms in §2 must be modified for use with SLF-QR. This is discussed further in §3.3 and §3.4.

3.2 Gaussian Elimination with Row Partial-Pivoting

This transformation is similar to SLF-QR, but instead uses an LU-factorization:

$$\begin{bmatrix} T_i \\ S_{i+1} \end{bmatrix} = L \begin{bmatrix} U_i \\ 0 \end{bmatrix}$$

where $L^{-1} = \tilde{L}_n P_n \cdots \tilde{L}_2 P_2 \tilde{L}_1 P_1 \in \mathcal{R}^{2n\times 2n}$, $\tilde{L}_j$ is an elementary Gauss transformation, P_j is a permutation matrix, and $U_i \in \mathcal{R}^{n\times n}$ is upper-triangular. Conceptually, once L^{-1} is obtained the i-th slice is multiplied through by L^{-1} giving

$$\left[\begin{array}{cccccc|c} \cdots & \tilde{S}_i^{(k)} & U_i^{(k)} & \tilde{T}_i^{(k)} & & \cdots & \tilde{\phi}_i^{(k)} \\ \cdots & \tilde{S}_{i+1}^{(k)} & & \tilde{T}_{i+1}^{(k)} & & \cdots & \tilde{\phi}_{i+1}^{(k)} \end{array}\right] \longleftarrow L^{-1} \times \left[\begin{array}{ccccc|c} \cdots & S_i & T_i & & \cdots & \phi_i \\ \cdots & & S_{i+1} & T_{i+1} & \cdots & \phi_{i+1} \end{array}\right] \quad (16)$$

However, in practice, the factorization is computed in place with Gauss transformations which are stored in the space formerly occupied by the elements below $U_i^{(k)}$, and the permutation matrices are stored in a single integer vector of length n. As for the SLF-QR factorization, several modified Newton iterations (5) can be computed using the same SLF-LU factorization.

SLF-LU is obviously structurally equivalent to SLF-QR. Either can be used with the algorithms in §3.3 and §3.4.

3.3 The SLF Partitioning Algorithm

SLF transformations can be incorporated in a partitioning algorithm which generalizes cyclic reduction for use on architectures with $P \leq M$ processors. The ABD system is first partitioned into P blocks of M/P block-rows each; the first partition appears as

$$\left[\begin{array}{cccccccc|c}
S_0 & T_0 & & & & & & & \phi_0 \\
 & S_1 & T_1 & & & & & & \phi_1 \\
 & & S_2 & T_2 & & & & & \phi_2 \\
 & & & S_3 & T_3 & & & & \phi_3 \\
 & & & & \ddots & \ddots & & & \vdots \\
 & & & & & S_{m_0} & T_{m_0} & & \phi_{m_0}
\end{array}\right] \tag{17}$$

The reduction steps shown below for the first partition can be carried out simultaneously on all P partitions. The reduction starts at the top and works down. In the first step a factorization is computed for $[T_0^T S_1^T]^T$ and (17) is transformed to

$$\left[\begin{array}{cccccc|c}
\boxed{S_0^{(1)}} & \boxed{R_0^{(1)}} & \boxed{T_0^{(1)}} & & & & \boxed{\phi_0^{(1)}} \\
\boxed{S_1^{(1)}} & & \boxed{T_1^{(1)}} & & & & \boxed{\phi_1^{(1)}} \\
 & & S_2 & T_2 & & & \phi_2 \\
 & & & S_3 & T_3 & & \phi_3 \\
 & & & \ddots & \ddots & & \vdots \\
 & & & & S_{m_0} & T_{m_0} & \phi_{m_0}
\end{array}\right] \tag{18}$$

In the second step $[T_1^{(1)T} S_2^T]^T$ is factored and (18) is transformed to

$$\left[\begin{array}{cccccc|c}
S_0^{(1)} & R_0^{(1)} & T_0^{(1)} & & & & \phi_0^{(1)} \\
\boxed{S_1^{(2)}} & & \boxed{R_1^{(2)}} & \boxed{T_1^{(2)}} & & & \boxed{\phi_1^{(2)}} \\
\boxed{S_2^{(2)}} & & & \boxed{T_2^{(2)}} & & & \boxed{\phi_2^{(2)}} \\
 & & & S_3 & T_3 & & \phi_3 \\
 & & & & \ddots & \ddots & \vdots \\
 & & & & & S_{m_0} \; T_{m_0} & \phi_{m_0}
\end{array}\right] \tag{19}$$

After m_0 steps the partition has the form

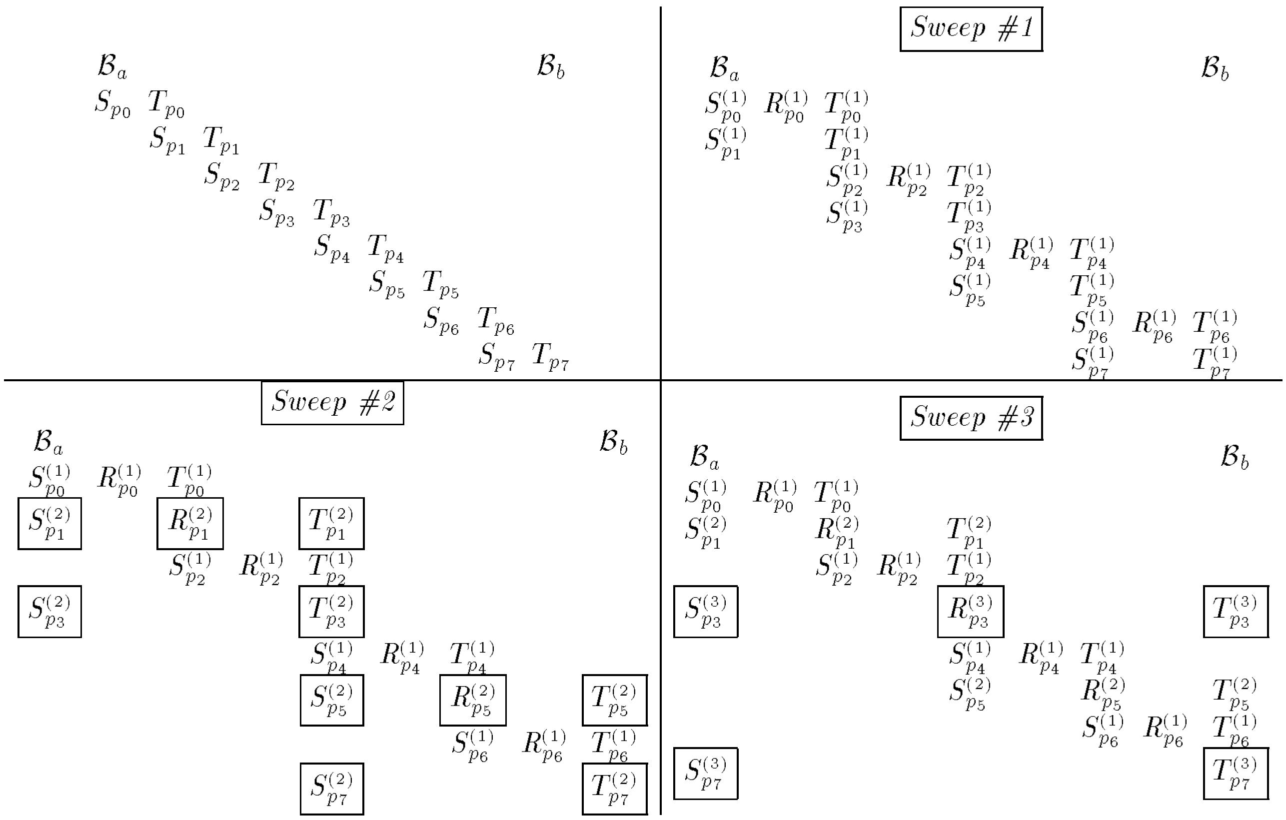
$$\left[\begin{array}{cccccccc|c}
S_0^{(1)} & R_0^{(1)} & T_0^{(1)} & & & & & & \phi_0^{(1)} \\
S_1^{(2)} & & R_1^{(2)} & T_1^{(2)} & & & & & \phi_1^{(2)} \\
S_2^{(3)} & & & R_2^{(3)} & T_2^{(3)} & & & & \phi_2^{(3)} \\
\vdots & & & & \ddots & \ddots & & & \vdots \\
\boxed{S_{m_0-1}^{(m_0)}} & & & & & \boxed{R_{m_0-1}^{(m_0)}} & \boxed{T_{m_0-1}^{(m_0)}} & & \boxed{\phi_{m_0-1}^{(m_0)}} \\
\boxed{S_{m_0}^{(m_0)}} & & & & & & \boxed{T_{m_0}^{(m_0)}} & & \boxed{\phi_{m_0}^{(m_0)}}
\end{array}\right] \tag{20}$$

Once y_0 and y_{m_0+1}, the end mesh points of this partition, are known, the interior points are computed by back-substitution; each step requires solving the upper-triangular system

$$R_{i-1}^{(i)} y_i = \phi_{i-1}^{(i)} - S_{i-1}^{(i)} y_0 - T_{i-1}^{(i)} y_{i+1}, \quad i = m_0, m_0 - 1, \ldots, 1. \tag{21}$$

As with the reduction, this can be done simultaneously on all P partitions.

Figure 1: Cyclic reduction using stable local factorization



The end mesh points for each partition are computed by solving a compacted system of order P constructed from the last block-row of each partition. Letting $[S_{m_i}^{(m_i)}, T_{m_i}^{(m_i)}, \phi_{m_i}^{(m_i)}] \equiv [S_{p_i}, T_{p_i}, \phi_{p_i}], i = 0 \dots P-1$, this compacted system has the form

$$\left[\begin{array}{cccccc|c} \mathcal{B}_a & & & & & \mathcal{B}_b & g \\ S_{p_0} & T_{p_0} & & & & & \phi_{p_0} \\ & S_{p_1} & T_{p_1} & & & & \phi_{p_1} \\ & & S_{p_2} & T_{p_2} & & & \phi_{p_2} \\ & & & \ddots & \ddots & & \vdots \\ & & & & S_{p_{P-1}} & T_{p_{P-1}} & \phi_{p_{P-1}} \end{array}\right] \tag{22}$$

which has the same structure as (7). It could therefore be solved in $O(P)$ block-steps using a stable sequential code, but since P processors are available it is possible to apply cyclic reduction to solve it in $O(\log P)$ block-steps. Two versions of cyclic reduction using SLF-transformations are described below. To simplify the figures, P is set to 8 and the right-hand sides are omitted. Generalizing the algorithms for more processors is straightforward.

The first version (Figure 1) requires $\log P$ reduction sweeps and $\log P$ back-solve sweeps. During the first reduction sweep 4 processors are active computing SLF-transformations simultaneously on block-rows 0-1, 2-3, 4-5, and 6-7. During the second sweep, 2 processors are active transforming block-rows 1-3 and 5-7, and during the final sweep 1 processor is active transforming block-rows 3-7. y_{p_0} and y_{p_7+1} are then computed by solving a $2n \times 2n$ system constructed from the first and last block-rows:

$$\begin{bmatrix} \mathcal{B}_a & \mathcal{B}_b \\ S^{(3)}_{p_7} & T^{(3)}_{p_7} \end{bmatrix} \begin{bmatrix} y_{p_0} \\ y_{p_7+1} \end{bmatrix} = \begin{bmatrix} g \\ \phi^{(3)}_{p_7} \end{bmatrix} \tag{23}$$

The back-solve works in a reverse fashion. During the first sweep 1 processor is active solving the triangular system in block-row 3 for y_{p_4}. During the second sweep 2 processors are active solving systems in block-rows 1 and 5 for y_{p_2} and y_{p_6}, and during the final sweep 4 processors are active solving systems in block-rows 0, 2, 4, and 6 for y_{p_1}, y_{p_3}, y_{p_5}, and y_{p_7}.

Several subsequent right-hand sides can be transformed during a modified Newton iteration (5) without recomputing the SLFs. If the Householder or Gauss transformations are stored as explained in §3.1 or §3.2, the reduction sweeps simply involve applying the stored transformations to the new right-hand side at a cost per block-row pair of $3n^2$ flops for Householder or $\frac{3}{2}n^2$ flops for Gauss.

In the second version (Figure 2) the need for $\log P$ back-solve sweeps is eliminated by making better use of processors during the reduction. Processors that were idle in Figure 1 are now used to "push" S and T blocks outwards resulting in an N-shaped Jacobian. Clearly, once y_{p_0} and y_{p_7+1} are known, the back-solve on this Jacobian can be accomplished in 1 sweep using P processors. The push begins at sweep #3 and requires an additional sweep at the end. The savings of a completely parallel back-solve, however, can easily outweigh the overhead of the additional reduction sweep when the Jacobian is kept fixed for several iterations in (5).

The usual SLF-transformation keeps $P/2^k$ processors active during sweep k. The S and T block pushes keep an additional $P - 4P/2^k$ processors active during sweeps $k = 3, \ldots, 1 + \log P$. Therefore, for this version,

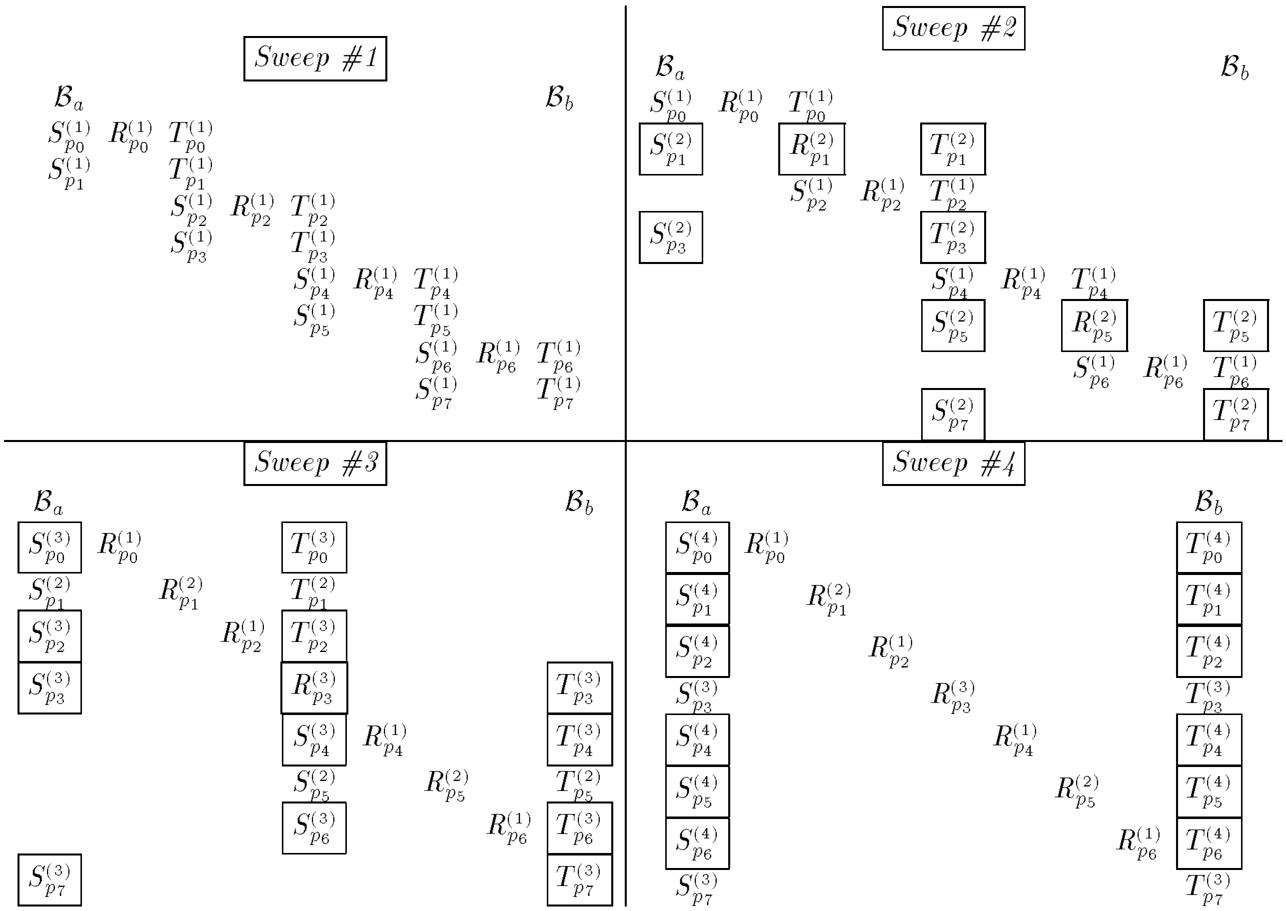
$$\text{total \# of active processors} = \begin{cases} P/2^k & 1 \le k \le 2 \\ P - 3P/2^k & 3 \le k \le \log P \\ P - 2 & k = 1 + \log P. \end{cases}$$

Thus, in contrast to the first version, the number of active processors *increases* during the reduction to a maximum of $P - 2$.

In order for the second version to be efficient each processor must push its blocks in the same time required for the SLF-transformation (or less). For example, in sweep #3 of Figure 2 one processor computes the usual SLF-transformation on block-rows 3-7. Simultaneously, 4 other processors push S and T blocks outwards in rows 0, 2, 4, and 6. As shown below for block-row 0, a "push" requires 4 steps:

1. $V = (R^{(2)}_{p_1})^{-1} S^{(2)}_{p_1}$
2. $W = (R^{(2)}_{p_1})^{-1} T^{(2)}_{p_1}$
3. $S^{(3)}_{p_0} = S^{(1)}_{p_0} - T^{(1)}_{p_0} V$
4. $T^{(3)}_{p_0} = -T^{(1)}_{p_0} W$

Figure 2: Cyclic reduction to N-shape Jacobian using stable local factorization



Since $R_{p_1}^{(2)}$ is upper-triangular, steps 1 and 2 require $n^3/2$ flops each. The total for all four steps is therefore $3n^3 + O(n^2)$ which, fortunately, is less than the flop count for the fastest SLF-transformation (see §5).

One possible way of optimizing this process stems from the observation that often in a BVP the boundary conditions have little influence on the solution at interior mesh points. As a result, in many problems the interior S and T blocks "vanish" long before they reach the left/right boundaries. Numerical experiments show that these blocks can be dropped when their norm falls below a threshold tolerance (close to machine epsilon) without affecting the solution. Of course the overhead of monitoring the norm must be taken into consideration when evaluating the cost effectiveness of such a scheme.

3.4 An Alternative Partitioning Algorithm

Instead of applying SLF transformations as shown in (17)-(20), cyclic reduction can be used in a sequential fashion on each partition. To illustrate, let $m_0 = 7$ in (17), so the first partition appears as

$$\left[\begin{array}{cccccccc|c} S_0 & T_0 & & & & & & & \phi_0 \\ & S_1 & T_1 & & & & & & \phi_1 \\ & & S_2 & T_2 & & & & & \phi_2 \\ & & & S_3 & T_3 & & & & \phi_3 \\ & & & & S_4 & T_4 & & & \phi_4 \\ & & & & & S_5 & T_5 & & \phi_5 \\ & & & & & & S_6 & T_6 & \phi_6 \\ & & & & & & & S_7 \quad T_7 & \phi_7 \end{array}\right] \tag{24}$$

Since only one processor is available for this partition, the reduction requires 7 steps as opposed to 3 sweeps in Figure 1. After 4 steps (24) is transformed to

$$\left[\begin{array}{ccccccccc|c} S_0^{(1)} & R_0^{(1)} & T_0^{(1)} & & & & & & & \phi_0^{(1)} \\ S_1^{(1)} & & T_1^{(1)} & & & & & & & \phi_1^{(1)} \\ & & S_2^{(2)} & R_2^{(2)} & T_2^{(2)} & & & & & \phi_2^{(2)} \\ & & S_3^{(2)} & & T_3^{(2)} & & & & & \phi_3^{(2)} \\ & & & & S_4^{(3)} & R_4^{(3)} & T_4^{(3)} & & & \phi_4^{(3)} \\ & & & & S_5^{(3)} & & T_5^{(3)} & & & \phi_5^{(3)} \\ & & & & & & S_6^{(4)} & R_6^{(4)} & T_6^{(4)} & \phi_6^{(4)} \\ & & & & & & S_7^{(4)} & & T_7^{(4)} & \phi_7^{(4)} \end{array}\right] \tag{25}$$

After 2 more steps (25) is transformed to

$$\left[\begin{array}{ccccccccc|c}
S_0^{(1)} & R_0^{(1)} & T_0^{(1)} & & & & & & & \phi_0^{(1)} \\
\boxed{S_1^{(5)}} & & \boxed{R_1^{(5)}} & & \boxed{T_1^{(5)}} & & & & & \boxed{\phi_1^{(5)}} \\
& & S_2^{(2)} & R_2^{(2)} & T_2^{(2)} & & & & & \phi_2^{(2)} \\
\boxed{S_3^{(5)}} & & & & \boxed{T_3^{(5)}} & & & & & \boxed{\phi_3^{(5)}} \\
& & & & S_4^{(3)} & R_4^{(3)} & T_4^{(3)} & & & \phi_4^{(3)} \\
& & & & \boxed{S_5^{(6)}} & & \boxed{R_5^{(6)}} & & \boxed{T_5^{(6)}} & \boxed{\phi_5^{(6)}} \\
& & & & & & S_6^{(4)} & R_6^{(4)} & T_6^{(4)} & \phi_6^{(4)} \\
& & & & \boxed{S_7^{(6)}} & & & & \boxed{T_7^{(6)}} & \boxed{\phi_7^{(6)}}
\end{array}\right] \qquad (26)$$

After the final step the partition has the form

$$\left[\begin{array}{ccccccccc|c}
S_0^{(1)} & R_0^{(1)} & T_0^{(1)} & & & & & & & \phi_0^{(1)} \\
S_1^{(5)} & & R_1^{(5)} & & T_1^{(5)} & & & & & \phi_1^{(5)} \\
& & S_2^{(2)} & R_2^{(2)} & T_2^{(2)} & & & & & \phi_2^{(2)} \\
\boxed{S_3^{(7)}} & & & & \boxed{R_3^{(7)}} & & & & \boxed{T_3^{(7)}} & \boxed{\phi_3^{(7)}} \\
& & & & S_4^{(3)} & R_4^{(3)} & T_4^{(3)} & & & \phi_4^{(3)} \\
& & & & S_5^{(6)} & & R_5^{(6)} & & T_5^{(6)} & \phi_5^{(6)} \\
& & & & & & S_6^{(4)} & R_6^{(4)} & T_6^{(4)} & \phi_6^{(4)} \\
\boxed{S_7^{(7)}} & & & & & & & & \boxed{T_7^{(7)}} & \boxed{\phi_7^{(7)}}
\end{array}\right] \qquad (27)$$

The end mesh points of this partition, y_0 and y_8, are computed by solving a compacted system constructed from the last block-row of each partition as before (Figures 1 and 2). Once the end points are known, the interior points are computed in 7 back-substitution steps similar to (21), except that here the sequence is block-row 3,1,5,0,2,4,6.

The cost of applying cyclic reduction to each partition is the same as that of the algorithm in §3.3. Which algorithm is better in terms of stability, however, is currently an open question. As pointed out in [Wrig 91b], SLF-LU when implemented as described in §3.3 is potentially unstable on certain problems when the partitions become large. Whether or not cyclic reduction is a more stable alternative for these problems will be investigated in [Panc 92].

4 Global Stability Control

As shown in the previous section, SLF-transformations can be used with partitioning and cyclic reduction to provide a stable algorithm for solving ABD systems. With as few as $M/\log M$ processors, a parallel complexity of $O(\log M)$ block-steps is attainable. Theoretically this is the optimal speed-up with respect to M. Nevertheless, this algorithm can still be improved by reducing the local operation counts (with respect to n) and storage requirements. In this section a new algorithm based on *eigenvalue rescaling* is proposed which requires approximately half the local execution time and half the over-all storage as the fastest SLF algorithm.

4.1 Improving SLF-Transformations

Three ways in which SLF-transformations differ from the elementary transformation used in block cyclic reduction (14) are:

1. the $n \times n$ block $T_i^{(k)}$ in (15) and (16) fills-in requiring extra storage and resulting in a more costly backsolve,
2. a $2n \times n$ matrix factorization followed by four $n \times n$ matrix products is required, as opposed to a single $n \times n$ matrix product, and
3. the identity matrix that appears in the second form of the Jacobian (8) is destroyed, resulting in additional fill-in if the second form arises naturally from the discretization.

Indeed, the first two differences also hold when comparing SLF to state-of-the-art sequential codes such as COLROW, except that the n^3 local operation in COLROW is not due to matrix multiplication.

So the advantages of block cyclic reduction are obvious—it is faster and requires less storage. The main drawback, of course, is that it is not necessarily stable when used to solve ABD systems arising from BVPs. The algorithms in §2 *are* stable, though, when used to solve certain classes of linear recurrence relations. In particular, consider the two-term recurrence

$$y_0 = \alpha, \quad y_{i+1} = V_i y_i + \phi_i \quad i = 0, 1, \ldots \tag{28}$$

where $\|V_i\|_2 \leq 1$ for all i. This recurrence is stable and can be computed in parallel using cyclic reduction ([Sche 84]). In fact, this is precisely the recurrence that arises when solving a stable linear IVP. As noted in [Gear 88], fast methods for the solution of linear IVPs can be constructed from fast algorithms for solving (28).

Now, if the ABD system (8) could be transformed and the equations recast as (28), the analysis in [Sche 84] and [Gear 88] would apply and block cyclic reduction could be used on this problem. In order to be competitive with SLF the transformation cannot be expensive, and it should not have an adverse effect on the condition of the system (c.f. the normal equations method in [Asch 89]). The rescaling algorithm described below is based on such a transformation. Since the mesh is changed globally during the reduction, the technique is said to use *global stability control.*

4.2 Sequential Rescaling

This form of the algorithm uses a sequential mesh transformation to produce an ABD system that can be solved stably using block cyclic reduction. Instead of solving $\mathcal{J}Y = \Phi$ for Y, $(\mathcal{J}C)\tilde{Y} = \Phi$ is solved for $\tilde{Y} = C^{-1}Y$, where

$$C = \begin{bmatrix} I & -\sigma I & & & & \\ & I & -\sigma I & & & \\ & & I & -\sigma I & & \\ & & & \ddots & \ddots & \\ & & & & I & -\sigma I \\ & & & & & I \end{bmatrix}, \quad C^{-1} = \begin{bmatrix} I & \sigma I & \sigma^2 I & \sigma^3 I & \cdots & \sigma^M I \\ & I & \sigma I & \sigma^2 I & \cdots & \sigma^{M-1} I \\ & & I & \sigma I & \cdots & \sigma^{M-2} I \\ & & & \ddots & \ddots & \vdots \\ & & & & I & \sigma I \\ & & & & & I \end{bmatrix} \tag{29}$$

For ease of notation, $\sigma = 1$ is assumed throughout most of this section. The optimal choice of σ is currently an open question. Numerical experiments show $\sigma = 1$ is appropriate for most test problems. Varying σ, though, can affect the accuracy of the solution, as is discussed further in §6.

Once $\tilde{Y}$ has been computed, the solution on the original mesh can be recovered easily:

$$y_i = \tilde{y}_i - \tilde{y}_{i+1}, \quad i = 0, \ldots, M-2. \tag{30}$$

The algorithm proceeds as follows—first, the system is transformed (the right-hand side is not affected):

$$\left[\begin{bmatrix} \mathcal{B}_a & & & & & & \mathcal{B}_b \\ V_0 & I & & & & & \\ & V_1 & I & & & & \\ & & \ddots & \ddots & & & \\ & & & V_{M-3} & I & & \\ & & & & V_{M-2} & I & \\ & & & & & V_{M-1} & I \end{bmatrix} \times \begin{bmatrix} I & -I & & & & & \\ & I & -I & & & & \\ & & I & -I & & & \\ & & & \ddots & \ddots & & \\ & & & & I & -I & \\ & & & & & I & -I \\ & & & & & & I \end{bmatrix} \left| \begin{matrix} g \\ \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{M-3} \\ \phi_{M-2} \\ \phi_{M-1} \end{matrix} \right. \right]$$

$$= \left[\begin{array}{cccccccc|c} \mathcal{B}_a & -\mathcal{B}_a & & & & & & \mathcal{B}_b & g \\ V_0 & I-V_0 & -I & & & & & & \phi_0 \\ & V_1 & I-V_1 & -I & & & & & \phi_1 \\ & & \ddots & \ddots & \ddots & & & & \vdots \\ & & & V_{M-3} & I-V_{M-3} & -I & & & \phi_{M-3} \\ & & & & V_{M-2} & I-V_{M-2} & -I & & \phi_{M-2} \\ & & & & & V_{M-1} & I-V_{M-1} & & \phi_{M-1} \end{array}\right] \tag{31}$$

This requires no extra storage and only minimal computation in the actual implementation due to the simple form of C. Now (31) is no longer ABD, but it can be transformed back by a sequential process starting at the bottom block-row and working up. First, block-row $M-1$ is multiplied through by $(I - V_{M-1})^{-1}$ giving

$$\left[\begin{array}{cccccccc|c} \mathcal{B}_a & -\mathcal{B}_a & & & & & & \mathcal{B}_b & g \\ V_0 & I-V_0 & -I & & & & & & \phi_0 \\ & V_1 & I-V_1 & -I & & & & & \phi_1 \\ & & \ddots & \ddots & \ddots & & & & \vdots \\ & & & V_{M-3} & I-V_{M-3} & -I & & & \phi_{M-3} \\ & & & & V_{M-2} & I-V_{M-2} & -I & & \phi_{M-2} \\ & & & & & \boxed{V_{M-1}^{(1)}} & \boxed{I} & & \boxed{\phi_{M-1}^{(1)}} \end{array}\right] \tag{32}$$

Next, block-row $M-1$ is added to block-row $M-2$ to annihilate the $-I$ in block-row $M-2$.

Then block-row $M-2$ is multiplied through by $(I - V_{M-2} + V_{M-1}^{(1)})^{-1}$ giving

$$\left[\begin{array}{cccccccc|c} \mathcal{B}_a & -\mathcal{B}_a & & & & & & \mathcal{B}_b & g \\ V_0 & I{-}V_0 & -I & & & & & & \phi_0 \\ & V_1 & I{-}V_1 & -I & & & & & \phi_1 \\ & & \ddots & \ddots & \ddots & & & & \vdots \\ & & & V_{M-3} & I{-}V_{M-3} & -I & & & \phi_{M-3} \\ & & & & \boxed{V_{M-2}^{(2)}} & \boxed{I} & & & \boxed{\phi_{M-2}^{(2)}} \\ & & & & & V_{M-1}^{(1)} & I & & \phi_{M-1}^{(1)} \end{array}\right] \tag{33}$$

Block-rows $M-3, \ldots, 0$ are transformed similarily giving

$$\left[\begin{array}{ccccccc|c} \mathcal{B}_a & -\mathcal{B}_a & & & & & \mathcal{B}_b & g \\ \boxed{V_0^{(M)}} & \boxed{I} & & & & & & \boxed{\phi_0^{(M)}} \\ & V_1^{(M-1)} & I & & & & & \phi_1^{(M-1)} \\ & & \ddots & \ddots & & & & \vdots \\ & & & V_{M-3}^{(3)} & I & & & \phi_{M-3}^{(3)} \\ & & & & V_{M-2}^{(2)} & I & & \phi_{M-2}^{(2)} \\ & & & & & V_{M-1}^{(1)} & I & \phi_{M-1}^{(1)} \end{array}\right] \tag{34}$$

Finally, $-\mathcal{B}_a$ in the top block-row is annihilated:

$$\left[\begin{array}{ccccccc|c} \boxed{\mathcal{B}_a^{(M+1)}} & & & & & & \mathcal{B}_b & \boxed{g^{(M+1)}} \\ V_0^{(M)} & I & & & & & & \phi_0^{(M)} \\ & V_1^{(M-1)} & I & & & & & \phi_1^{(M-1)} \\ & & \ddots & \ddots & & & & \vdots \\ & & & V_{M-3}^{(3)} & I & & & \phi_{M-3}^{(3)} \\ & & & & V_{M-2}^{(2)} & I & & \phi_{M-2}^{(2)} \\ & & & & & V_{M-1}^{(1)} & I & \phi_{M-1}^{(1)} \end{array}\right] \tag{35}$$

This last system is once again ABD. More importantly, numerical experiments and preliminary analysis indicate that it can be solved stably using the algorithms in §2 because, for well-conditioned problems, the above transformation effectively rescales the eigenvalues λ and the norm of each V_i so that

1. $|\lambda| \in (0, 1+\epsilon_1)$, and
2. $\|V_i\|_2 \in (1-\epsilon_2, 1+\epsilon_3)$

where ϵ_1, ϵ_2, and ϵ_3 can be made arbitrarily small with a suitable choice for σ in (29). For the test problems considered in §6, $\sigma = 1$ is sufficient for stability. In addition, even though theoretically the transformation may fail for any given σ if $(I - \sigma(V_i + V_{i+1}^{(k)}))$ becomes numerically singular for some i, numerical experiments have shown that a dynamic shift in σ at this stage redistributes the eigenvalues and allows the algorithm to continue. At the time of writing, the only ABDs known to require this dynamic shift are themselves very poorly conditioned. These issues will be investigated further in [Panc 92].

The total flop-count for this algorithm—including operations required for the transformation, cyclic reduction, back-solve and mesh recovery (30)—is approximately 1/2 that of SLF-LU. Multiple right-hand sides can be handled in $O(Mn^2)$ time if the local factorizations of $(I - \sigma(V_i + V_{i+1}^{(k)}))$ are stored. In fact, once the Jacobian has been reduced, the time required to solve with a different right-hand side is comparable to that of algorithms based on alternate row and column pivoting. See §5 for details.

Rescaling requires $2Mn^2$ storage if the local factorizations are kept—the same as COLROW and 1/2 that of SLF-based algorithms. Since rescaling is indifferent to the separability of the boundary conditions, systems with coupled BCs can be solved without fill-in.

4.3 Parallel Rescaling

Although (35) can be solved in parallel, constructing it as shown in (32)-(35) is a sequential process. This section explains how the rescaling transformation can be integrated with partitioning and cyclic reduction to provide a parallel rescaling algorithm.

As in §3.3, the ABD system is first partitioned into P blocks of M/P block-rows each. The transformation C is modified slightly by omitting the $-I$ at the boundary between each partition. For example, the transformation across the first partition has the form

$$\left[\begin{bmatrix} \ddots & \ddots & & & & & \\ & V_{m_0-2} & I & & & & \\ & & V_{m_0-1} & I & & & \\ & & & V_{m_0} & I & & \\ & & & & V_{m_0+1} & I & \\ & & & & & V_{m_0+2} & I \\ & & & & & & \ddots \;\; \ddots \end{bmatrix} \times \begin{bmatrix} \ddots & \ddots & & & & & \\ & I & -I & & & & \\ & & I & -I & & & \\ & & & I & & & \\ & & & & I & -I & \\ & & & & & I & -I \\ & & & & & & \ddots \;\; \ddots \end{bmatrix} \left| \begin{matrix} \vdots \\ \phi_{m_0-2} \\ \phi_{m_0-1} \\ \phi_{m_0} \\ \phi_{m_0+1} \\ \phi_{m_0+2} \\ \vdots \end{matrix} \right. \right]$$

$$= \left[\begin{array}{cccccccc|c} \ddots & \ddots & \ddots & & & & & & \vdots \\ & V_{m_0-2} & I - V_{m_0-2} & -I & & & & & \phi_{m_0-2} \\ & & V_{m_0-1} & I - V_{m_0-1} & & & & & \phi_{m_0-1} \\ & & & V_{m_0} & I & -I & & & \phi_{m_0} \\ & & & & V_{m_0+1} & I - V_{m_0+1} & -I & & \phi_{m_0+1} \\ & & & & & V_{m_0+2} & I - V_{m_0+2} & -I & \phi_{m_0+2} \\ & & & & & \ddots & \ddots & \ddots & \vdots \end{array} \right] \tag{36}$$

The steps shown below on the first partition can be carried out simultaneously on all P partitions. Processing starts at block-row $m_0 - 1$ and works up. The partition is changed back to block-diagonal form as in the sequential algorithm, but now V_{m_0} is "dragged" to the left as well; i.e. the reduction phase is interleaved with the rescaling. (This could also be done in the sequential algorithm.) So the first step consists of

1. $V_{m_0-1}^{(1)} = (I - V_{m_0-1})^{-1} V_{m_0-1}$, $\phi_{m_0-1}^{(1)} = (I - V_{m_0-1})^{-1} \phi_{m_0-1}$,

2. $V_{m_0}^{(1)} = -V_{m_0} V_{m_0-1}^{(1)}$, $\phi_{m_0}^{(1)} = \phi_{m_0} - V_{m_0} \phi_{m_0-1}^{(1)}$.

After the first step the partition is

$$\left[\begin{array}{ccccccc|c}
\ddots & \ddots & \ddots & & & & & \vdots \\
V_{m_0-2} & I-V_{m_0-2} & -I & & & & & \phi_{m_0-2} \\
 & \boxed{V^{(1)}_{m_0-1}} & \boxed{I} & & & & & \boxed{\phi^{(1)}_{m_0-1}} \\
 & \boxed{V^{(1)}_{m_0}} & & I & -I & & & \boxed{\phi^{(1)}_{m_0}} \\
 & & & V_{m_0+1} & I-V_{m_0+1} & -I & & \phi_{m_0+1} \\
 & & & & V_{m_0+2} & I-V_{m_0+2} & -I & \phi_{m_0+2} \\
 & & & & & \ddots & \ddots\;\ddots & \vdots
\end{array}\right] \tag{37}$$

Next, the $-I$ in block-row m_0-2 is annihilated, block-row m_0-2 is multiplied through by $(I - V_{m_0-2} + V^{(1)}_{m_0-1})^{-1}$, and $V^{(1)}_{m_0}$ is dragged one block-column to the left:

$$\left[\begin{array}{ccccccc|c}
\ddots & \ddots & \ddots & & & & & \vdots \\
\boxed{V^{(2)}_{m_0-2}} & \boxed{I} & & & & & & \boxed{\phi^{(2)}_{m_0-2}} \\
 & V^{(1)}_{m_0-1} & I & & & & & \phi^{(1)}_{m_0-1} \\
\boxed{V^{(2)}_{m_0}} & & & I & -I & & & \boxed{\phi^{(2)}_{m_0}} \\
 & & & V_{m_0+1} & I-V_{m_0+1} & -I & & \phi_{m_0+1} \\
 & & & & V_{m_0+2} & I-V_{m_0+2} & -I & \phi_{m_0+2} \\
 & & & & & \ddots & \ddots\;\ddots & \vdots
\end{array}\right] \tag{38}$$

Block-rows $m_0-3, \ldots, 0$ are transformed similarily. Assuming $m_0 = m_1$, the second partition will have been transformed by this time as well:

$$\left[\begin{array}{ccccccc|c}
\mathcal{B}_a & -\mathcal{B}_a & & & & & & g \\
\boxed{V_0^{(m_0)}} & \boxed{I} & & & & & & \boxed{\phi_0^{(m_0)}} \\
 & \ddots & \ddots & & & & & \vdots \\
 & & V^{(1)}_{m_0-1} & I & & & & \phi^{(1)}_{m_0-1} \\
\boxed{V^{(m_0)}_{m_0}} & & & & I & -I & & \boxed{\phi^{(m_0)}_{m_0}} \\
 & & & & \boxed{V^{(m_1)}_{m_0+1}} & \boxed{I} & & \boxed{\phi^{(m_1)}_{m_0+1}} \\
 & & & & & V^{(m_1-1)}_{m_0+2} & I & \phi^{(m_1-1)}_{m_0+2} \\
 & & & & & & \ddots\;\ddots & \vdots
\end{array}\right] \tag{39}$$

Finally, $-\mathcal{B}_a$ in the top block-row is annihilated, as is the $-I$ appearing in the bottom block-row of each partition (except the last):

$$\left[\begin{array}{ccccccc|c}
\boxed{\mathcal{B}_a^{(m_0+1)}} & & & & & & & \boxed{g^{(m_0+1)}} \\
V_0^{(m_0)} & I & & & & & & \phi_0^{(m_0)} \\
 & \ddots & \ddots & & & & & \vdots \\
 & & V^{(1)}_{m_0-1} & I & & & & \phi^{(1)}_{m_0-1} \\
V^{(m_0)}_{m_0} & & & & \boxed{W^{(m_1+1)}_{m_0}} & & & \boxed{\phi^{(m_1+1)}_{m_0}} \\
 & & & & V^{(m_1)}_{m_0+1} & I & & \phi^{(m_1)}_{m_0+1} \\
 & & & & & V^{(m_1-1)}_{m_0+2} & I & \phi^{(m_1-1)}_{m_0+2} \\
 & & & & & & \ddots\;\ddots & \vdots
\end{array}\right] \tag{40}$$

where $W_{m_0}^{(m_1+1)} = I + V_{m_0+1}^{(m_1)}$, $\phi_{m_0}^{(m_1+1)} = \phi_{m_0}^{(m_0)} + \phi_{m_0+1}^{(m_1)}$.

$\tilde{y}_0$ and $\tilde{y}_{m_0+1}$ are the end mesh points of the first partition. Once they are known, the interior points can be computed by a forward recurrence and the original mesh can be restored:

$$\tilde{y}_{i+1} = \phi_i^{(k)} - V_i^{(k)}\tilde{y}_i, \quad y_i = \tilde{y}_i - \tilde{y}_{i+1}, \quad i = 0, \ldots, m_0 - 1 \tag{41}$$

$$y_{m_0} = \tilde{y}_{m_0} - \tilde{y}_{m_0+1}, \quad y_{m_0+1} = \tilde{y}_{m_0+1}$$

As with (35), numerical experiments show that the eigenvalues and norm of each $V_i^{(k)}$ are rescaled by the above transformation allowing (41) to be computed stably.

The end mesh points for each partition are computed by solving a compacted system of order P constructed from the last block-row of each partition. Letting

$$[\mathcal{B}_a^{(m_0+1)} g^{(m_0+1)}] \equiv [\hat{\mathcal{B}}_a \; \hat{g}], \quad [V_{m_i}^{(m_i)} W_{m_i}^{(m_i+1)} \phi_{m_i}^{(m_i+1)}] \equiv [V_{p_i} \; W_{p_i} \; \phi_{p_i}], \quad i = 0 \ldots P-1,$$

this compacted system has the form

$$\left[\begin{array}{cccccc|c}
\hat{\mathcal{B}}_a & & & & & \mathcal{B}_b & \hat{g} \\
V_{p_0} & W_{p_0} & & & & & \phi_{p_0} \\
& V_{p_1} & W_{p_1} & & & & \phi_{p_1} \\
& & \ddots & \ddots & & & \vdots \\
& & & V_{p_{P-2}} & W_{p_{P-2}} & & \phi_{p_{P-2}} \\
& & & & V_{p_{P-1}} & I & \phi_{p_{P-1}}
\end{array}\right] \tag{42}$$

This system has a slightly different structure than (8)—the identity is replaced by W_{p_i} in rows $0 \ldots P-2$. Unlike (7), (42) *cannot* be transformed by multiplying each block-row through by $W_{p_i}^{-1}$, because W_{p_i} is not necessarily well-conditioned.[1] Nevertheless, mesh transformation can still be used with cyclic reduction to solve this system in $O(\log P)$ block-steps. Two versions of the algorithm are described below (again, $P = 8$ is used to simplify the figures).

The first version requires $\log P$ reduction sweeps and $\log P$ back-solve sweeps. During the first reduction sweep, (42) is transformed as follows:

$$\left[\begin{array}{c|c}
\begin{bmatrix}
\hat{\mathcal{B}}_a & & & & & & & \mathcal{B}_b \\
V_{p_0} & W_{p_0} & & & & & & \\
& V_{p_1} & W_{p_1} & & & & & \\
& & V_{p_2} & W_{p_2} & & & & \\
& & & V_{p_3} & W_{p_3} & & & \\
& & & & V_{p_4} & W_{p_4} & & \\
& & & & & V_{p_5} & W_{p_5} & \\
& & & & & & V_{p_6} & W_{p_6} \\
& & & & & & & V_{p_7} \quad I
\end{bmatrix}
\times
\begin{bmatrix}
I & -I & & & & & & \\
& I & & & & & & \\
& & I & -I & & & & \\
& & & I & & & & \\
& & & & I & -I & & \\
& & & & & I & & \\
& & & & & & I & -I \\
& & & & & & & I \\
& & & & & & & & I
\end{bmatrix}
&
\begin{matrix} \hat{g} \\ \phi_{p_0} \\ \phi_{p_1} \\ \phi_{p_2} \\ \phi_{p_3} \\ \phi_{p_4} \\ \phi_{p_5} \\ \phi_{p_6} \\ \phi_{p_7} \end{matrix}
\end{array}\right]$$

[1] The same is true for T_{p_i} in (22).

$$
= \left[\begin{array}{ccccccccc|c}
\hat{\mathcal{B}}_a & -\hat{\mathcal{B}}_a & & & & & & & \mathcal{B}_b & \hat{g} \\
V_{p0} & W_{p0}-V_{p0} & & & & & & & & \phi_{p0} \\
 & V_{p1} & W_{p1} & -W_{p1} & & & & & & \phi_{p1} \\
 & & V_{p2} & W_{p2}-V_{p2} & & & & & & \phi_{p2} \\
 & & & V_{p3} & W_{p3} & -W_{p3} & & & & \phi_{p3} \\
 & & & & V_{p4} & W_{p4}-V_{p4} & & & & \phi_{p4} \\
 & & & & & V_{p5} & W_{p5} & -W_{p5} & & \phi_{p5} \\
 & & & & & & V_{p6} & W_{p6}-V_{p6} & & \phi_{p6} \\
 & & & & & & & V_{p7} & I & \phi_{p7}
\end{array}\right] \tag{43}
$$

This transformation *reconditions* $(W_{p_i} - V_{p_i}), i = 0, 2, 4, 6$, so these block-rows can now be multiplied through by $(W_{p_i} - V_{p_i})^{-1}$. For $i = 0, 2, 4, 6$, each of 4 processors is active during the first sweep computing

1. $V_{p_i}^{(1)} = (W_{p_i} - V_{p_i})^{-1} V_{p_i}, \quad \phi_{p_i}^{(1)} = (W_{p_i} - V_{p_i})^{-1} \phi_{p_i},$
2. $V_{p_{i+1}}^{(1)} = -V_{p_{i+1}} V_{p_i}^{(1)}, \quad \phi_{p_{i+1}}^{(1)} = \phi_{p_{i+1}} - V_{p_{i+1}} \phi_{p_i}^{(1)}.$

In addition, $-\hat{\mathcal{B}}_a$ and $-W_{p_i}, i = 1, 3, 5$, are annihilated, resulting in a system of the form

$$
\left[\begin{array}{ccccccccc|c}
\hat{\mathcal{B}}_a^{(1)} & & & & & & & & \mathcal{B}_b & \hat{g}^{(1)} \\
V_{p0}^{(1)} & I & & & & & & & & \phi_{p0}^{(1)} \\
V_{p1}^{(1)} & & W_{p1}^{(1)} & & & & & & & \phi_{p1}^{(1)} \\
 & & V_{p2}^{(1)} & I & & & & & & \phi_{p2}^{(1)} \\
 & & V_{p3}^{(1)} & & W_{p3}^{(1)} & & & & & \phi_{p3}^{(1)} \\
 & & & & V_{p4}^{(1)} & I & & & & \phi_{p4}^{(1)} \\
 & & & & V_{p5}^{(1)} & & W_{p5}^{(1)} & & & \phi_{p5}^{(1)} \\
 & & & & & & V_{p6}^{(1)} & I & & \phi_{p6}^{(1)} \\
 & & & & & & V_{p7}^{(1)} & & I & \phi_{p7}^{(1)}
\end{array}\right] \tag{44}
$$

This uncouples the odd-indexed variables from the system. Once $\tilde{y}_{p_i}, i = 0, 2, 4, 6, 8$ are known, $\tilde{y}_{p_{i+1}}, i = 0, 2, 4, 6$ can be computed and the transformation in (43) reversed by

$$
\tilde{y}_{p_{i+1}} = \phi_{p_i}^{(1)} - V_{p_i}^{(1)} \tilde{y}_{p_i}, \quad y_{p_i} = \tilde{y}_{p_i} - \tilde{y}_{p_{i+1}}, \quad y_{p_{i+1}} = \tilde{y}_{p_{i+1}}, \quad i = 0, 2, 4, 6. \tag{45}
$$

The even-indexed variables are computed during the second sweep by solving a reduced system constructed from the odd rows and columns of (44). This system together with the transformation that is applied during the second sweep is shown below:

$$
\left[\begin{array}{c|c}
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(1)} & & & & \mathcal{B}_b \\
V_{p1}^{(1)} & W_{p1}^{(1)} & & & \\
 & V_{p3}^{(1)} & W_{p3}^{(1)} & & \\
 & & V_{p5}^{(1)} & W_{p5}^{(1)} & \\
 & & & V_{p7}^{(1)} & I
\end{bmatrix}
\times
\begin{bmatrix}
I & -I & & & \\
 & I & & & \\
 & & I & -I & \\
 & & & I & \\
 & & & & I
\end{bmatrix}
&
\begin{array}{c}
\hat{g}^{(1)} \\ \phi_{p1}^{(1)} \\ \phi_{p3}^{(1)} \\ \phi_{p5}^{(1)} \\ \phi_{p7}^{(1)}
\end{array}
\end{array}\right]
$$

$$
= \left[\begin{array}{ccccc|c}
\hat{\mathcal{B}}_a^{(1)} & -\hat{\mathcal{B}}_a^{(1)} & & & \mathcal{B}_b & \hat{g}^{(1)} \\
V_{p_1}^{(1)} & W_{p_1}^{(1)} - V_{p_1}^{(1)} & & & & \phi_{p_1}^{(1)} \\
 & V_{p_3}^{(1)} & W_{p_3}^{(1)} & -W_{p_3}^{(1)} & & \phi_{p_3}^{(1)} \\
 & & V_{p_5}^{(1)} & W_{p_5}^{(1)} - V_{p_5}^{(1)} & & \phi_{p_5}^{(1)} \\
 & & & V_{p_7}^{(1)} & I & \phi_{p_7}^{(1)}
\end{array}\right] \tag{46}
$$

The algorithm continues recursively for $\log P$ sweeps resulting in a final $2n \times 2n$ compacted system. This small system is solved by a stable method, such as Gaussian elimination, followed by $\log P$ back-solve/transformation reversal sweeps of the form (45).

In the second version (Figures 3 and 4) the need for $\log P$ back-solve sweeps is eliminated by continuously utilizing P processors during the reduction. In constrast to the first version where every other block-row and column is skipped, the complete system is now considered at each sweep. As shown in Figure 3, the reconditioning transformation creates *upward*-pointing spikes, which can be annihilated concurrently as the downward-pointing spikes are "pushed" to the left.

During the first sweep the compacted system (42) is transformed by C_1 in Figure 4 which reconditions $(W_{p_i} - V_{p_i}), i = 0, 2, 4, 6$, and then 4 processors are used as explained in (43)-(44) resulting in the system depicted on the right in *Sweep #1* of Figure 3. During the second sweep the system is transformed by C_2 in Figure 4, reconditioning $(W_{p_i}^{(1)} - V_{p_i}^{(1)}), i = 1, 5$. As shown in *Sweep #2* of Figure 3, this creates 2 upward-pointing spikes of 2 blocks each—one above block-row 1 and the other above block-row 5. The upward-pointing spikes are annihilated simultaneously on 4 processors by subtracting appropriate multiples of block-row 1 or 5, while concurrently the downward-pointing spikes are pushed to the left by subtracting appropriate multiples of block-row 1 or 5 on 4 other processors. Finally, during the third sweep the system is transformed by C_3 in Figure 4, reconditioning $W_{p_3}^{(2)} - V_{p_3}^{(2)}$. As shown in *Sweep #3* of Figure 3, this creates an upward spike of 4 blocks which is annihilated simultaneously on 4 processors by subtracting appropriate multiples of block-row 3, while concurrently the spike below $W_{p_3}^{(2)} - V_{p_3}^{(2)}$ is pushed to the left by subracting appropriate multiples of block-row 3 on 4 other processors. In general, this version keeps P processors active during each sweep after the first.

The final system is depicted on the right in *Sweep #3* of Figure 3—a Λ-shaped Jacobian requiring a single-sweep back-solve once $\tilde{y}_{p_0}$ has been computed. As in Figure 2, $\tilde{y}_{p_0}$ and $\tilde{y}_{p_7+1}$ are computed by solving a $2n \times 2n$ system constructed from the first and last block-rows. Finally, the mesh transformation is reversed: $Y = C_1 C_2 C_3 \tilde{Y}$ where $C_1 C_2 C_3$ is shown in the last frame of Figure 4. This, of course, is a highly parallel operation and requires as few as $\log(\log P)$ vector subtractions.

5 Storage and Operation Counts

Table 1 gives the storage requirements and operation counts of the algorithms in §3 and §4

- *SLF*-QR $\rightarrow$ stable local factorization using Householder transformations,
- *SLF*-QR-N $\rightarrow$ *SLF*-QR with push to N-shaped Jacobian,
- *SLF*-GE/rpp $\rightarrow$ *SLF* using Gauss transformations with row partial-pivoting,

Figure 3: Cyclic reduction to Λ-shape Jacobian using Rescaling

Sweep #1

$$
\begin{bmatrix}
\hat{\mathcal{B}}_a & -\hat{\mathcal{B}}_a & & & & & & & \mathcal{B}_b \\
V_{p_0} & W_{p_0}-V_{p_0} & & & & & & & \\
 & V_{p_1} & W_{p_1} & -W_{p_1} & & & & & \\
 & & V_{p_2} & W_{p_2}-V_{p_2} & & & & & \\
 & & & V_{p_3} & W_{p_3} & -W_{p_3} & & & \\
 & & & & V_{p_4} & W_{p_4}-V_{p_4} & & & \\
 & & & & & V_{p_5} & W_{p_5} & -W_{p_5} & \\
 & & & & & & V_{p_6} & W_{p_6}-V_{p_6} & \\
 & & & & & & & V_{p_7} & I
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(1)} & & & & & & & & \mathcal{B}_b \\
V_{p_0}^{(1)} & I & & & & & & & \\
V_{p_1}^{(1)} & & W_{p_1}^{(1)} & & & & & & \\
 & & V_{p_2}^{(1)} & I & & & & & \\
 & & V_{p_3}^{(1)} & & W_{p_3}^{(1)} & & & & \\
 & & & & V_{p_4}^{(1)} & I & & & \\
 & & & & V_{p_5}^{(1)} & & W_{p_5}^{(1)} & & \\
 & & & & & & V_{p_6}^{(1)} & I & \\
 & & & & & & V_{p_7}^{(1)} & & I
\end{bmatrix}
$$

Sweep #2

$$
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(1)} & & -\hat{\mathcal{B}}_a^{(1)} & & & & & & \mathcal{B}_b \\
V_{p_0}^{(1)} & I & -V_{p_0}^{(1)} & & & & & & \\
V_{p_1}^{(1)} & & W_{p_1}^{(1)}-V_{p_1}^{(1)} & & & & & & \\
 & & V_{p_2}^{(1)} & I & & & & & \\
 & & V_{p_3}^{(1)} & & W_{p_3}^{(1)} & & -W_{p_3}^{(1)} & & \\
 & & & & V_{p_4}^{(1)} & I & -V_{p_4}^{(1)} & & \\
 & & & & V_{p_5}^{(1)} & & W_{p_5}^{(1)}-V_{p_5}^{(1)} & & \\
 & & & & & & V_{p_6}^{(1)} & I & \\
 & & & & & & V_{p_7}^{(1)} & & I
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(2)} & & & & & & & & \mathcal{B}_b \\
V_{p_0}^{(2)} & I & & & & & & & \\
V_{p_1}^{(2)} & & I & & & & & & \\
V_{p_2}^{(2)} & & & I & & & & & \\
V_{p_3}^{(2)} & & & & W_{p_3}^{(2)} & & & & \\
 & & & & V_{p_4}^{(2)} & I & & & \\
 & & & & V_{p_5}^{(2)} & & I & & \\
 & & & & V_{p_6}^{(2)} & & & I & \\
 & & & & V_{p_7}^{(2)} & & & & I
\end{bmatrix}
$$

Sweep #3

$$
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(2)} & & & & -\hat{\mathcal{B}}_a^{(2)} & & & & \mathcal{B}_b \\
V_{p_0}^{(2)} & I & & & -V_{p_0}^{(2)} & & & & \\
V_{p_1}^{(2)} & & I & & -V_{p_1}^{(2)} & & & & \\
V_{p_2}^{(2)} & & & I & -V_{p_2}^{(2)} & & & & \\
V_{p_3}^{(2)} & & & & W_{p_3}^{(2)}-V_{p_3}^{(2)} & & & & \\
 & & & & V_{p_4}^{(2)} & I & & & \\
 & & & & V_{p_5}^{(2)} & & I & & \\
 & & & & V_{p_6}^{(2)} & & & I & \\
 & & & & V_{p_7}^{(2)} & & & & I
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
\hat{\mathcal{B}}_a^{(3)} & & & & & & & & \mathcal{B}_b \\
V_{p_0}^{(3)} & I & & & & & & & \\
V_{p_1}^{(3)} & & I & & & & & & \\
V_{p_2}^{(3)} & & & I & & & & & \\
V_{p_3}^{(3)} & & & & I & & & & \\
V_{p_4}^{(3)} & & & & & I & & & \\
V_{p_5}^{(3)} & & & & & & I & & \\
V_{p_6}^{(3)} & & & & & & & I & \\
V_{p_7}^{(3)} & & & & & & & & I
\end{bmatrix}
$$

Figure 4: Mesh transformation used prior to each sweep in Figure 3

$\boxed{\textit{Sweep \#1}}$

$$C_1 = \begin{bmatrix} I & -I & & & & & & & \\ & I & & & & & & & \\ & & I & -I & & & & & \\ & & & I & & & & & \\ & & & & I & -I & & & \\ & & & & & I & & & \\ & & & & & & I & -I & \\ & & & & & & & I & \\ & & & & & & & & I \end{bmatrix}$$

$\boxed{\textit{Sweep \#2}}$

$$C_2 = \begin{bmatrix} I & & -I & & & & & & \\ & I & & & & & & & \\ & & I & & & & & & \\ & & & I & & & & & \\ & & & & I & & -I & & \\ & & & & & I & & & \\ & & & & & & I & & \\ & & & & & & & I & \\ & & & & & & & & I \end{bmatrix}$$

$\boxed{\textit{Sweep \#3}}$

$$C_3 = \begin{bmatrix} I & & & & -I & & & & \\ & I & & & & & & & \\ & & I & & & & & & \\ & & & I & & & & & \\ & & & & I & & & & \\ & & & & & I & & & \\ & & & & & & I & & \\ & & & & & & & I & \\ & & & & & & & & I \end{bmatrix}$$

$$C_1C_2C_3 = \begin{bmatrix} I & -I & -I & & -I & & & & \\ & I & & & & & & & \\ & & I & -I & & & & & \\ & & & I & & & & & \\ & & & & I & -I & -I & & \\ & & & & & I & & & \\ & & & & & & I & -I & \\ & & & & & & & I & \\ & & & & & & & & I \end{bmatrix}$$

- SLF-GE/rpp-N $\rightarrow$ SLF-GE/rpp with push to N-shaped Jacobian,
- GSC-RS $\rightarrow$ global stability control using rescaling,
- GSC-RS-Λ $\rightarrow$ GSC-RS with push to Λ-shaped Jacobian,

along with those of four algorithms discussed in [Wrig 90b] and [Wrig 91a]

- Structured QR $\rightarrow$ Wright's structured orthogonal factorization algorithm,
- Structured LU $\rightarrow$ Wright's structured Gaussian elimination algorithm,
- ROWPP $\rightarrow$ a plain row partial-pivoting code,
- DECOMP/SOLVE $\rightarrow$ routines from the PASVA codes [Lent 77],

and two sequential codes designed for problems with separated boundary conditions only: SOLVEBLOK [deBo 80] and COLROW [Diaz 83].

Structured QR and Structured LU are similar to SLF-QR and SLF-GE/rpp, respectively. SLF-QR, SLF-QR-N, SLF-GE/rpp, SLF-GE/rpp-N, GSC-RS, GSC-RS-Λ, Structured QR and Structured LU are all indifferent to the separability of the boundary conditions, so the entries in Table 1 for these algorithms apply to either the separated or nonseparated case. The operation counts and storage requirements for ROWPP and DECOMP/SOLVE, however, increase noticeably in the nonseparated case. For these two algorithms, the subscript (S) or (C) denotes separated or coupled boundary conditions, respectively. SOLVEBLOK and COLROW can be used on problems with separated boundary conditions only. Problems with coupled boundary conditions may be reformulated by increasing the number of differential equations so that the boundary conditions are separated ([Asch 88, page 5]).

The following notation is used in Table 1: P is the number of processors, M is the number of mesh points, n is the dimension of y, n_a is the number of left boundary conditions, n_b is the number of right boundary conditions (n_a and n_b apply to ROWPP$_{(S)}$, DECOMP/SOLVE$_{(S)}$, SOLVEBLOK and COLROW only), and R is the number of right-hand sides. The storage requirements take into account the Householder or Gauss transformations generated during the factorization which must be retained in order to process subsequent right-hand sides during a modified Newton iteration (5). The discrepancy in operation counts between Structured LU and SLF-GE/rpp can be attributed to a slightly different implementation. The details of the implementation are discussed in [Panc 92].

If $P \geq M$ processors are available the operation counts may be reduced by overlapping some of the computation. For example, forward elimination on the initial right-hand side during the first modified Newton iteration (5) can be carried out concurrently with the factorization, which can reduce the coefficient of the n^2 term. Several other such optimizations are possible when enough processors are available.

The counts for GSC-RS are based on the second form of the Jacobian (8). The current implementation of GSC-RS assumes the second form, so if it does not arise naturally the Jacobian must be preprocessed as described in §1. Multiplying the i-th block row of (7) through by T_i^{-1} adds $1\frac{1}{3}n^3$ to the operation count in Table 1 and n^2 to the storage requirement if subsequent right-hand sides in (5) are to be processed. However, this preprocessing stage may not be necessary as preliminary numerical experiments have shown GSC-RS can be applied directly to (7). This will be investigated further in [Panc 92].

Table 1: Operation counts and storage requirements for various ABD solvers

Algorithm	Operation Count	Storage
SLF-QR	$(\frac{M}{P}+\log P)[\frac{46}{3}n^3+(15R+30)n^2]$	$4Mn^2$
SLF-QR-$N_{(P>1)}$	$\frac{M}{P}[\frac{46}{3}n^3+(15R+30)n^2]+\log P[\frac{46}{3}n^3+(4R+30)n^2]$ $+3n^3+\frac{5}{2}Rn^2$	$(4M+P\log P)n^2$
SLF-GE/rpp	$(\frac{M}{P}+\log P)[4n^3+(4R-1)n^2]$	$4Mn^2$
SLF-GE/rpp-$N_{(P>1)}$	$\frac{M}{P}[4n^3+(4R-1)n^2]+\log P[4n^3+(\frac{3}{2}R-1)n^2]$ $+3n^3+\frac{5}{2}Rn^2$	$(4M+P\log P)n^2$
GSC-RS	$\frac{M}{P}[2\frac{1}{3}n^3+3Rn^2]+\log P[3\frac{1}{3}n^3+4Rn^2]$	$(2M+P)n^2$
GSC-RS-$\Lambda_{(P>1)}$	$\frac{M}{P}[2\frac{1}{3}n^3+3Rn^2]+\log P[3\frac{1}{3}n^3+2Rn^2]+n^2$	$(2M+P\log P)n^2$
Structured QR	$(\frac{M}{P}+\log P)[\frac{46}{3}n^3+(15R+30)n^2]$	$4Mn^2$
Structured LU	$(\frac{M}{P}+\log P)[5n^3+(5R-\frac{3}{2})n^2]$	$4Mn^2$
ROWPP$_{(S)}$	$M[\frac{5}{3}n^3+3n_an^2+R(4n^2+2n_an)]$	$3Mn^2$
ROWPP$_{(C)}$	$M[\frac{23}{3}n^3+8Rn^2]$	$4Mn^2$
DECOMP/SOLVE$_{(S)}$	$M[\frac{2}{3}n^3+(4R+5n_a)n^2-2n_a^2n]$	$2Mn^2$
DECOMP/SOLVE$_{(C)}$	$M[\frac{14}{3}n^3+4Rn^2]$	$3Mn^2$
COLROW$_{(S)}$	$M[\frac{5}{6}n^3+\frac{1}{2}(n_bn^2-n_b^2n)-n^2+R(3n^2-2n)]$	$2Mn^2$
SOLVEBLOK$_{(S)}$	$M[\frac{5}{6}n^3+\frac{3}{2}n_an^2+R(3n^2-2n)]$	$2Mn^2+2Mn_an$

6 Numerical Results

The ABD-solvers described in this paper have been implemented in Fortran-77 double precision on a Sun 4/IPC and tested on some of the model problems recently appearing in the literature. This section compares the accuracy of the cyclic-reduction variation of the algorithms ($P = M/2$ in the partitioned approach).

Acronym	Algorithm	Location
BlkCR	unmodified block cyclic reduction	§2 equations (9)-(11)
SLF-QR	stable local factorization using Householder transformations	§3.1, §3.3 Figure 1
SLF-QR-N	*SLF*-QR with push to N-shaped Jacobian	§3.1, §3.3 Figure 2
SLF-GE/rpp	stable local factorization using Gauss transformations with row partial-pivoting	§3.2, §3.3 Figure 1
SLF-GE/rpp-N	*SLF*-GE/rpp with push to N-shaped Jacobian	§3.2, §3.3 Figure 2
GSC-RS	global stability control using rescaling	§4.3 equations (43)-(46)
GSC-RS-Λ	*GSC*-RS with push to Λ-shaped Jacobian	§4.3 Figures 3 and 4

The solvers have been incorporated into both a finite difference code based on the midpoint rule and a multiple shooting code based on Shampine and Gordon's DESTEP, with the IVP error tolerance set to 10^{-10}. An equally-spaced mesh of M points was used for both methods; tests were run with $M = 32, 64, 256$, and 1024 for the finite difference method, and $M = 16, 32, 64$ and 128 for multiple shooting. For problems where the boundary conditions are separable, the condition number $\mathcal{K}$ of the ABD matrix is estimated by solving the system with Linpack's bandsolver DGBCO, and the algebraic error $\|\hat{Y} - Y_{band}\|_\infty$ is listed. The true error $\|\hat{Y} - Y_{true}\|_\infty$ is shown as well (in brackets), where $y_{true,i} = y(t_i)$ is the analytic solution to the BVP (1)-(2) at the mesh point t_i. (The problems are constructed so that the analytic solution is known.) Note, however, that since the true error is dominated by the discretization error rather than the algebraic error, it probably is not the best way to gauge the accuracy of the solution to the ABD system. In fact, in all of the test problems the *SLF* and *GSC* algorithms are indistinguishable with respect to the true error. The algebraic error, however, varies slightly among the algorithms. In particular, for *GSC*-RS and *GSC*-RS-Λ, the choice of σ in (29) seems to have some effect on the algebraic error. To illustrate this, results for $\sigma = +1$ and $\sigma = -1$ are tabulated—in most cases the latter yields the smallest error of all methods tested. The reason for the difference is currently unknown. It should be noted, however, that in the context of solving a BVP these small variations in the algebraic error normally are of no concern since the true error is usually several orders of magnitude larger. Also, since the algebraic error is measured with respect to the bandsolver DGBCO, the method with the "smallest" algebraic error simply agrees best with that particular code. Consequently, the algebraic errors listed below may not be completely accurate, and the small variations may be insignificant.

All of the problems considered in this section are linear—each is of the form

$$y'(t) = A(t)y(t) + q(t), \quad t \in [a, b],$$

where $y, q \in \mathcal{R}^n$ and $A \in \mathcal{R}^{n \times n}$.

Problem 1 (Ascher and Chan [Asch 89]) $a = 0, b = 1, n = 2$,

$$A(t) = \begin{bmatrix} -\lambda \cos 2\omega t & \omega + \lambda \sin 2\omega t \\ -\omega + \lambda \sin 2\omega t & \lambda \cos 2\omega t \end{bmatrix}, \quad q(t) = (I - A(t)) \begin{bmatrix} e^t \\ e^t \end{bmatrix},$$

subject to the boundary conditions $y_1(0) = 1,\ y_1(1) = e$. The true solution is $y(t) = e^t(1,1)^T$.

A fundamental solution matrix is

$$Y(t) = \begin{bmatrix} \cos \omega t & \sin \omega t \\ -\sin \omega t & \cos \omega t \end{bmatrix} \begin{bmatrix} e^{-\lambda t} & 0 \\ 0 & e^{\lambda t} \end{bmatrix},$$

so the dichotomy is clear: there is one decaying and one growing mode. The problem was first solved with $\lambda = 10$ and $\omega = 1$. With these values the modes are "slow", and as a result all of the solvers—even BlkCR—perform well.

Problem #1 ($\lambda = 10$, $\omega = 1$); discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
$\mathcal{K}$(full jac.)	.7D+01	.1D+02	.5D+02	.2D+03
BlkCR	.86D-12 (.25D-03)	.84D-12 (.64D-04)	.34D-11 (.40D-05)	.70D-12 (.25D-06)
SLF-QR	.27D-14 (.25D-03)	.31D-14 (.64D-04)	.53D-14 (.40D-05)	.12D-13 (.25D-06)
SLF-QR-*N*	.22D-14 (.25D-03)	.36D-14 (.64D-04)	.53D-14 (.40D-05)	.12D-13 (.25D-06)
SLF-GE/rpp	.22D-14 (.25D-03)	.18D-14 (.64D-04)	.22D-14 (.40D-05)	.44D-14 (.25D-06)
SLF-GE/rpp-*N*	.13D-14 (.25D-03)	.22D-14 (.64D-04)	.27D-14 (.40D-05)	.53D-14 (.25D-06)
GSC-RS$_{\sigma=+1}$	.49D-14 (.25D-03)	.22D-14 (.64D-04)	.80D-14 (.40D-05)	.13D-13 (.25D-06)
GSC-RS$_{\sigma=-1}$	.89D-15 (.25D-03)	.13D-14 (.64D-04)	.18D-14 (.40D-05)	.44D-14 (.25D-06)
GSC-RS-$\Lambda_{\sigma=+1}$	.12D-13 (.25D-03)	.18D-13 (.64D-04)	.86D-13 (.40D-05)	.64D-12 (.25D-06)
GSC-RS-$\Lambda_{\sigma=-1}$	.89D-15 (.25D-03)	.18D-14 (.64D-04)	.22D-14 (.40D-05)	.49D-14 (.25D-06)

Problem #1 ($\lambda = 10$, $\omega = 1$); discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
$\mathcal{K}$(full jac.)	.5D+01	.8D+01	.1D+02	.3D+02
BlkCR	.30D-12 (.20D-10)	.37D-11 (.19D-10)	.15D-11 (.18D-10)	.11D-11 (.13D-10)
SLF-QR	.18D-14 (.20D-10)	.13D-14 (.19D-10)	.22D-14 (.18D-10)	.31D-14 (.13D-10)
SLF-QR-*N*	.22D-14 (.20D-10)	.18D-14 (.19D-10)	.22D-14 (.18D-10)	.36D-14 (.13D-10)
SLF-GE/rpp	.89D-15 (.20D-10)	.13D-14 (.19D-10)	.13D-14 (.18D-10)	.18D-14 (.13D-10)
SLF-GE/rpp-*N*	.89D-15 (.20D-10)	.89D-15 (.19D-10)	.13D-14 (.18D-10)	.16D-14 (.13D-10)
GSC-RS$_{\sigma=+1}$	.27D-14 (.20D-10)	.31D-14 (.19D-10)	.27D-14 (.18D-10)	.60D-14 (.13D-10)
GSC-RS$_{\sigma=-1}$	.44D-15 (.20D-10)	.89D-15 (.19D-10)	.13D-14 (.18D-10)	.18D-14 (.13D-10)
GSC-RS-$\Lambda_{\sigma=+1}$	.33D-14 (.20D-10)	.73D-14 (.19D-10)	.21D-13 (.18D-10)	.40D-13 (.13D-10)
GSC-RS-$\Lambda_{\sigma=-1}$	.89D-15 (.20D-10)	.11D-14 (.19D-10)	.18D-14 (.18D-10)	.22D-14 (.13D-10)

With $\lambda = 200$ and $\omega = 1$, the inability of BlkCR to decouple the fundamental solution modes becomes evident as all accuracy is lost to machine round-off. The *SLF* and *GSC* algorithms give solutions which, when compared to the band-solver, are accurate up to the conditioning of the discrete system. Of particular interest in the multiple shooting table is the ABD arising from shooting over $M = 16$ intervals. This system is so poorly conditioned that Ascher and Chan's normal equations method [Asch 89] breaks down. In contrast, the mesh transformation used in the *GSC* algorithms does *not* adversely affect the condition of the ABD, so these algorithms perform well on this poorly conditioned system suggesting that they are not sensitive to poorly conditioned problems, a conclusion supported by many other numerical test results.

Problem #1 ($\lambda = 200, \omega = 1$); discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
$\mathcal{K}$(full jac.)	.5D+01	.8D+01	.5D+01	.1D+02
BlkCR	.19D-06 (.49D-03)	.15D+27 (.15D+27)	.36D+76 (.36D+76)	.14D+72 (.14D+72)
SLF-QR	.22D-14 (.49D-03)	.89D-14 (.10D-03)	.36D-14 (.51D-05)	.44D-14 (.32D-06)
SLF-QR-*N*	.27D-14 (.49D-03)	.89D-14 (.10D-03)	.40D-14 (.51D-05)	.49D-14 (.32D-06)
SLF-GE/rpp	.11D-14 (.49D-03)	.22D-14 (.10D-03)	.27D-14 (.51D-05)	.27D-14 (.32D-06)
SLF-GE/rpp-*N*	.22D-14 (.49D-03)	.27D-14 (.10D-03)	.22D-14 (.51D-05)	.31D-14 (.32D-06)
GSC-RS$_{\sigma=+1}$	.54D-13 (.49D-03)	.16D-12 (.10D-03)	.61D-13 (.51D-05)	.96D-13 (.32D-06)
GSC-RS$_{\sigma=-1}$	.13D-14 (.49D-03)	.22D-14 (.10D-03)	.13D-14 (.51D-05)	.18D-14 (.32D-06)
GSC-RS-$\Lambda_{\sigma=+1}$	.54D-13 (.49D-03)	.16D-12 (.10D-03)	.16D-12 (.51D-05)	.42D-12 (.32D-06)
GSC-RS-$\Lambda_{\sigma=-1}$	.18D-14 (.49D-03)	.22D-14 (.10D-03)	.22D-14 (.51D-05)	.22D-14 (.32D-06)

Problem #1 ($\lambda = 200, \omega = 1$); discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
$\mathcal{K}$(full jac.)	.2D+06	.5D+03	.3D+02	.8D+01
BlkCR	.28D+66 (.28D+66)	.73D+68 (.73D+68)	.15D+69 (.15D+69)	.99D+70 (.99D+70)
SLF-QR	.96D-10 (.17D-06)	.37D-12 (.17D-09)	.23D-13 (.92D-10)	.40D-14 (.53D-10)
SLF-QR-*N*	.96D-10 (.17D-06)	.37D-12 (.17D-09)	.23D-13 (.92D-10)	.47D-14 (.53D-10)
SLF-GE/rpp	.57D-10 (.17D-06)	.13D-12 (.17D-09)	.84D-14 (.92D-10)	.36D-14 (.53D-10)
SLF-GE/rpp-*N*	.49D-10 (.17D-06)	.24D-12 (.17D-09)	.15D-13 (.92D-10)	.53D-14 (.53D-10)
GSC-RS$_{\sigma=+1}$	.23D-09 (.17D-06)	.92D-12 (.17D-09)	.12D-12 (.92D-10)	.54D-13 (.53D-10)
GSC-RS$_{\sigma=-1}$	.39D-10 (.17D-06)	.12D-12 (.17D-09)	.62D-14 (.92D-10)	.16D-14 (.53D-10)
GSC-RS-$\Lambda_{\sigma=+1}$	.23D-09 (.17D-06)	.92D-12 (.17D-09)	.13D-12 (.92D-10)	.13D-12 (.53D-10)
GSC-RS-$\Lambda_{\sigma=-1}$	.39D-10 (.17D-06)	.12D-12 (.17D-09)	.67D-14 (.92D-10)	.22D-14 (.53D-10)

Problem 2A (Wright [Wrig 91a]) $a = 0, b = \pi, n = 3$,

$$A(t) = \begin{bmatrix} 1 - 19\cos 2t & 0 & 1 + 19\sin 2t \\ 0 & 19 & 0 \\ -1 + 19\sin 2t & 0 & 1 + 19\cos 2t \end{bmatrix}, \quad q(t) = (I - A(t)) \begin{bmatrix} e^t \\ e^t \\ e^t \end{bmatrix},$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= 1 \\ y_2(\pi) &= e^{\pi} \\ y_1(\pi) + 3y_3(\pi) &= 4e^{\pi}. \end{aligned}$$

The true solution is $y(t) = e^t(1,1,1)^T$. A fundamental solution matrix is

$$Y(t) = \begin{bmatrix} \sin t & 0 & -\cos t \\ 0 & 1 & 0 \\ \cos t & 0 & \sin t \end{bmatrix} \begin{bmatrix} e^{20t} & 0 & 0 \\ 0 & e^{19t} & 0 \\ 0 & 0 & e^{-18t} \end{bmatrix}.$$

Problem #2A ; discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
$\mathcal{K}$(full jac.)	.1D+03	.6D+01	.2D+02	.7D+02
BlkCR	.46D+47 (.46D+47)	.17D+14 (.17D+14)	.85D+11 (.85D+11)	.85D+11 (.85D+11)
SLF-QR	.30D-12 (.28D-01)	.21D-13 (.68D-02)	.32D-13 (.42D-03)	.60D-13 (.26D-04)
SLF-QR-*N*	.30D-12 (.28D-01)	.18D-13 (.68D-02)	.32D-13 (.42D-03)	.57D-13 (.26D-04)
SLF-GE/rpp	.18D-12 (.28D-01)	.71D-14 (.68D-02)	.89D-14 (.42D-03)	.18D-13 (.26D-04)
SLF-GE/rpp-*N*	.19D-12 (.28D-01)	.12D-13 (.68D-02)	.11D-13 (.42D-03)	.21D-13 (.26D-04)
GSC-RS$_{\sigma=+1}$	.14D-11 (.28D-01)	.17D-12 (.68D-02)	.51D-12 (.42D-03)	.69D-12 (.26D-04)
GSC-RS$_{\sigma=-1}$	.76D-13 (.28D-01)	.53D-14 (.68D-02)	.14D-13 (.42D-03)	.21D-13 (.26D-04)
GSC-RS-$\Lambda_{\sigma=+1}$	.15D-11 (.28D-01)	.54D-12 (.68D-02)	.37D-11 (.42D-03)	.15D-10 (.26D-04)
GSC-RS-$\Lambda_{\sigma=-1}$	.80D-13 (.28D-01)	.71D-14 (.68D-02)	.18D-13 (.42D-03)	.28D-13 (.26D-04)

Problem #2A ; discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
$\mathcal{K}$(full jac.)	.1D+03	.2D+02	.1D+02	.1D+02
BlkCR	.10D+10 (.10D+10)	.23D+10 (.23D+10)	.37D+11 (.37D+11)	.48D+11 (.48D+11)
SLF-QR	.48D-13 (.59D-09)	.99D-13 (.20D-09)	.14D-13 (.25D-09)	.21D-13 (.16D-09)
SLF-QR-*N*	.45D-13 (.59D-09)	.99D-13 (.20D-09)	.14D-13 (.25D-09)	.21D-13 (.16D-09)
SLF-GE/rpp	.18D-13 (.59D-09)	.25D-13 (.20D-09)	.89D-14 (.25D-09)	.71D-14 (.16D-09)
SLF-GE/rpp-*N*	.14D-13 (.59D-09)	.25D-13 (.20D-09)	.11D-13 (.25D-09)	.89D-14 (.16D-09)
GSC-RS$_{\sigma=+1}$	.13D-12 (.59D-09)	.83D-13 (.20D-09)	.37D-12 (.25D-09)	.60D-12 (.16D-09)
GSC-RS$_{\sigma=-1}$	.21D-13 (.59D-09)	.11D-13 (.20D-09)	.53D-14 (.25D-09)	.71D-14 (.16D-09)
GSC-RS-$\Lambda_{\sigma=+1}$	.12D-12 (.59D-09)	.20D-12 (.20D-09)	.44D-12 (.25D-09)	.17D-11 (.16D-09)
GSC-RS-$\Lambda_{\sigma=-1}$	.21D-13 (.59D-09)	.14D-13 (.20D-09)	.11D-13 (.25D-09)	.71D-14 (.16D-09)

Problem 2B (Mattheij [Matt 85])

Same as Problem 2A, but with coupled boundary conditions

$$\begin{aligned} y_1(0) &= 1 \\ y_3(0) + y_3(\pi) &= 1 + e^\pi \\ y_2(0) + y_2(\pi) &= 1 + e^\pi. \end{aligned}$$

Since the linear system is not banded for this problem, DGBCO cannot be used for the condition number estimate and algebraic error. Consequently, only the true error is shown.

Problem #2B ; discretization: *midpoint finite difference*

	$M=32$	$M=64$	$M=256$	$M=1024$
BlkCR	.46D+47	.52D+14	.85D+11	.85D+11
SLF-QR	.28D-01	.68D-02	.42D-03	.26D-04
SLF-QR-N	.28D-01	.68D-02	.42D-03	.26D-04
SLF-GE/rpp	.28D-01	.68D-02	.42D-03	.26D-04
SLF-GE/rpp-N	.28D-01	.68D-02	.42D-03	.26D-04
GSC-RS$_{\sigma=+1}$	.28D-01	.68D-02	.42D-03	.26D-04
GSC-RS$_{\sigma=-1}$	.28D-01	.68D-02	.42D-03	.26D-04
GSC-RS-$\Lambda_{\sigma=+1}$	.28D-01	.68D-02	.42D-03	.26D-04
GSC-RS-$\Lambda_{\sigma=-1}$	.28D-01	.68D-02	.42D-03	.26D-04

Problem #2B ; discretization: *multiple shooting with DE*

	$M=16$	$M=32$	$M=64$	$M=128$
BlkCR	.10D+10	.23D+10	.37D+11	.48D+11
SLF-QR	.59D-09	.20D-09	.25D-09	.16D-09
SLF-QR-N	.59D-09	.20D-09	.25D-09	.16D-09
SLF-GE/rpp	.59D-09	.20D-09	.25D-09	.16D-09
SLF-GE/rpp-N	.59D-09	.20D-09	.25D-09	.16D-09
GSC-RS$_{\sigma=+1}$	.59D-09	.20D-09	.25D-09	.16D-09
GSC-RS$_{\sigma=-1}$	.59D-09	.20D-09	.25D-09	.16D-09
GSC-RS-$\Lambda_{\sigma=+1}$	.59D-09	.20D-09	.25D-09	.16D-09
GSC-RS-$\Lambda_{\sigma=-1}$	.59D-09	.20D-09	.25D-09	.16D-09

Problem 3A (Wright [Wrig 91a]) $a=0, b=1, n=5$,

$$A(t)=\begin{bmatrix} -\lambda_1\cos 2\omega_1 t & 0 & \omega_1+\lambda_1\sin 2\omega_1 t & 0 & 0 \\ 0 & -\lambda_2\cos 2\omega_2 t & 0 & \omega_2+\lambda_2\sin 2\omega_2 t & 0 \\ -\omega_1+\lambda_1\sin 2\omega_1 t & 0 & \lambda_1\cos 2\omega_1 t & 0 & 0 \\ 0 & -\omega_2+\lambda_2\sin 2\omega_2 t & 0 & \lambda_2\cos 2\omega_2 t & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 \end{bmatrix},$$

$$q(t)=(I-A(t))\begin{bmatrix} e^t \\ e^t \\ e^t \\ e^t \\ e^t \end{bmatrix},$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= 1 \\ y_2(0)+4y_5(0) &= 5 \end{aligned}$$

$$\begin{aligned} y_1(1) &= e \\ -y_3(1)+y_4(1) &= 0 \\ -4y_2(1)+5y_5(1) &= e. \end{aligned}$$

The true solution is $y(t) = e^t(1,1,1,1,1)^T$. The problem is solved with $\lambda_1 = 200$, $\lambda_2 = 50$, $\lambda_3 = 10$, $\omega_1 = 1$, and $\omega_2 = 25$. A fundamental solution is

$$Y(t) = \begin{bmatrix} \cos t & 0 & \sin t & 0 & 0 \\ 0 & \cos 25t & 0 & \sin 25t & 0 \\ -\sin t & 0 & \cos t & 0 & 0 \\ 0 & -\sin 25t & 0 & \cos 25t & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e^{-200t} & 0 & 0 & 0 & 0 \\ 0 & e^{-50t} & 0 & 0 & 0 \\ 0 & 0 & e^{200t} & 0 & 0 \\ 0 & 0 & 0 & e^{50t} & 0 \\ 0 & 0 & 0 & 0 & e^{10t} \end{bmatrix}.$$

Problem #3A ; discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
$\mathcal{K}$(full jac.)	.2D+02	.3D+02	.1D+03	.5D+03
BlkCR	.15D+08 (.15D+08)	.15D+27 (.15D+27)	.36D+76 (.36D+76)	.14D+72 (.14D+72)
SLF-QR	.36D-14 (.52D-03)	.89D-14 (.13D-03)	.16D-13 (.82D-05)	.42D-13 (.51D-06)
SLF-QR-N	.44D-14 (.52D-03)	.89D-14 (.13D-03)	.16D-13 (.82D-05)	.42D-13 (.51D-06)
SLF-GE/rpp	.22D-14 (.52D-03)	.22D-14 (.13D-03)	.27D-14 (.82D-05)	.44D-14 (.51D-06)
SLF-GE/rpp-N	.22D-14 (.52D-03)	.27D-14 (.13D-03)	.27D-14 (.82D-05)	.49D-14 (.51D-06)
GSC-RS$_{\sigma=+1}$	.10D-11 (.52D-03)	.16D-12 (.13D-03)	.64D-13 (.82D-05)	.12D-12 (.51D-06)
GSC-RS$_{\sigma=-1}$	.22D-14 (.52D-03)	.22D-14 (.13D-03)	.24D-14 (.82D-05)	.27D-14 (.51D-06)
GSC-RS-$\Lambda_{\sigma=+1}$	.10D-11 (.52D-03)	.16D-12 (.13D-03)	.22D-12 (.82D-05)	.11D-11 (.51D-06)
GSC-RS-$\Lambda_{\sigma=-1}$	.18D-14 (.52D-03)	.22D-14 (.13D-03)	.27D-14 (.82D-05)	.27D-14 (.51D-06)

Problem #3A ; discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
$\mathcal{K}$(full jac.)	.2D+06	.1D+04	.1D+03	.6D+02
BlkCR	.39D+61 (.39D+61)	.88D+65 (.88D+65)	.64D+68 (.64D+68)	.99D+70 (.99D+70)
SLF-QR	.55D-10 (.98D-07)	.20D-12 (.21D-09)	.20D-13 (.94D-10)	.49D-14 (.54D-10)
SLF-QR-N	.55D-10 (.98D-07)	.20D-12 (.21D-09)	.20D-13 (.94D-10)	.53D-14 (.54D-10)
SLF-GE/rpp	.38D-10 (.98D-07)	.18D-12 (.21D-09)	.12D-13 (.94D-10)	.36D-14 (.54D-10)
SLF-GE/rpp-N	.54D-10 (.98D-07)	.30D-12 (.21D-09)	.17D-13 (.94D-10)	.36D-14 (.54D-10)
GSC-RS$_{\sigma=+1}$	.18D-09 (.98D-07)	.23D-11 (.21D-09)	.21D-12 (.94D-10)	.42D-13 (.54D-10)
GSC-RS$_{\sigma=-1}$	.38D-10 (.98D-07)	.99D-13 (.21D-09)	.67D-14 (.94D-10)	.20D-14 (.54D-10)
GSC-RS-$\Lambda_{\sigma=+1}$	.18D-09 (.98D-07)	.23D-11 (.21D-09)	.47D-12 (.94D-10)	.29D-12 (.54D-10)
GSC-RS-$\Lambda_{\sigma=-1}$	.38D-10 (.98D-07)	.99D-13 (.21D-09)	.71D-14 (.94D-10)	.20D-14 (.54D-10)

Problem 3B (Wright [Wrig 91a])

Same as Problem 3A, but with coupled boundary conditions

$$\begin{aligned} y_1(0) &= 1 \\ y_2(0) + 4y_5(0) + y_3(1) &= 5 + e \\ -5y_1(0) + y_1(1) &= -5 + e \\ 3y_2(0) - y_3(1) + y_4(1) &= 3 \\ -4y_2(1) + 5y_5(1) &= e. \end{aligned}$$

Only the true error is shown.

Problem #3B ; discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
BlkCR	.15D+08	.21D+34	.64D+79	.14D+74
SLF-QR	.20D-02	.51D-03	.32D-04	.20D-05
SLF-QR-*N*	.20D-02	.51D-03	.32D-04	.20D-05
SLF-GE/rpp	.20D-02	.51D-03	.32D-04	.20D-05
SLF-GE/rpp-*N*	.20D-02	.51D-03	.32D-04	.20D-05
GSC-RS$_{\sigma=+1}$	.20D-02	.51D-03	.32D-04	.20D-05
GSC-RS$_{\sigma=-1}$	.20D-02	.51D-03	.32D-04	.20D-05
GSC-RS-$\Lambda_{\sigma=+1}$	.20D-02	.51D-03	.32D-04	.20D-05
GSC-RS-$\Lambda_{\sigma=-1}$	.20D-02	.51D-03	.32D-04	.20D-05

Problem #3B ; discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
BlkCR	.49D+61	.88D+65	.64D+68	.99D+70
SLF-QR	.98D-07	.26D-09	.94D-10	.54D-10
SLF-QR-*N*	.98D-07	.26D-09	.94D-10	.54D-10
SLF-GE/rpp	.98D-07	.26D-09	.94D-10	.54D-10
SLF-GE/rpp-*N*	.98D-07	.26D-09	.94D-10	.54D-10
GSC-RS$_{\sigma=+1}$	.98D-07	.26D-09	.94D-10	.54D-10
GSC-RS$_{\sigma=-1}$	.98D-07	.26D-09	.94D-10	.54D-10
GSC-RS-$\Lambda_{\sigma=+1}$	.98D-07	.26D-09	.94D-10	.54D-10
GSC-RS-$\Lambda_{\sigma=-1}$	.98D-07	.26D-09	.94D-10	.54D-10

Problem 4 $a = 0, b = 10, n = 2,$

$$A(t) = U^T(t)\begin{bmatrix} -1 & 10 \\ 0 & -2 \end{bmatrix} U(t), \quad q(t) = (I - 10A(t))\begin{bmatrix} e^{t/10} \\ e^{t/10} \end{bmatrix},$$

where

$$U(t) = \begin{bmatrix} \cos 2\pi t & -\sin 2\pi t \\ \sin 2\pi t & \cos 2\pi t \end{bmatrix},$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= 10 \\ y_1(10) &= 10e. \end{aligned}$$

The true solution is $y(t) = 10e^{t/10}(1,1)^T$.

Since $U(t)$ is orthonormal, the eigenvalues of $A(t)$ are $[-1,-2]$ for all t. This problem was chosen mainly as a test case for the *GSC*-algorithms because, in a boundary-value problem, it is not typical for the spectrum of $A(t)$ to be negative for all t. However, the kinematic eigenvalues are 3.3583 and -6.3583, so this ODE has an exponential dichotomy and the problem is well-posed. (See [Asch 88] for a discussion of these concepts.)

Problem #4 ; discretization: *midpoint finite difference*

	$M = 32$	$M = 64$	$M = 256$	$M = 1024$
$\mathcal{K}$(full jac.)	.2D+03	.1D+02	.2D+02	.7D+02
BlkCR	.60D-13 (.17D-01)	.32D-03 (.17D-02)	.17D+00 (.17D+00)	.31D-01 (.31D-01)
SLF-QR	.46D-13 (.17D-01)	.36D-13 (.14D-02)	.32D-13 (.77D-04)	.71D-13 (.48D-05)
SLF-QR-N	.64D-13 (.17D-01)	.39D-13 (.14D-02)	.39D-13 (.77D-04)	.75D-13 (.48D-05)
SLF-GE/rpp	.75D-13 (.17D-01)	.14D-13 (.14D-02)	.25D-13 (.77D-04)	.28D-13 (.48D-05)
SLF-GE/rpp-N	.60D-13 (.17D-01)	.28D-13 (.14D-02)	.28D-13 (.77D-04)	.28D-13 (.48D-05)
GSC-RS$_{\sigma=+1}$	.21D-12 (.17D-01)	.32D-12 (.14D-02)	.37D-12 (.77D-04)	.61D-12 (.48D-05)
GSC-RS$_{\sigma=-1}$	.18D-12 (.17D-01)	.39D-13 (.14D-02)	.21D-13 (.77D-04)	.25D-13 (.48D-05)
GSC-RS-$\Lambda_{\sigma=+1}$	.60D-12 (.17D-01)	.59D-12 (.14D-02)	.22D-11 (.77D-04)	.77D-11 (.48D-05)
GSC-RS-$\Lambda_{\sigma=-1}$	.19D-12 (.17D-01)	.43D-13 (.14D-02)	.25D-13 (.77D-04)	.32D-13 (.48D-05)

Problem #4 ; discretization: *multiple shooting with DE*

	$M = 16$	$M = 32$	$M = 64$	$M = 128$
$\mathcal{K}$(full jac.)	.1D+02	.6D+01	.7D+01	.1D+02
BlkCR	.27D-01 (.27D-01)	.68D-01 (.68D-01)	.29D+00 (.29D+00)	.48D-01 (.48D-01)
SLF-QR	.92D-13 (.94D-09)	.32D-13 (.38D-09)	.21D-13 (.33D-09)	.25D-13 (.21D-09)
SLF-QR-N	.92D-13 (.94D-09)	.39D-13 (.38D-09)	.25D-13 (.33D-09)	.36D-13 (.21D-09)
SLF-GE/rpp	.46D-13 (.94D-09)	.14D-13 (.38D-09)	.11D-13 (.33D-09)	.18D-13 (.21D-09)
SLF-GE/rpp-N	.25D-13 (.94D-09)	.21D-13 (.38D-09)	.14D-13 (.33D-09)	.21D-13 (.21D-09)
GSC-RS$_{\sigma=+1}$	.77D-12 (.94D-09)	.69D-12 (.38D-09)	.12D-11 (.33D-09)	.37D-12 (.21D-09)
GSC-RS$_{\sigma=-1}$	.28D-13 (.94D-09)	.14D-13 (.38D-09)	.18D-13 (.33D-09)	.16D-13 (.21D-09)
GSC-RS-$\Lambda_{\sigma=+1}$	.82D-12 (.94D-09)	.75D-12 (.38D-09)	.12D-11 (.33D-09)	.66D-12 (.21D-09)
GSC-RS-$\Lambda_{\sigma=-1}$	.28D-13 (.94D-09)	.18D-13 (.38D-09)	.14D-13 (.33D-09)	.18D-13 (.21D-09)

References

[Asch 89] U.M. Ascher and P.S.Y. Chan, *On parallel methods for boundary value ODEs*, University of British Columbia, Department of Computer Science, Technical Report 89-19, 1989.

[Asch 88] U.M. Ascher, R.M.M. Mattheij, and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, 1988.

[Asch 81] U.M. Ascher, J. Christiansen, and R.D. Russell, *Collocation software for boundary-value ODEs*, ACM Trans. Math. Software, 7 (1981), pp. 209-222.

[Bade 87] G. Bader and U.M. Ascher, *A new basis implementation for a mixed order boundary value ODE solver*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 483-500.

[Benn 90] K.R. Bennett and G. Fairweather, *PCOLNEW: A parallel boundary-value ODE code for shared-memory machines*, University of Kentucky, Center for Computational Sciences, Technical Report CCS-90-8, 1990.

[deBo 80] C. de Boor and R. Weiss, *SOLVEBLOK: A package for solving almost block diagonal linear systems*, ACM Trans. Math. Software, 6 (1980), pp. 80-87.

[Diaz 83] J.C. Diaz, G. Fairweather, and P. Keast, *COLROW and ARCECO: Fortran packages for solving certain block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, 9 (1983), pp. 358-377.

[Gear 88] C.W. Gear, *Massive parallelism across the method in ODEs*, University of Illinois, Computer Science Department, Technical Report UIUCDCS-R-88-1442, 1988.

[Golu 83] G. Golub and C. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1983.

[Hell 76] D. Heller, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[Hock 65] R. Hockney, *A Fast Direct Solution of Poisson's Equation Using Fourier Analysis*, J. ACM, 12 (1965), pp. 95-113.

[Hock 70] R. Hockney, *The Potential Calculation and Some Applications*, Meth. Comput. Phys., 9 (1970), pp. 135-211.

[Lent 77] M. Lentini and V. Pereyra, *An adaptive finite difference solver for nonlinear two-point boundary value problems with mild boundary layers*, SIAM J. Numer. Anal., 14 (1977), pp. 91-111.

[Lent 89] M. Lentini, *Parallel solution of special large block tridiagonal systems: TPBVP*, manuscript (1989).

[Matt 85] R.M.M. Mattheij, *Decoupling and stability of algorithms for boundary value problems*, SIAM Review, 27 (1985), pp. 1-44.

[Muir 91] P. Muir, *Private communication.*

[Panc 92] R.N. Pancer, *The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs*, Ph.D. thesis, University of Toronto, (in progress).

[Papr 89] M. Paprzycki and I. Gladwell, *Solving almost block diagonal systems on parallel computers*, SMU Mathematics Department Technical Report 89-18, 1989.

[Sche 84] U. Schendel, *Introduction to Numerical Methods for Parallel Computers*, Ellis Horwood, New York, 1984.

[Wrig 90a] S.J. Wright and V. Pereyra, *Adaptation of a two-point boundary value problem solver to a vector-multiprocessor environment*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 425-449.

[Wrig 90b] S.J. Wright, *Stable parallel algorithms for two-point boundary value problems*, Preprint MCS-P178-0990, Mathematics and Computer Science Division, Argonne National Laboratory (1990). To appear in SIAM J. Sci. Stat. Comput. (1992).

[Wrig 91a] S.J. Wright, *Stable parallel elimination for boundary value ODEs*, Preprint MCS-P229-0491, Mathematics and Computer Science Division, Argonne National Laboratory (1991).

[Wrig 91b] S.J. Wright, *A collection of problems for which Gaussian elimination with partial pivoting is unstable*, Manuscript, Mathematics and Computer Science Division, Argonne National Laboratory (1991).