The Parallel Solution of Almost Block Diagonal Systems Arising in Numerical Methods for BVPs for ODEs

R.N. Pancer and K.R. Jackson
Department of Computer Science, University of Toronto, Toronto, Canada
e-mail: pancer@cs.toronto.edu, krj@cs.toronto.edu

Keywords: Two-point BVP, ODE, ABD linear system, parallel numerical method

ABSTRACT

Three parallel algorithms are presented for solving the Almost Block Diagonal (ABD) systems of linear algebraic equations that arise during the numerical solution of Boundary Value Problems (BVPs) for Ordinary Differential Equations (ODEs). Until recently, the solution of these systems has proven to be the execution-time bottleneck in the parallel implementation of most BVP codes. Several numerical examples are given comparing the three algorithms with respect to accuracy and speed.

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

$$y'(t) = f(t, y(t)), \quad t \in [a, b], \quad g(y(a), y(b)) = 0, \tag{1}$$

where $y, f, g \in \mathcal{R}^n$. When solving this problem numerically, the interval $[a, b]$ is first subdivided into a mesh: $a = t_0 < t_1 < \cdots < t_M = b$. Many techniques then proceed to compute a discrete approximation of the form $Y = \left[y_0^T, y_1^T, \ldots, y_M^T\right]^T \in \mathcal{R}^{(M+1)n}$ where $y_i \approx y(t_i)$. 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 $\Phi(Y)$ 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. In this case,

$$\Phi(Y) = \left[g(y_0, y_M)^T, \phi_0(y_0, y_1)^T, \phi_1(y_1, y_2)^T, \ldots, \phi_{M-1}(y_{M-1}, y_M)^T\right]^T, \tag{2}$$

where each $\phi_i \in \mathcal{R}^n$ and $\Phi : \mathcal{R}^{(M+1)n} \longrightarrow \mathcal{R}^{(M+1)n}$.

When Newton's method or one of its variants is used to solve (2), the Jacobian $\mathcal{J}(Y) = \partial\Phi(Y)/\partial Y$ has a "staircase" or Almost Block Diagonal (ABD) form. Augmented with its right-hand side, the linear system that arises during the q-th Newton-iteration is

$$\left[\mathcal{J}(Y^{(q)}) | \Phi(Y^{(q)})\right] = \left[\begin{array}{cccccc|c} \mathcal{B}_a & & & & & \mathcal{B}_b & g \\ S_0 & T_0 & & & & & \phi_0 \\ & S_1 & T_1 & & & & \phi_1 \\ & & \ddots & \ddots & & & \vdots \\ & & & S_{M-1} & T_{M-1} & & \phi_{M-1} \end{array}\right], \tag{3}$$

where $\mathcal{B}_a = \partial g/\partial y_0$, $\mathcal{B}_b = \partial g/\partial y_M$, and $[S_i, T_i] = [\partial\phi_i/\partial y_i, \partial\phi_i/\partial y_{i+1}]$. $\mathcal{J}$ may be re-evaluated and re-factored at each iteration, or may be held fixed for several iterations if the rate of convergence is acceptable. These issues and several others that need to be addressed in the numerical solution of BVPs are discussed further in [1].

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 $\mathcal{J}(Y^{(q)})$ can be constructed independently each time $\mathcal{J}$ is re-evaluated. Analysis in [3] and [7] shows that the sequential solution of (3) can easily become the execution-time bottleneck in a BVP code once the residual evaluation and Jacobian construction phases have been parallelized, emphasizing the importance of developing a parallel ABD solver.

The greatest potential for speed-up in the parallel solution of ABD linear systems exists across the blocks (i.e. in M), since typically $M \gg n$ in a BVP code. Suppressing the dependence on n, the theoretically best parallel complexity is $O(\log M)$ block-steps. Most parallel algorithms proposed in the past, however, either do not achieve the optimal speed-up ([6], [11]) or suffer from poor stability properties ([2], [10]). Unstable algorithms either fail to control the growth of the underlying exponentially increasing fundamental solution modes in (1), which is usually required for a stable numerical solution, or perform a transformation that adversely affects the condition of the ABD matrix.

Wright [12] was the first to present a *stable* parallel algorithm that attains the theoretically optimal speed-up for this problem. A faster but less stable version of his algorithm is presented in [13]. The *SLF*-QR and *SLF*-LU algorithms described in the next section of this paper were discovered independently by us [8] and are similar to those of Wright. Also presented in the next section is RSCALE, a third algorithm developed by us which requires fewer local operations than either *SLF*-QR or *SLF*-LU and has better stability properties than *SLF*-LU. The three algorithms are compared with respect to accuracy and speed in the Numerical Results section.

OUTLINE OF THE ALGORITHMS

Several variations of *SLF*-QR, *SLF*-LU and RSCALE are discussed in [8] and [9]. Probably the most suitable variation for architectures with a modest number of processors is a two-level factorization. In the first level of such a scheme, the last M block-rows of (3) are divided into P partitions of M/P block-rows each in such a way that each partition can be factored independently. In the second level, the boundary blocks and a single block-row from each partition are extracted to form a $P+1$ block-row ABD matrix which is factored sequentially. The three algorithms differ in how local transformations are applied within each partition during the first level of the factorization. This is discussed below.

Let $[i]_r$ denote the block-row containing $[S_i, T_i, \phi_i]$. Starting at the top block-row pair $[i, i{+}1]_r$ of each partition, *SLF*-QR and *SLF*-LU proceed to compute the factorization $[T_i^T,\, S_{i+1}^T]^T = E[\tilde{R}_i^T,\, 0]^T$ where $E \in \mathcal{R}^{2n\times 2n}$ and $\tilde{R}_i \in \mathcal{R}^{n\times n}$ is upper-triangular. Conceptually, once E is obtained, $[i, i{+}1]_r$ is multiplied through by E^{-1} giving

$$\left[\begin{array}{cccccc|c} \multicolumn{6}{l|}{\text{top of partition}} & \\ \hline \searrow & S_i & T_i & & & \cdots & \phi_i \\ \cdots & & S_{i+1} & T_{i+1} & & \cdots & \phi_{i+1} \\ \cdots & & & S_{i+2} & T_{i+2} & \cdots & \phi_{i+2} \end{array}\right] \longrightarrow \left[\begin{array}{cccccc|c} & & & & & & \\ \hline \cdots & \tilde{S}_i & \tilde{R}_i & \tilde{T}_i & & \cdots & \tilde{\phi}_i \\ \searrow & \tilde{S}_{i+1} & & \tilde{T}_{i+1} & & \cdots & \tilde{\phi}_{i+1} \\ \cdots & & & S_{i+2} & T_{i+2} & \cdots & \phi_{i+2} \end{array}\right]$$

These steps are applied to successive block-row pairs $[i{+}1, 1{+}2]_r$, $[i{+}2, i{+}3]_r$, $\ldots$, until the bottom of the partition is reached. When the first-level factorization is complete, the mesh variables associated with the block-rows at the bottom of each partition have been effectively uncoupled from the original system, and can be extracted to form a $P+1$ block-row ABD matrix which is factored sequentially. Further details on this two-level factorization scheme are given in [8], [9], [12] and [13].

In *SLF*-QR, E is orthogonal. As pointed out in [12], *SLF*-QR is essentially QR-factorization applied to a row and column-permutation of the original ABD matrix, and hence is stable. In *SLF*-LU, $E^{-1} = \tilde{L}_n P_n \cdots \tilde{L}_2 P_2 \tilde{L}_1 P_1$, where $\tilde{L}_j$ is an elementary Gauss transformation and P_j is a permutation.

Figure 1: *Steps in the RSCALE local transformation.* $[i]_r$ *and* $[i]_c$ *denote the block-row containing* $[S_i, T_i, \phi_i]$ *and block-column containing* $[T_i^T, S_{i+1}^T]^T$*, respectively.*

$$\left[\begin{array}{cccc|c} S_{i-1} & T_{i-1} & & & \phi_{i-1}\\ & S_i & T_i & & \phi_i \\ & \nwarrow & S_{i+1} & T_{i+1} & \phi_{i+1} \\ \hline \multicolumn{4}{l|}{\text{bottom of partition}} & \end{array}\right] \xrightarrow{\text{(a)}} \left[\begin{array}{cccc|c} S_{i-1} & T_{i-1} & -T_{i-1} & & \phi_{i-1} \\ & S_i & T_i - S_i & & \phi_i \\ & & S_{i+1} & T_{i+1} & \phi_{i+1} \\ \hline & & & & \end{array}\right]$$

$$\xrightarrow{\text{(b)}} \left[\begin{array}{cccc|c} S_{i-1} & T_{i-1} & -T_{i-1} & & \phi_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ & & S_{i+1} & T_{i+1} & \phi_{i+1} \end{array}\right] \xrightarrow{\text{(c)}} \left[\begin{array}{cccc|c} S_{i-1} & T_{i-1} & -T_{i-1} & & \phi_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ & \tilde{S}_{i+1} & & T_{i+1} & \tilde{\phi}_{i+1} \end{array}\right]$$

$$\xrightarrow{\text{(d)}} \left[\begin{array}{cccc|c} S_{i-1} & T_{i-1} - S_{i-1} & -T_{i-1} & & \phi_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ & \tilde{S}_{i+1} & & T_{i+1} & \tilde{\phi}_{i+1} \end{array}\right] \xrightarrow{\text{(e)}} \left[\begin{array}{cccc|c} S_{i-1} & T_{i-1}(I + \tilde{S}_i) - S_{i-1} & & & \tilde{\phi}_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ & \tilde{S}_{i+1} & & T_{i+1} & \tilde{\phi}_{i+1} \end{array}\right]$$

$$\xrightarrow{\text{(f)}} \left[\begin{array}{cccc|c} \tilde{S}_{i-1} & I & & & \hat{\phi}_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ & \tilde{S}_{i+1} & & T_{i+1} & \tilde{\phi}_{i+1} \end{array}\right] \xrightarrow{\text{(g)}} \left[\begin{array}{cccc|c} \tilde{S}_{i-1} & I & & & \hat{\phi}_{i-1} \\ & \tilde{S}_i & I & & \tilde{\phi}_i \\ \hat{S}_{i+1} & & & T_{i+1} & \hat{\phi}_{i+1} \end{array}\right]$$

$$\begin{array}{lrcl} \text{(a)} & [i]_c & \longleftarrow & [i]_c - [i-1]_c \\ \text{(b)} & [i]_r & \longleftarrow & (T_i - S_i)^{-1}[i]_r \\ \text{(c)} & [i+1]_r & \longleftarrow & [i+1]_r - S_{i+1}\,[i]_r \end{array} \qquad \begin{array}{lrcl} \text{(d)} & [i-1]_c & \longleftarrow & [i-1]_c - [i-2]_c \\ \text{(e)} & [i-1]_r & \longleftarrow & [i-1]_r + T_{i-1}\,[i]_r \\ \text{(f)} & [i-1]_r & \longleftarrow & (T_{i-1}(I + \tilde{S}_i) - S_{i-1})^{-1}[i-1]_r \\ \text{(g)} & [i+1]_r & \longleftarrow & [i+1]_r - \tilde{S}_{i+1}\,[i-1]_r \end{array}$$

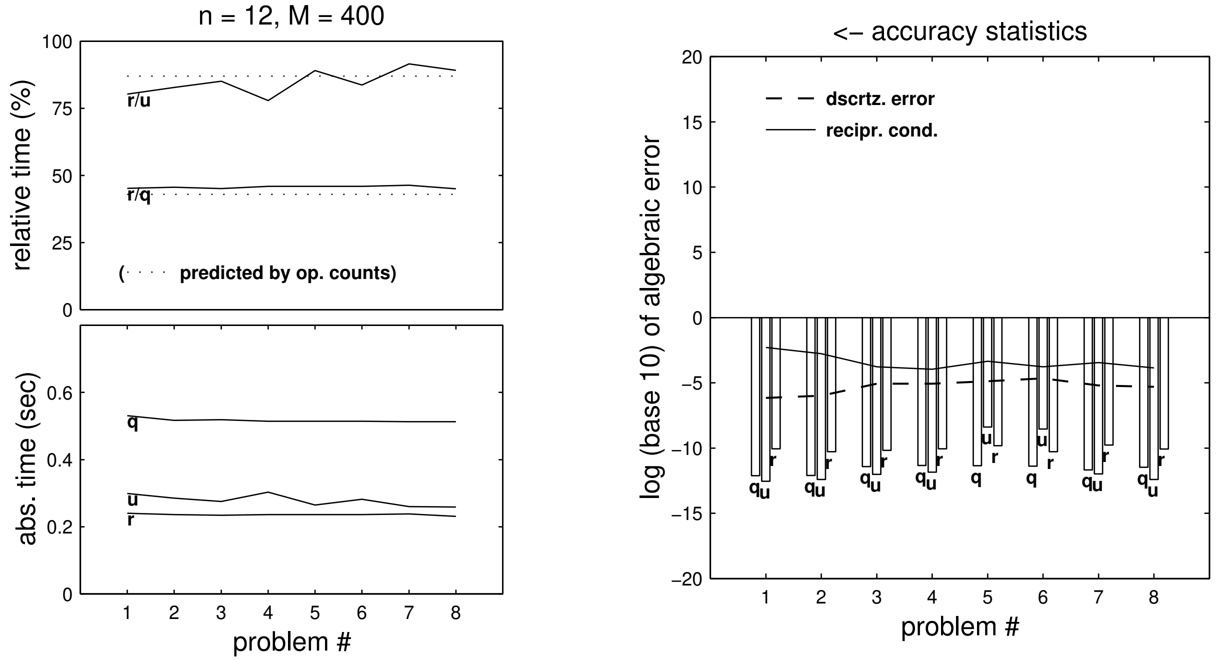
The pivoting strategy represented by $P_j, j = 1, \ldots, n$ must control the growth of the underlying increasing fundamental solution modes to ensure stability. This is discussed further in [9], [13], and in the next section of this paper.

In RSCALE (Figure 1), local transformation starts at the *bottom* of each partition and moves upward. Steps (a) and (d) each involve a change of mesh variable which is later reversed. Steps (b) and (f) are each implemented using LU-factorization followed by $n + 1$ back-solves; matrix inversion is shown for ease of notation only. Steps (d), (e), (f) and (g) are applied to successive block-row triples $[i\text{–}2, i\text{–}1, i\text{+}1]_r$, $[i\text{–}3, i\text{–}2, i\text{+}1]_r$, $\ldots$, until the top of the partition is reached. As with *SLF*-QR and *SLF*-LU, the first-level factorization effectively uncouples the mesh variables associated with the block-rows at the bottom of each partition. The second-level factorization is similar to that of *SLF*-QR and *SLF*-LU.

Repetitive matrix multiplication in step (g) does *not* result in excessive growth of the underlying increasing fundamental solution modes. RSCALE controls the growth by rescaling the eigenvalues λ_j of $\tilde{S}_{i-1}$ in step (f) so that $\max_j |\lambda_j| \in (0, 1+\epsilon)$ where ϵ can be made arbitrarily small. A stability proof for certain classes of ABD matrices is given in [9].

Local operation counts for *SLF*-QR, *SLF*-LU and RSCALE are $7\frac{2}{3}n^3 + \mathcal{O}(n^2)$, $3\frac{5}{6}n^3 + \mathcal{O}(n^2)$ and $3\frac{1}{3}n^3 + \mathcal{O}(n^2)$ flops per block-row, respectively. If $T_i = I, i = 0, \ldots, M-1$, matrix multiplication in step (e) of Figure 1 is not required, and RSCALE uses only $2\frac{1}{3}n^3 + \mathcal{O}(n^2)$ flops per block-row. ABD systems of this form arise naturally, for example, if (2) is based on multiple shooting or a standard Runge-Kutta method [1]. The performance of *SLF*-LU is somewhat dependent on its pivoting strategy. On problems where few cross-block pivots are made, the local flop count may be slightly less than stated above. Unfortunately, *SLF*-LU is often unstable on these problems. See [9] for further discussion of these issues and a description of the FORTRAN implementation of the three

Figure 2: *Timing and accuracy of single-partition RSCALE (**r**),* SLF-*LU (**u**) and* SLF-*QR (**q**). Eight linear problems are solved on a 167MHz Sun SPARCstation Ultra 2.*



algorithms. The next section of this paper gives numerical examples where the relative speeds of the three algorithms agree very well with what one would expect from the operation counts listed above.

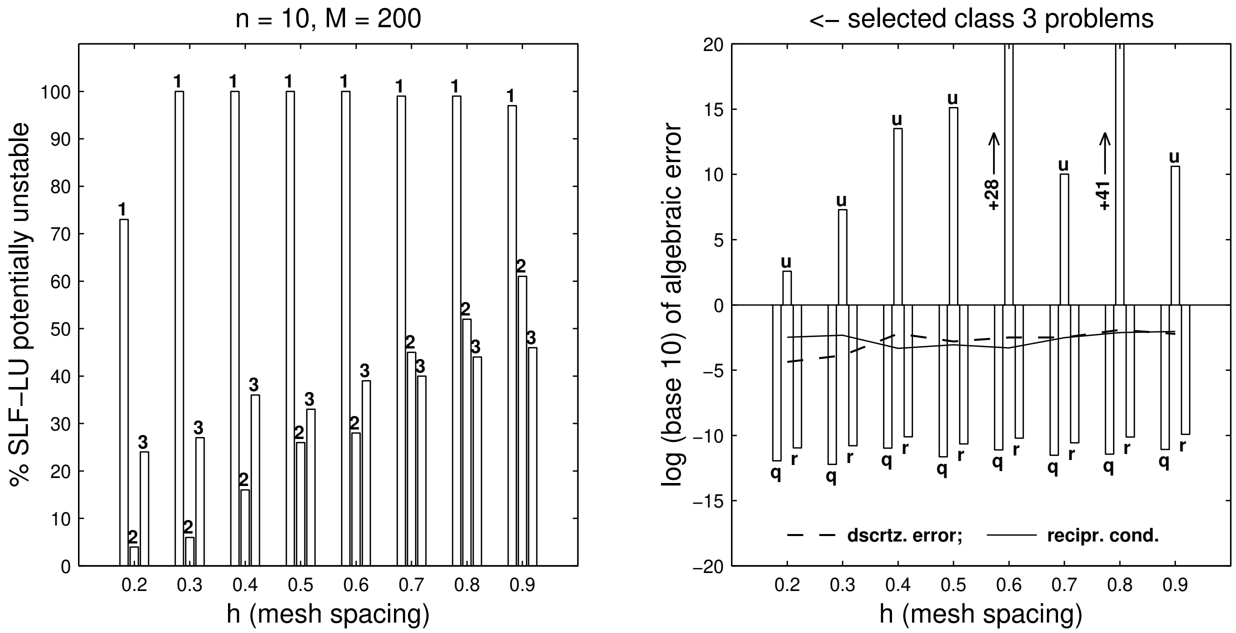
The three algorithms also differ slightly in the amount of fill-in generated during factorization. *SLF*-QR and *SLF*-LU each require an additional $2n^2$ elements per block-row, whereas RSCALE requires only n^2. The extra storage in RSCALE is used to keep a copy of T_i so that subsequent right-hand sides may be processed when $\mathcal{J}$ is held fixed for several iterations in (3). If $T_i = I$, this extra storage is not required. In general, the local transformation used in *SLF*-QR and *SLF*-LU fills-in the identity if $T_i = I$, so these algorithms cannot easily be adapted to take advantage of this special structure.

NUMERICAL RESULTS

The timing and accuracy of the three algorithms run on a sequential machine is illustrated in Figure 2. Eight linear problems are solved using a trapezoidal finite-difference discretization [1]. The graph on the left shows absolute and relative execution times, and the bar graph on the right gives the accuracy of the computed solutions $\tilde{Y}$. In the graph on the right, the algebraic error is measured as $\|\tilde{Y} - Y_b\|_\infty$, where Y_b is the solution obtained with LAPACK's band-solver. The band-solver also returns a condition number estimate for each ABD matrix (solid line). The problems are constructed so that the true solution, Y_t, is known. The discretization error (dashed line) is measured as $\|Y_b - Y_t\|_\infty$. Each algorithm is acceptably accurate on all problems in this figure, given that the algebraic errors are all several orders of magnitude less than the discretization error. Many more numerical examples comparing the timing and accuracy of the three algorithms may be found in [9].

The potential for instability in *SLF*-LU is illustrated in Figure 3. In the graph on the left, 300 randomly-generated linear problems are discretized on each mesh with stepsize h using the trapezoidal finite difference scheme. Each of the resulting ABD matrices is partially factored with *SLF*-LU, and the number of times *SLF*-LU exhibits instability is recorded. The criteria used to detect instability is discussed in [9]. The bar graph on the right shows the accuracy of the three algorithms when used to fully-solve a sample problem selected from each mesh.

Figure 3: *The graph on the left shows the number of times* SLF-*LU exhibits instability on 100 randomly generated class 1 (*$[\partial f/\partial y]_{ij} \in (-1,0] \bigcup [1]$*), class 2 (*$[\partial f/\partial y]_{ij} \in [0,1]$*) and class 3 (*$[\partial f/\partial y]_{ij} \in (-1,1]$*) linear problems. The accuracy of RSCALE (**r**),* SLF-*LU (**u**) and* SLF-*QR (**q**) on selected class 3 problems is shown in the graph on the right.*



Wright [14] has done similar random testing—showing *SLF*-LU instability can occur when there is too little or too much pivoting—but found relatively few problems where *SLF*-LU breaks down. Further analysis in [9] shows that instability can occur frequently when solving linear problems with larger stepsize h, even when the *correct* number of pivots is used, and that this instability can adversely affect the performance of parallel algorithms for solving non-linear problems [5].

The timing and speed-up of the three algorithms run on a parallel machine is illustrated in Figure 4. The execution time of COLROW [4], a commonly-used fast sequential ABD system-solver, is included to provide some indication of how many processors are required before the speed-up achieved from parallelism begins to pay-off.

REFERENCES

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

[2] U.M. Ascher, S.Y.P. Chan, *On parallel methods for boundary value ODEs*, Computing 46/1 (1991), pp. 1-17.

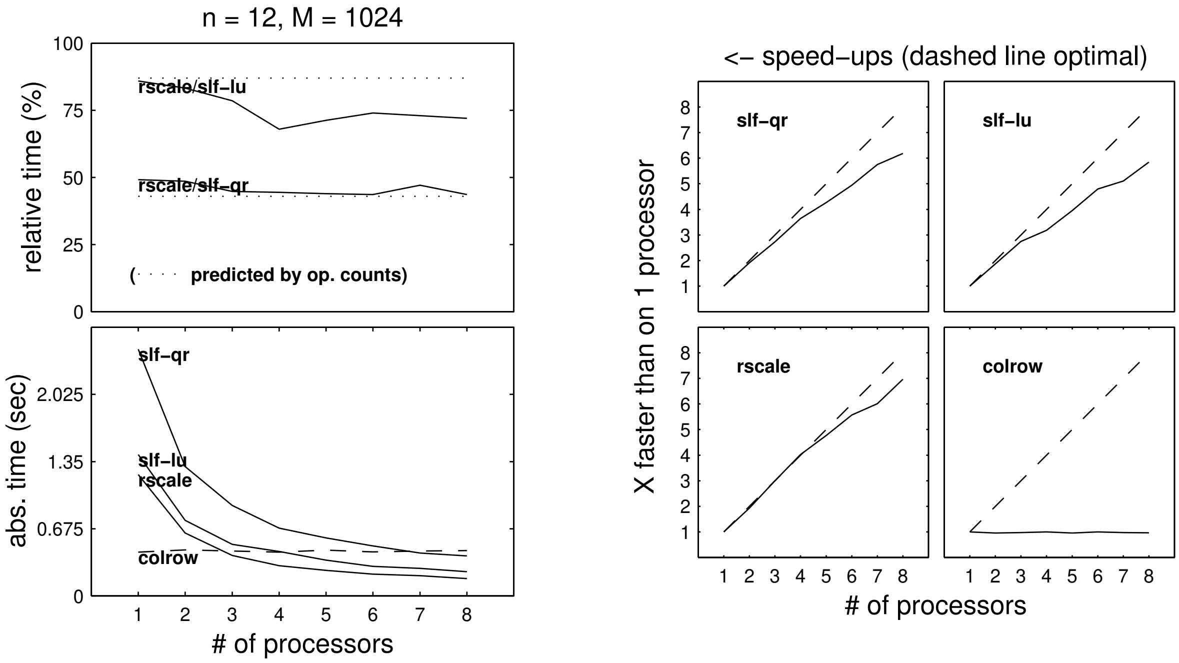
[3] K.R. Bennett, G. Fairweather, *PCOLNEW: A parallel boundary-value ODE code for shared-memory machines*, Univ. of Kentucky, Center for Comp. Sciences, T.R. CCS-90-8, 1990.

[4] J.C. Diaz, G. Fairweather, P. Keast, *Algorithm 603. COLROW and ARCECO: FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software 9/3 (1983), pp. 376-380.

[5] W.H. Enright, P.H. Muir, *Runge-Kutta software with defect control for boundary value ODEs*, SIAM J. Sci. Comput. 17/2 (1996), pp. 479-497.

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

Figure 4: *Timing and speed-up of multi-partition RSCALE,* SLF-*LU and* SLF-*QR. A single linear problem is solved on an SGI Challenge L with eight 150 MHZ R4400 processors. The timing of COLROW, a commonly-used fast sequential ABD system-solver, is included for reference.*



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

[8] R.N. Pancer, K.R. Jackson, *The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs*, Univ. of Toronto, Dept. of Computer Science, T.R. 255/91, 1992.

[9] R.N. Pancer, *The parallel solution of ABD systems arising in numerical methods for BVPs for ODEs*, Ph.D. thesis, Univ. of Toronto, Dept. of Computer Science (in progress).

[10] M. Paprzycki, I. Gladwell, *Solving almost block diagonal systems on parallel computers*, Parallel Comput. 17/2 (1991), pp. 133-153.

[11] S.J. Wright, V. Pereyra, *Adaptation of a two-point boundary value problem solver to a vector-multiprocessor environment*, SIAM J. Sci. Stat. Comput. 11/3 (1990), pp. 425-449.

[12] S.J. Wright, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Stat. Comput. 13/3 (1992), pp. 742-764.

[13] S.J. Wright, *Stable parallel elimination for boundary value ODEs*, Numer. Math. 67/4 (1994), pp. 521-535.

[14] S.J. Wright, *A collection of problems for which Gaussian elimination with partial pivoting is unstable*, SIAM J. Sci. Comput. 14/1 (1993), pp. 231-238.