PMIRKDC: a Parallel Mono-Implicit Runge-Kutta code with Defect Control for Boundary Value ODEs *

P.H. Muir† R.N. Pancer‡ K.R. Jackson§

February 18, 2003

Abstract

We describe parallel software, PMIRKDC, for solving boundary value ordinary differential equations (BVODEs). This software is based on the package, MIRKDC, which employs mono-implicit Runge-Kutta schemes within a defect control algorithm. The primary computational costs involve the treatment of large, almost block diagonal (ABD) linear systems. The most significant feature of PMIRKDC is the replacement of sequential ABD software, COLROW, with new parallel ABD software, RSCALE, based on a parallel block eigenvalue rescaling algorithm. Other modifications involve parallelization of the setup of the ABD systems and solution interpolants, and defect estimation. Numerical results show almost linear speedups.

Keywords: Parallel software, boundary value ordinary differential equations, almost block diagonal linear systems, Runge-Kutta methods.

AMS Subject Classifications: 65L10, 65Y05

1 Introduction

In this paper we consider the development of software for the efficient numerical solution of boundary value ordinary differential equations (BVODEs) on parallel computers based on shared memory architectures. We shall assume BVODEs which have the general form

$$\underline{y}'(t) = \underline{f}(t, \underline{y}(t)) \tag{1}$$

with boundary conditions in separated form

$$\underline{g}_a(\underline{y}(a)) = \underline{0}, \quad \underline{g}_b(\underline{y}(b)) = \underline{0}, \tag{2}$$

*This work is supported by the Natural Sciences and Engineering Research Council of Canada, the Information Technology Research Center of Ontario, and Communications and Information Technology Ontario.

†Department of Mathematics and Computing Science, Saint Mary's University, Halifax, Nova Scotia, Canada B3H 3C3, (Paul.Muir@stmarys.ca).

‡Division of Physical Sciences, University of Toronto at Scarborough, Scarborough, Ontario, Canada M1C 1A4,(pancer@cs.toronto.edu)

§Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4, (krj@cs.toronto.edu)

where $y : \Re \to \Re^n$, $f : \Re \times \Re^n \to \Re^n$, $g_a : \Re^n \to \Re^p$ and $g_b : \Re^n \to \Re^q$, with $p + q = n$. Such problems arise in a wide variety of application areas; see [7] for further details and examples.

There has been a considerable history of high quality software for the efficient numerical solution of (1), (2). The packages can be roughly divided into those based on initial value methods and those based on global methods. In the former group, most codes are based on some form of multiple shooting; one subdivides the problem interval with a set of breakpoints, and considers an associated initial value ordinary differential equation (IVODE) (with estimated initial conditions) on each subinterval; the IVODEs are solved to a given user provided tolerance using high quality IVODE software. The IVODE solutions are then evaluated at the breakpoints and improved estimates of the initial conditions are computed, within the context of a Newton iteration. A refinement of the mesh of breakpoints is then performed to provide a better adaptation of the mesh to solution behavior. The process terminates when the IVODE solutions match at the breakpoints to within the user tolerance. See, e.g., [7], Chapter 4, for further details.

Within the class of global methods, the general algorithmic approach is to subdivide the problem interval with a set of meshpoints and then apply a numerical discretization method, e.g., a finite difference method, to replace the ODEs with a system of algebraic equations, which are then solved with Newton's method. An estimate of the error is then used to assess solution quality and determine a better adapted mesh, and the process repeats until the estimated error satisfies the user tolerance. See, e.g., [7], Chapter 5, for further details. This class includes finite-difference, Runge-Kutta, and collocation methods.

In both the groups of software, the dominant computational effort involves the solution of a large system of nonlinear equations by a Newton iteration. Because the computations can be localized within each subinterval, the Newton matrices generally have a sparsity structure which has been referred to in the literature as *almost block diagonal* (ABD) [15]. The excellent survey [3] provides an extensive review of ABD systems, including contexts in which they arise and methods for solving them. A number of software packages for the efficient factorization and solution of linear systems for which the coefficient matrix has an ABD structure have been developed over the last 25 years.

In §2, we will briefly review sequential software for the numerical solution of BVODEs and for the efficient treatment of ABD systems. This section also includes a brief discussion of more general BVODE software for the efficient treatment of a sequence of BVODEs arising in a *parameter continuation* framework.

Parameter continuation computations, since they involve repeated use of a BVODE code, can be very computationally intensive. As well, the application of the transverse method of lines in the numerical solution of systems of parabolic partial differential equations (see, e.g., [7]) can lead to large BVODE systems which require substantial computational expense. Since parallel computers have become commonly available, it has become natural to consider the efficient numerical solution of BVODEs on such architectures. A brief investigation of the relative costs of the principal algorithmic components of a typical BVODE software package reveals that the only significant computations which cannot be parallelized in a straightforward fashion are the factorization and solution of the ABD linear systems. The last 10-15 years have seen considerable effort expended in the search for efficient and stable parallel algorithms for ABD systems and a substantial number of papers have been written on the subject; we briefly review this work in §3. *Despite the relatively large effort that has been invested in the development of parallel ABD algorithms, relatively little*

work has been done on the subsequent development of parallel software for BVODEs, employing the new parallel ABD algorithms; we survey previous work in the development of parallel BVODE software in §4.

The goal of our paper is to describe the development of a parallel BVODE solver based on the MIRKDC software package [35]. A principal component of this modification is the replacement of the sequential ABD system solver, COLROW [26], [27], with a new parallel ABD system solver, RSCALE [43], [65], [64]. In §5, we briefly describe the underlying algorithms and provide some implementation details for the MIRKDC package and the RSCALE code. In §6, we briefly discuss the modifications to the MIRKDC package required in order to enable parallel execution of the major algorithmic components. In §7, we identify the parallel architectures and test problems to be employed and then present and discuss results from several numerical experiments which demonstrate that nearly optimal linear speedup in the number of processors is attained in the parallel MIRKDC code. During our experimentation, it became apparent that in some instances the use of the RSCALE algorithm leads to performance improvements, *observable even on a sequential computer*, which appear to be attributable to the better stability of the RSCALE algorithm. We report on these, also in §7. Our paper closes with §8 in which we present our conclusions and identify areas for future work.

2 Sequential BVODE and ABD Software

2.1 BVODE Software

As mentioned earlier, software for BVODEs can be roughly divided into two groups: initial value methods (multiple shooting) and global methods.

The text [7] mentions a number of software packages based on multiple shooting for nonlinear BVODEs which have been developed over the last 25 to 30 years. This list includes the code of Bulirsch, Stoer, and Deuflhard, and the MUSN code of Mattheij and Staarink. Among the currently available packages based on multiple shooting are the BVPMS code of IMSL [42], the codes D02HAF, D02HBF, D02AGF, and D02SAF from the NAG library [60] , and the MUSN code, available from the ODE collection of netlib [61].

In the group of packages based on global methods, upon which we shall largely focus for the remainder of this paper, among the earliest codes are the PASVAR routines, e.g., PASVA3, PASVA4, [44], [45], of Lentini and Pereyra, which are based on the trapezoidal finite difference scheme coupled with an iterated deferred correction algorithm. Another early code is the COLSYS package [5], [6], by Ascher, Christiansen, and Russell, which is based on collocation and which employs the B-spline package [15] of deBoor for the representation of the piecewise polynomials upon which the collocation method is based. A later modification of the COLSYS code called COLNEW, [10], developed by Bader and Ascher, replaces the B-spline package with an implementation based on monomial splines. This code also performs a local treatment (for each subinterval) of some of the linear algebra computations, using a "condensation" process.

Runge-Kutta methods, see, e.g., [20], originally developed for the solution of IVODEs, have also served as the basic methods in several codes for BVODEs. In fact, the trapezoidal method of the PASVAR codes and collocation methods, for (1), are equivalent to Runge-Kutta methods. A modification of the PASVA3 code which employs mono-implicit Runge-Kutta (MIRK)

methods, [21], in a deferred correction algorithm is the code ABVRKS [39] of Gupta. The TWPBVP code [22], [23], developed by Cash and Wright, is also based on MIRK methods and a deferred correction algorithm. MIRK methods are employed in a defect control algorithm in the code, MIRKDC [35], of Enright and Muir. The MATLAB problem solving environment [52] also includes a BVODE solver, bvp4c, [50] by Kierzenka and Shampine, based on a MIRK method and using defect control. The IMSL library includes the finite difference code BVPFD (based on PASVA3/PASVA4). The NAG library includes the codes D02GAF, D02RAF (based on PASVA3/PASVA4), and D02TKF (based on COLSYS/COLNEW). The ODE collection at netlib includes the COLSYS and COLNEW codes and the TWPBVP and MIRKDC codes. The COLSYS code is also available in the ACM Collected Algorithms library [9]. All of the above mentioned codes are implementations in Fortran for sequential computers.

2.2 ABD Software

In this paper, we are interested in a special case of the ABD system structure which generally arises within almost all of the codes mentioned above, i.e., the multiple shooting, Runge-Kutta, collocation (with condensation), and finite difference codes, when they are applied to (1),(2). The Newton matrix has a block row structure in which all blocks in a row are zero except for those along the bi-diagonal, which we will refer to as L_i and R_i, for the ith row of blocks. That is, the Newton matrix has the structure,

$$\begin{bmatrix} B_a & & & & & \\ L_1 & R_1 & & & & \\ & L_2 & R_2 & & & \\ & & \ddots & \ddots & & \\ & & & L_{Nsub} & R_{Nsub} & \\ & & & & B_b & \end{bmatrix}, \tag{3}$$

The blocks L_i and R_i are associated with the application of a numerical method on the ith subinterval and the blocks B_a and B_b are associated with the derivatives of the boundary conditions. The L_i and R_i blocks are $n \times n$; the B_a block is $p \times n$; the B_b block is $q \times n$. *Nsub* is the number of subintervals. A matrix with the structure exhibited in (3) is referred to in [3] as being in "standard ABD form".

There is a considerable history of high quality software for ABD systems, for sequential computers; see [3]. The earliest of these packages SOLVEBLOK [16], [17], developed by DeBoor and Weiss, employs Gaussian elimination with partial pivoting, taking into account the ABD structure of the matrix. The stability of this algorithm follows from standard results for Gaussian elimination with partial pivoting for banded systems. This algorithm, however, introduces fill-in and thus requires extra storage and computation. Fill-in can be eliminated and the computation can be made more efficient if a more sophisticated elimination algorithm is employed; the modified alternate row and column elimination algorithm of Diaz, Fairweather, and Keast, is implemented in the ARCECO package [26], [27], (for a general ABD structure of Figure 1.1 of [3]), and in the COLROW package [26], [27], (for a more specialized ABD structure of Figure 1.2 of [3]). The stability of this algorithm follows from the observation that it can be expressed as Gaussian elimination with a restricted form of partial pivoting [70]. A version of ARCECO modified by Brankin

and Gladwell to use the level 2 BLAS [32], [33] is included in the NAG library as F01LHF [18]. The approaches based on multiple shooting or collocation with condensation lead to an ABD matrix structure as in (3) but with the R_i block equal to the identity. In this case it is possible to exploit this extra structure; the modified alternate row and column elimination algorithm is implemented for such matrix systems in the code ABBPACK [54], [55], of Majaess, Keast, Fairweather, and Bennett. The SOLVEBLOK, ARCECO, COLROW, and ABBPACK packages are available through netlib in the ACM Collected Algorithms library. All of these packages are implemented in Fortran for sequential computers.

Because the efficient treatment of the Newton matrices is critical to the overall efficiency of a BVODE, the BVODE codes mentioned earlier employ algorithms specifically designed to handle the ABD matrices which arise. In many cases, the authors of the BVODE software have employed, possibly with some modification, one of the ABD software packages mentioned above. The COLSYS package employs the original SOLVEBLOK code while the COLNEW package employs a modified version of SOLVEBLOK. The NAG collocation code, D02TKF, employs the NAG version of ARCECO, F01LHF. The TWPBVP code employs a modified version of the COLROW package and the MIRKDC code employs the original COLROW package.

On the other hand, in the PASVAR codes, the authors of the BVODE software have also implemented algorithms for handling the ABD systems. In the PASVAR codes, the ABD factorization and backsolve routines are called DECOMP and SOLVE. A feature of these routines is that they can handle matrices that have what is referred to in [3] as a bordered ABD (BABD) structure; such matrices arise from the application of any of the numerical methods discussed above to the BVODE, (1), with nonseparated boundary conditions

$$\underline{g}(\underline{y}(a), \underline{y}(b)) = \underline{0}, \tag{4}$$

where $g : \Re^n \times \Re^n \rightarrow \Re^n$. The factorization algorithm employed in DECOMP/SOLVE is of alternate row and column elimination type, similar to that of [26], [27], with fill-in along the bottom block row associated with the presence of the nonseparated boundary conditions. See [79] for a detailed description of the underlying algorithms. Although all of the above codes except the PASVAR codes and others based on them can only handle separated boundary conditions, (2), directly, it is possible to recast a BVODE to remove the unseparated boundary conditions, a process which can double the number of ODEs to be treated and thus substantially increase the cost of solving the recast system; see [8]. The standard BABD structure is

$$\begin{bmatrix} D_a & & & & D_b \\ L_1 & R_1 & & & \\ & L_2 & R_2 & & \\ & & \ddots & \ddots & \\ & & & L_{Nsub} & R_{Nsub} \end{bmatrix}, \tag{5}$$

where D_a and D_b are $n \times n$ blocks associated with the derivatives of the boundary conditions. The stability of methods for BABD systems can be much different than for ABD systems: the paper [77] demonstrates with some simple examples that Gaussian elimination with partial pivoting can be unstable for BABD systems.

An interesting comment on the treatment of BABD systems is provided by Kraut and Gladwell in [51]; after separating the boundary conditions (through the introduction of extra unknowns

and ODEs), the resultant BVODE leads to an ABD system rather than a BABD which can then be handled efficiently and in a stable fashion by a Gaussian elimination type algorithm (such as alternating row and column elimination). The authors compare the cost of this approach with the cost of applying a (stable) QR factorization directly to the original BABD system, and show that both approaches have approximately the same cost.

2.3 BVODE Parameter Continuation Software

For difficult parameter dependent BVODEs, having the general form

$$\underline{y}'(t) = \underline{f}(t, \underline{y}(t), \underline{\lambda}), \tag{6}$$

with parameter $\underline{\lambda} \in \Re^l$, it is common to employ a BVODE code within a "parameter continuation" algorithm — see, e.g., [7] and references within. To obtain the solution to a problem corresponding to a difficult parameter value, one solves a sequence of BVODEs beginning with a simpler problem and proceeding through several parameter values until the difficult parameter range is reached, using the final mesh and solution from one problem as the initial mesh and initial solution estimate for the next. In its simplest form, the selection of the sequence of parameter values is done by the user and the BVODE code is simply called within a loop. In some cases a code will have a built-in parameter continuation feature and the selection of the parameter sequence is chosen automatically within the code. A recent code of this type is COLMOD [24] by Cash, Moore, and Wright; the continuation algorithm obtains an estimate of the next parameter value based on linear or quadratic interpolation from previous parameter values, coupled with a variety of heuristics. This code employs COLNEW (with a modified mesh selection algorithm [75]) but the parameter continuation implementation is external to the COLNEW code and thus the linear algebra software components are the same as in COLNEW, i.e., a modified SOLVEBLOK package. The same parameter continuation algorithm is implemented in the code ACDC [24] also by Cash, Moore, and Wright, which uses TWPBVP (with the mono-implicit Runge-Kutta formulas replaced by Lobatto collocation formulas [12]) to solve each BVODE problem in the sequence. Again the continuation algorithm is external to the modified TWPBVP package and thus the same linear algebra software, i.e., a modification of the COLROW package, is employed. The COLMOD and ACDC packages are available through the ODE library of netlib. Automatic parameter continuation is also available in the PASVA3/PASVA4 codes.

In standard parameter continuation, the purpose of solving the sequence of problems associated with different parameter values is simply to get a sufficiently good initial mesh and initial solution estimate so that the final, most difficult problem can be successfully solved. A generalization of the application of parameter continuation is in homotopy path following, where one wishes to follow solution values for a path of parameter values, or in bifurcation analysis, where one considers multiple solutions for a BVODE with a parameter. See, e.g., [7] and references within. A well-known BVODE code of this type is AUTO (see [29] and references within) by Doedel et al. Because of the presence of parameters and the possibility of nonseparated boundary conditions, the matrix structure that arises is BABD with an extra column of blocks of width l associated with the additional parameters $\underline{\lambda}$. The linear algebra algorithm employed in AUTO is based on a structured Gaussian elimination (LU) algorithm with partial pivoting. An interesting aspect of this algorithm is that a similar version was later presented for use in the *parallel* solution of ABD systems by

Wright in [78]. In [46], Liu and Russell replace the structured LU algorithm with a structured QR algorithm, previously considered by Wright in [76] for use in the parallel solution of ABD systems. They report improved stability associated with the use of the latter algorithm. The AUTO software is available from its primary author [28]. Another path following code based on the collocation algorithm employed in COLNEW, is the COLCON code [11] of Bader and Kunkel; it employs a further modification of the SOLVEBLOK code to handle the linear systems that arise.

3 Parallel Algorithms for ABD Systems

Over the last 15 years, a number of papers by various researchers have addressed the question of developing efficient parallel algorithms for the factorization and solution of ABD systems. Many of these algorithms allow a bordered ABD (BABD) matrix. An extensive survey of parallel algorithms for ABD and BABD systems is provided in [3].

3.1 Algorithms for Vector Processors

About two decades ago, commonly available "parallel" computers were limited to those which had a vector processing capability and thus the earliest papers in the parallel treatment of ABD systems considered algorithms based on vector processors. In [38], Goldmann describes a vectorized version of a cyclic reduction implementation of a "condensation" method of Bulirsch (not the same as the condensation employed in COLNEW) for BABD systems, modified to improve its stability, since the original algorithm appears to be related to the well-known compactification method, which is know to be unstable — see, e.g. [46]. In [40], Hay and Gladwell develop and test vectorized versions of the SOLVEBLOK, ARCECO, and COLROW codes and show that significant improvements in execution time are attained by the vectorized codes. In a related paper [36], these same authors consider the reductions in linear algebra costs associated with replacing the linear algebra packages in COLSYS and COLNEW with ARCECO, including consideration of vectorizability within the linear algebra computations. Both from an analysis of operation counts and from numerical experiments, they show that the use of COLSYS or COLNEW with the vectorized ARCECO code is superior to the use of these codes with the vectorized SOLVEBLOK or modified SOLVEBLOK routines. The paper also considers possible conflicts between vectorization and parallelization of a BVODE code. In [79], Wright and Pereyra, investigate several vector-multiprocessor algorithms for the efficient treatment of BABD systems arising in PASVA4. For larger systems of BVODEs, the vector capabilities of the new algorithms are better demonstrated, and the algorithms are shown to improve upon the original DECOMP/SOLVE routines. However, the underlying parallel algorithm is essentially equivalent to compactification and thus will be unstable for some problems.

3.2 Algorithms based on Level 3 BLAS

The presence of multiple processors is exploited at a lower level of granularity in the papers [37], [67], by Gladwell and Paprzycki, which describe algorithms for the treatment of ABD systems using level 3 BLAS [30],[31]; parallelism is achieved through the parallel execution of the computations inside the BLAS routines. On a Cray X-MP, the authors show that parallelism at this

level can lead to speedups of about 4 or 5 when up to 40 processors are employed; thus this approach may be appropriate primarily for solving large BVODE systems on architectures with few processors.

3.3 Algorithms for Parallel Architectures

In addition to [79], another early attempt at the development of a parallel algorithm for the treatment of ABD or BABD systems was that of Ascher and Chan [4]. By first forming the normal equations corresponding to the original ABD system, Ascher and Chan obtained a positive definite, block tridiagonal system which can then be solved, stably, using standard block cyclic reduction (which is not stable for general linear systems since it does not include pivoting). Of course the difficulty with this approach is that the 2-norm condition number of the linear system to be solved is potentially squared. Another algorithm, considered by Paprzycki and Gladwell [66], involves "tearing" the ABD system into a sequence of smaller, independent ABD systems with the same structure, each of which is solved on a different processor, using the sequential ABD solver F01LHF. The authors report speedups of only 3 or 4 using up to 40 processors. An additional difficulty with this approach is that the conditioning of the smaller ABD systems may be poor even when the original ABD system is well-conditioned. A more recent followup to this work is that of Amodio and Paprzycki [1] in which an improved version of the algorithm of [66] is described. The improved algorithm features a reduction in the number of operations and in the amount of fill-in and is specifically directed to distributed memory parallel architectures. The reported potential speedups improve on those of [66] but are still substantially less than linear in the number of processors.

The first two algorithms to be both stable and exhibit almost linear speedups in the number of processors were published by Wright [76], [78]; each partitions the BABD matrix into collections of contiguous block rows; the first, the structured QR (SQR) algorithm, employs local QR factorizations within each partition and can be proven to be stable; the second, the structured LU (SLU) algorithm, uses local LU factorizations within each partition, exhibits stable behavior for many problems (stability can be proven under certain assumptions), and involves about half the computational cost of the former. Wright followed up the second of these papers with [77], commenting on the possibility of the SLU algorithm exhibiting unstable behavior on simple BABD systems. These SQR and SLU algorithms were also developed independently by Jackson and Pancer [43], and subsequent work by these researchers has further investigated the potential for instability within the SLU algorithm [64], [65]. An algorithm essentially equivalent to the SLU algorithm was also proposed around this same time by Wright [74]; it employs local LU factorizations with partial pivoting within a block cyclic reduction algorithm rather than within a partitioning algorithm. In [2], Amodio and Paprzycki examine the standard block cyclic reduction algorithm for ABD systems and describe a "stabilized" version which uses block LU factorizations; this algorithm is related to the SLU algorithm and the authors base an argument for its stability on the stability of the SLU algorithm. The authors report better performance than the SLU algorithm when the number of BVODEs is large.

An algorithm which improves upon the SQR algorithm is described by Mattheij and Wright in [53]; a key idea in this approach is to explicitly consider the presence of non-increasing and non-decreasing fundamental solution modes associated with the BVODE system from which the ABD

systems arise. An explicit decoupling of the columns of each ABD block corresponding to the two modes leads to a stable version of the compactification algorithm [7]. Mattheij and Wright describe a parallel version of this algorithm which partitions the ABD systems over the available processors and then applies stabilized compactification on each partition. The decoupling step, in addition to using a QR factorization, also employs a generalized SVD decomposition. (The authors note that for some problems, it may be necessary to reapply the decoupling step several times.) The factorizations employed within the stabilized compactification algorithm are QR factorizations, and in fact, the SQR algorithm is used on the reduced ABD system that arises after the parallel compactification phase. The new algorithm is reported to have the same stability as the SQR algorithm but with the computational costs of the SLU algorithm. (We note, however, that a careful implementation of the SLU algorithm can have a computational cost of about $\mathcal{O}(4n^3)$ — see [43], pg. 25 — rather than the $\mathcal{O}(\frac{23}{3}n^3)$ cost reported in [53], making the SLU algorithm substantially less computationally expensive than the parallel compactification algorithm of [53].)

Another recently proposed algorithm which is related to the SQR algorithm is that of Osborne [63], in which a stabilized cyclic reduction algorithm based on the use of local QR factorizations is discussed. This was followed by the paper of Hegland and Osborne [41] in which local orthogonal factorizations (i.e., QR factorizations) are employed in a novel partitioning algorithm, described by the authors as a generalization of cyclic reduction partitioning, and called wrap-around partitioning. The basic idea is to reorder the block rows and columns of the original BABD so that the blocks appearing in each lower diagonal partition are independent of each other and can thus be treated in parallel. After the factorization phase, the resultant system can also be treated with wrap-around partitioning, allowing for recursive use of this algorithm. The algorithm has the same stability as the SQR algorithm; it also appears to have the same operation count as the SQR algorithm but may have advantages for architectures with vector capabilities.

Jackson and Pancer have developed a new algorithm called eigenvalue rescaling [43], [65], which in terms of stability, compares well with the SQR algorithm — see [64], and which has the computational costs somewhat better than those of the SLU algorithm. (As mentioned earlier, a detailed analysis shows that a precise implementation of the SLU algorithm has a cost that is $\mathcal{O}(4n^3)$; the new eigenvalue rescaling algorithm has a cost which is $\mathcal{O}(3\frac{1}{3}n^3)$ or, in some cases, as low as $\mathcal{O}(2\frac{1}{3}n^3)$.) It is implemented in the RSCALE package and will be discussed further later in this paper.

4 Parallel Software for BVODEs

4.1 Parallel Implementations for Multiple Shooting

In one of the earliest papers in this area, Goldmann [38] considers the development of a vectorized multiple shooting algorithm. This includes identification of algorithms for mesh selection, integration of the local IVODEs, and a modified Newton iteration, that are appropriate for vector machines. The new code is compared, in both sequential and vector mode, with several sequential BVODE codes; the numerical results show that the new code run in sequential mode compares well with the best performances of the other sequential codes and that in vector mode it achieves speedups of about 3. Another early investigation of parallel implementations for multiple shooting was conducted by Keller and Nelson [48], [49], in which they considered hypercube parallel archi-

tectures. After identifying the two major computational phases, integration of the local IVODEs and factorization and solution of the ABD system, they consider two approaches to parallelization: domain decomposition, which is the conventional approach of distributing the work associated with each subinterval across the available processors, and column decomposition, which is an alternative approach in which work associated with corresponding columns of the fundamental solution matrices for each subinterval in the IVODE phase and work associated with corresponding columns for each block of the ABD matrix are treated by the same processor. The authors do not consider parallel treatment of the ABD factorization and solution within the domain decomposition; a sequential alternate row and column elimination algorithm is assumed. The paper includes a careful analysis of efficiency for each approach, for each phase.

The paper [73] by Womble, Allen, and Baca, discusses a parallel algorithm for the method of invariant imbedding (which is related to multiple shooting since the solution of a linear BVODE is computed via the solution of a number of IVODEs). They also consider the parallel treatment of partial differential equations using a (transverse) method-of-lines approach, which leads to a system of BVODEs which can be treated using their parallel algorithm. The paper [56] by Miele and Wang describes a method for the parallel solution of linear BVODEs based on the method of particular solutions, which is again related to multiple shooting since it obtains the solution of the BVODEs through a partitioning of the original problem interval followed by the solution of IVODEs on each subinterval.

A more recent effort by Chow and Enright [25] considers standard multiple shooting for distributed memory parallel architectures. The BABD systems which arise are treated with a parallel algorithm which appears to be related to compactification and may therefore have stability difficulties. The authors also acknowledge that standard multiple shooting can have stability problems due to the presence of rapidly increasing fundamental solution modes, which can cause difficulties for the underlying IVODE solver. To address these stability difficulties, the authors introduce a parallel iterative refinement scheme, which is shown through numerical examples to provide improved performance, counteracting, to some extent, the negative performance effects associated with the stability difficulties.

4.2 Parallel Chopping Algorithms

A number of authors have considered approaches which generalize standard multiple shooting by considering *local* BVODEs on each subinterval rather than local IVODEs. The local BVODEs must, of course, then be provided with appropriate local boundary conditions.

Ascher and Chan discuss in [4] a parallel algorithm involving local (stable) BVODEs for which the local boundary conditions are chosen to be consistent with the increasing and decreasing fundamental solution modes of the global BVODE. Some preliminary work in the development of a practical algorithm for the determination of the local boundary conditions is discussed.

Paprzycki and Gladwell report in [68] on a parallel chopping algorithm based on the sequential chopping algorithm of Russell and Christiansen [71]. Using estimated boundary conditions, the independent BVODEs are treated in parallel using COLNEW. This is coarse-grain parallelism since it involves running the COLNEW code independently on each of several processors, as opposed to executing components of a single run of COLNEW in parallel. Their results show quite substantial speedups on some test problems and suggest the need for further investigation of the

approach. A later paper by Katti and Goel, [47], again considers the parallel chopping algorithm; however in their approach the individual BVODEs are treated using a fourth order finite difference method derived by the authors rather than by using standard BVODE software. The method they derive requires a rather specialized assumption on the BVODE which in general does not hold for nonlinear BVODEs.

As in previous papers, Pasic and Zhang [69] consider a parallel algorithm based on local BVODEs. The authors comment that once the local solutions are obtained the usual approach is to impose continuity constraints globally by considering all the constraints associated with all the subintervals simultaneously; this gives the usual ABD matrix. In [69], the authors present algorithms which attempt to treat the continuity constraints locally, i.e., between pairs of adjacent subintervals, in an iterative context. This allows for an obvious parallel treatment of the continuity constraints. Examples are provided for single second order ODEs but the authors indicate that the approach generalizes to systems.

4.3 Parallelization Across the Method

Several authors have considered approaches which involve parallel execution of significant algorithmic components of existing sequential BVODE codes.

Gladwell and Hay, in [36], show that, for the COLSYS and COLNEW codes, linear algebra computations make a substantial contribution to the overall cost, and highlight the factorization and solution of the ABD systems as particular bottlenecks. Their paper was written prior to the extensive work on parallel algorithms for ABD systems and thus pointed out the need for further research in this area. In the paper by Bennett and Fairweather [14], a parallelization of the COLNEW code is described. They observe that, in COLNEW in sequential mode, the most significant computational cost is in the *setup* of the ABD systems; the factorization and solution of the ABD is actually relatively less important. (This is due to the condensation phase implemented in COLNEW; in COLSYS the factorization phase is relatively more costly.) Hence, the authors focus on the parallel implementation of the setup of the ABD systems. They attain nearly linear speedup on several test problems for the computational components associated with the condensation phase. They report that on systems with even a small number of parallel processors the ABD setup costs can be reduced to the point where the factorization of the ABD system can represent about 50% of the overall execution time, further supporting the need for research in the parallel treatment of the ABD systems. These authors followed up on this effort by modifying COLNEW to obtain the parallel code, PCOLNEW [13], which also included an implementation of the parallel SLU algorithm of Wright [78], mentioned earlier, for the parallel factorization and backsolve of the ABD systems.

Muir and Remington [58] report on a modification of an early version of the MIRKDC code to incorporate parallelization of the ABD system setup, factorization, and solution phases, as well as the interpolant setup and defect estimation phases. The parallel ABD factorization and solution algorithm was based on an implementation of Wright's SLU algorithm. For the ABD setup, interpolant setup, and defect estimation phases, they report nearly linear speedups for most phases; however the speedups reported for the factorization and solution phases do not become apparent until 3 or more processors are employed and do not show as much speedup as do the other phases.

Despite a substantial volume of work in the investigation of the efficient and stable parallel

treatment of ABD systems arising in the numerical solution of BVODEs, one can observe from the above discussion that relatively little has been done in following up this work by implementing any of the stable, parallel ABD algorithms within a BVODE package. The only reported work of this type, to our knowledge, is that mentioned above in [13] and [58].

5 Overview of MIRKDC and RSCALE

5.1 MIRKDC: Mono-implicit Runge-Kutta Software with Defect Control

The underlying algorithms employed in the MIRKDC code are described in detail in [34] and [35]. The numerical schemes used to discretize the ODEs are a special class of Runge-Kutta methods known as mono-implicit Runge-Kutta (MIRK) methods — see [19] and references within. Due to special structure which relates the stages of these methods, they can be implemented, in the BVODE context, more efficiently than standard implicit Runge-Kutta methods. The discretization process leads to a system of nonlinear algebraic equations which are solved by a modified Newton iteration. This gives a discrete numerical solution at the meshpoints which subdivide the problem interval. This solution can be efficiently extended to a continuous approximate solution by augmenting the MIRK methods with continuous MIRK methods [59]. This continuous solution approximation is a C^1 interpolant to the the discrete numerical solution of the whole problem interval. The availability of this high order interpolant allows the MIRKDC code to employ defect control as an alternative to the standard control of a global error estimate usually employed in BVODE codes. For a continuous approximate solution, $\underline{u}(t)$, and the ODE given in (1), the defect is given by, $\underline{\delta}(t) = \underline{u}'(t) - \underline{f}(t, \underline{u}(t))$. The MIRKDC code also employs adaptive mesh redistribution based on an equidistribution of the defect estimates for each subinterval.

The algorithm implemented in the MIRKDC software is as follows:

Initialization

- Initial mesh, $\pi \equiv \{t_i\}_{i=0}^{Nsub}$, partitioning $[a, b]$,
- Initial solution approximation, $Y \equiv \{\underline{y}_i\}_{i=0}^{Nsub}$, at mesh points.

REPEAT

1. Based on current mesh and a MIRK method, discretize ODEs and boundary conditions to obtain a nonlinear algebraic system, $\Phi(Y) = 0$.
2. Compute discrete solution, Y, of $\Phi(Y) = 0$, using Newton's method:

   For $q = 0, 1, \ldots$

   (i) Setup Newton matrix, $\frac{\partial \Phi(Y^{(q)})}{\partial Y}$,

   (ii) Setup residual, $\Phi(Y^{(q)})$,

   (iii) Factor Newton matrix, $\frac{\partial \Phi(Y^{(q)})}{\partial Y}$,

   (iv) Solve Newton system, $\left[\frac{\partial \Phi(Y^{(q)})}{\partial Y}\right] \Delta Y^{(q)} = -\Phi(Y^{(q)})$, for $\Delta Y^{(q)}$,

(v) Update Newton iterate, $Y^{(q+1)} = Y^{(q)} + \Delta Y^{(q)}$.

3. Based on discrete solution, Y, obtained from step 2. and on a CMIRK method, construct interpolant, $\underline{u}(t)$.

4. Estimate defect, $\underline{\delta}(t) = \underline{u}'(t) - \underline{f}(t, \underline{u}(t))$, on each subinterval, to obtain defect estimates, $\{\underline{\delta}_i\}_{i=1}^{Nsub}$.

5. Check size of each defect estimate, $||\underline{\delta}_i||$, for $i = 1, \ldots,$ ***Nsub***:

   If $||\underline{\delta}_i|| < TOL$ then $CONVERGE = TRUE$

   else

   - $CONVERGE = FALSE$
   - Use current mesh, defect estimates, and TOL to construct new mesh, π^*, which equidistributes defect estimates with sufficient refinement to approximately satisfy TOL.
   - Evaluate $\underline{u}(t)$ on new mesh, π^*, to obtain new discrete initial solution approximation, Y^*, at new mesh points.

UNTIL $CONVERGE$

With respect to the above algorithm, note that:

- TOL is the user defined tolerance.
- To improve efficiency, fixed Newton Matrix iterations are sometimes employed, in which case, steps 2.(i) and 2.(iii) are skipped. (Also, damped Newton steps are sometimes employed in 2.(v).)
- The Newton iteration, step 2., terminates when the Newton correction is sufficiently small, e.g., $||\Delta(Y^{(q)})|| < TOL/100$.
- If the Newton iteration (Step 2.) fails to converge, the calculation is restarted with a "doubled mesh", obtained by dividing each subinterval of the current mesh in half.
- The defect is estimated by sampling it at several points per subinterval.
- The algorithm halts without obtaining a solution if a new mesh will require more than the maximum number of subintervals available (defined by the user).

An analysis of the relative execution costs for the components of the above algorithm (when the code is executed on a sequential architecture), shows that only a few of them dominate the computational costs (see §7). These are:

2.(i) Setup of the Newton matrix,

2.(ii) Setup of the residual,

2.(iii) Factorization of the Newton matrix,

2.(iv) Solution of the Newton system,

3. Setup of the interpolant,

4. Computation of the defect estimates.

It is worthwhile to note that for a single full Newton step, the costs of steps 2.(i) and 2.(iii) heavily dominate the costs of the other three steps. However, in a typical run of the MIRKDC code, fixed Newton Matrix iterations will frequently be employed, and this will increase the relative costs of steps 2.(ii) and 2.(iv), since they will be executed more often than steps 2.(i) and 2.(iii). Further analysis of this observation will be provided later in this paper.

5.2 RSCALE: Parallel Block Rescaling and Factorization

Assuming nonseparated boundary conditions (4), the Newton matrix that arises during the q-th iteration in step 2. of the algorithm of the previous subsection has the BABD system structure of (5). Augmenting this matrix with the corresponding right-hand side, we may write the Newton system as

$$\left[\frac{\partial \Phi(Y^{(q)})}{\partial Y} | \Phi(Y^{(q)})\right] = \left[\begin{array}{ccccc|c} D_a & & & & D_b & \underline{g} \\ L_1 & R_1 & & & & \underline{\phi}_1 \\ & L_2 & R_2 & & & \underline{\phi}_2 \\ & & \ddots & \ddots & & \vdots \\ & & & L_{Nsub} & R_{Nsub} & \underline{\phi}_{Nsub} \end{array}\right], \tag{7}$$

where $\underline{g}$ is the vector of boundary conditions and $\underline{\phi}_i$ is the i-th component of $\Phi(Y)$.

Several variations of RSCALE exist for solving (7) and are discussed in [64]. Probably the most suitable variation for architectures with a modest number of processors—the variation implemented in PMIRKDC—is a two-level factorization. In the first level of such a scheme, the last *Nsub* block-rows of (7) are divided into P partitions of *Nsub*$/P$ 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.

Figure 1 illustrates how local transformations are applied within each partition during the first level of the factorization. Transformation starts at the bottom of each partition and moves upward. Steps (a) and (d) incorporate block-column differencing to shift the eigenvalues in the R_i and R_{i-1} blocks, respectively. These are key steps in the rescaling process which improves stability in step (g) of the algorithm. 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 $[i\text{–}2, i\text{–}1, i\text{+}1]_r$, $[i\text{–}3, i\text{–}2, i\text{+}1]_r$, ..., until the top of the partition is reached. When the first-level factorization is complete, the unknown solution values 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 [64]. 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 of $\tilde{L}_{i-1}$

Figure 1: *Steps in the RSCALE local transformation.* $[i]_r$ *and* $[i]_c$ *denote the block-row containing* $[L_i, R_i, \underline{\phi}_i]$ *and block-column containing* $[R_i^T, L_{i+1}^T]^T$*, respectively.*

$$\left[\begin{array}{cccc|c} L_{i-1} & R_{i-1} & & & \underline{\phi}_{i-1} \\ & L_i & R_i & & \underline{\phi}_i \\ & \nwarrow & L_{i+1} & R_{i+1} & \underline{\phi}_{i+1} \\ \hline \multicolumn{4}{l|}{\text{bottom of partition}} & \end{array}\right] \xrightarrow{\text{(a)}} \left[\begin{array}{cccc|c} L_{i-1} & R_{i-1} & -R_{i-1} & & \underline{\phi}_{i-1} \\ & L_i & R_i - L_i & & \underline{\phi}_i \\ & & L_{i+1} & R_{i+1} & \underline{\phi}_{i+1} \\ \hline & & & & \end{array}\right]$$

$$\xrightarrow{\text{(b)}} \left[\begin{array}{cccc|c} L_{i-1} & R_{i-1} & -R_{i-1} & & \underline{\phi}_{i-1} \\ & \tilde{L}_i & I & & \underline{\tilde{\phi}}_i \\ & & L_{i+1} & R_{i+1} & \underline{\phi}_{i+1} \end{array}\right] \xrightarrow{\text{(c)}} \left[\begin{array}{cccc|c} L_{i-1} & R_{i-1} & -R_{i-1} & & \underline{\phi}_{i-1} \\ & \tilde{L}_i & I & & \underline{\tilde{\phi}}_i \\ & \tilde{L}_{i+1} & & R_{i+1} & \underline{\tilde{\phi}}_{i+1} \end{array}\right]$$

$$\xrightarrow{\text{(d)}} \left[\begin{array}{ccccc|c} L_{i-1} & R_{i-1} - L_{i-1} & -R_{i-1} & & & \underline{\phi}_{i-1} \\ & \tilde{L}_i & I & & & \underline{\tilde{\phi}}_i \\ & \tilde{L}_{i+1} & & & R_{i+1} & \underline{\tilde{\phi}}_{i+1} \end{array}\right] \xrightarrow{\text{(e)}} \left[\begin{array}{ccccc|c} L_{i-1} & R_{i-1}(I + \tilde{L}_i) - L_{i-1} & & & & \underline{\tilde{\phi}}_{i-1} \\ & \tilde{L}_i & & I & & \underline{\tilde{\phi}}_i \\ & \tilde{L}_{i+1} & & & R_{i+1} & \underline{\tilde{\phi}}_{i+1} \end{array}\right]$$

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

(a) $[i]_c \longleftarrow [i]_c - [i-1]_c$

(b) $[i]_r \longleftarrow (R_i - L_i)^{-1}[i]_r$

(c) $[i+1]_r \longleftarrow [i+1]_r - L_{i+1}\,[i]_r$

(d) $[i-1]_c \longleftarrow [i-1]_c - [i-2]_c$

(e) $[i-1]_r \longleftarrow [i-1]_r + R_{i-1}\,[i]_r$

(f) $[i-1]_r \longleftarrow (R_{i-1}(I + \tilde{L}_i) - L_{i-1})^{-1}[i-1]_r$

(g) $[i+1]_r \longleftarrow [i+1]_r - \tilde{L}_{i+1}\,[i-1]_r$

in step (f) so that $\|\hat{L}_{i+1}\|$ remains small. A stability proof for certain classes of ABD matrices is given in [64].

The local operation count for RSCALE is $3\frac{1}{3}n^3 + \mathcal{O}(n^2)$ flops per block-row. If $R_i = I, i = 1, \ldots,$ *Nsub*, as occurs in multiple shooting or collocation with condensation, 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.

We note that the requirement that RSCALE perform a stable and parallel factorization results in it having a higher local operation count than a standard sequential ABD factorization package such as COLROW; the corresponding local operation count for COLROW is $\frac{5}{6}n^3 + \mathcal{O}(n^2)$ flops per block-row. A comparison with the above local operation count for RSCALE shows that we can expect single-processor RSCALE to be about 4 times slower than COLROW. (This is verified by numerical experiments reported later in this paper.)

6 Parallelization of the MIRKDC code

Since the six components of the MIRKDC code identified in the previous section dominate the computational costs, it is important that our parallelization efforts focus on them. The parallelization process for each component can be described in terms of partitioning the associated computation and distributing the task over the available processors. Since the parallel architectures employed in the work described in this paper are of shared memory type, the parallelization of each code component requires a careful specification of the partitioning of memory over the processors. Parallel Fortran compiler directives (provided by the Silicon Graphics Parallel Fortran compiler) are used to implement parallelism and memory partitioning specifications. A detailed example of this procedure, for the parallelization of the residual computation, is provided in [57].

The interface for the RSCALE algorithms for the parallel factorization of the Newton matrix and the parallel backsolve of the Newton system are essentially identical to those for the corresponding components of COLROW and thus the replacement of COLROW by RSCALE in MIRKDC is relatively straightforward. However the RSCALE algorithm does require some additional memory as well as knowledge of the number of available processors. This requires some minor modifications to the required size of the work array and to the parameter lists for those MIRKDC routines that call RSCALE routines.

The resultant modified version of MIRKDC, with parallelization of the setup of the Newton matrix, the residual evaluation, the setup of the interpolant, the defect estimation, and the ABD linear system factorization and solution (by RSCALE), will be referred to as PMIRKDC. The original sequential version of MIRKDC (in which the ABD linear systems are factored and solved by COLROW) will continue to be referred to as MIRKDC.

7 Investigation of PMIRKDC

We have conducted substantial testing of PMIRKDC on several computer architectures and test problems. A full description of all the testing is provided in [57]. In this section we discuss selected results from these tests.

7.1 Parallel Architectures and Test Problems

We have employed three computer architectures in our work. The specifications for the three systems are given in Table 1. The Sun Ultra 2 is used as a sequential machine.

Table 1: Architecture specifications. (Vendors: SG:Silicon Graphics, SM:Sun Microsystems)

Architecture			Specifications			
			processors			memory
acronym	model	vendor	#	speed	type	
CHA	Challenge L	SG	8	150 MHZ	R4400	512 MB
ORG	Origin 2000	SG	8	250 MHZ	R10000	2 GB
ULT	Ultra 2/2170	SM	2	150 MHZ	SPARC	256 MB

The test problems are derived from a family of standard test problems described in [7] and referred to there as Swirling Flow III (SWF-III). The test family is:

$$y_1'(t) = y_2(t), \quad y_2'(t) = \frac{1}{\epsilon}\left(y_1(t)y_4(t) - y_2(t)y_3(t)\right), \quad y_3'(t) = y_4(t),$$

$$y_4'(t) = y_5(t), \quad y_5'(t) = y_6(t), \quad y_6'(t) = \frac{1}{\epsilon}\left(-y_3(t)y_6(t) - y_1(t)y_2(t)\right),$$

$$y_3(a) = y_4(a) = y_3(b) = y_4(b) = 0, \quad y_1(a) = -1, \quad y_1(b) = 1.$$

No closed form solution exists in general. A number of specific test problems, obtained by specifying values for the parameters ϵ, a, and b, were identified in [57]. From these we have selected problems A, D, and E - see Table 2.

Table 2: Three SWF-III problems. Each problem is specified by ϵ and the left and right endpoints of the problem interval, $[a, b]$. The problems are listed in increasing order of difficulty.

Problem	SWF-III parameters	
	ϵ	$[a, b]$
A	.002	$[0, 1]$
D	.0001	$[-1, 1]$
E	.00275	$[0, 10]$

The initial guesses for the solution components are chosen to be straight lines through the boundary conditions, for components $y_1(t)$, $y_3(t)$, and $y_4(t)$. We choose $y_5(t) = y_6(t) = 0$ as the initial guesses for the last two components, and choose the initial guess for $y_2(t)$ to be the slope of the line used as the initial guess for $y_1(t)$ (since $y_2(t)$ represents the derivative of $y_1(t)$).

These architectures and test problems provide the basis for three experiments which we will present and analyze in the next two subsections. The specifications for these experiments are

reported in Table 3. In all experiments, the MIRKDC and PMIRKDC codes are run with the fourth order option selected so that fourth order MIRK and CMIRK methods are employed — see [35]. For one of these experiments we were interested in studying code behavior on a difficult problem for which the codes, with the usual initial solution approximation and uniform initial mesh, would not be able to obtain a solution. In this case we used a parameter continuation strategy to provide a sufficiently good initial approximation and mesh for the problem. Table 3 below specifies the continuation problem sequence in terms of the parameter ϵ.

Table 3: Three MIRKDC vs. PMIRKDC experiments. Each experiment is specified by a host architecture (Table 1), a SWF-III problem (Table 2), and a solution strategy. A solution strategy is given by a defect tolerance τ_{defect}, a number of initial mesh subintervals $Nsub_0$, and possibly a parameter sequence.

Exp. #	arch.	prb.	solution strategy		
			τ_{defect}	$Nsub_0$	continuation strategy
I	CHA	A	10^{-11}	10	none
II	ORG	E	10^{-7}	10	$\epsilon = \{1, .1, .01, .005, .00275\}$
III	ULT	D	10^{-7}	10	none

7.2 Numerical Results for Parallel Execution

Since the setup of the interpolant is actually included as a preliminary step in the estimation of the defect, the costs for the interpolant setup are included in those for defect estimation and thus only costs for defect estimation are reported.

For some experiments, the results will show that the number of backsolves is greater than the number of right hand side evaluations. This difference arises from some of the heuristics employed within the modified Newton iterations. When the MIRKDC code attempts a fixed Jacobian step, the residual function is evaluated using the current iterate and a Newton correction is computed from a backsolve of the linear system with the new residual value as the right hand side and an "old" Jacobian as the coefficient matrix. Under certain conditions, the resulting new iterate may be rejected. In this event the current residual value is saved and the Jacobian is updated and factored. Then another backsolve using the new Jacobian as the coefficient matrix and the saved residual as the right hand side is performed. In other words, when a fixed Jacobian step fails, the code will solve two linear systems with the same right hand side but with different coefficient matrices, resulting in two backsolves but only one residual (right hand side) evaluation. A similar situation arises within the subroutine which computes a suitable damping factor for the damped Newton iteration.

We will now present results and discussion for the three experiments identified in Table 3. (Note that the associated figures, referred to in the following discussion, appear after the reference section.)

7.2.1 Experiment I

In Experiment I problem A is solved with $Nsub_0 = 10$. The small number of initial mesh subintervals is a realistic choice and forces the codes to work initially with smaller ABD systems, yielding a typical solution computation. Figure 2 shows results for overall execution time for MIRKDC and PMIRKDC on various numbers of processors. The speedup in overall execution time can be seen to behave in a nearly optimal (linear) fashion despite the presence of sequential code components. On 1 processor the overall PMIRKDC execution time is about one third greater than that of MIRKDC and the parallelism pays off when 2 or more processors are used. Figure 3 provides information about the sizes of the meshes employed by the codes while attempting to compute a solution whose defect meets the required tolerance. It also shows the total number of calls to each of the parallelized solution components; the number of calls to the residual evaluation and backsolve components is much greater than the number of calls to the other components. This is because of the relatively large number of fixed Newton matrix steps taken by the code (which do not require matrix setup or factorization computations). Figure 3 also shows the percentage of calls to the factorization and backsolve routines for each mesh the code uses. It is interesting to note that most of the factorization and backsolve calls take place on the coarse meshes, indicating that the codes use substantially more full Newton and fixed Newton matrix iterations on the coarse meshes, when less information about the solution is available and the meshes are not yet well adapted to the solution behavior. Figure 4 shows nearly optimal linear speedups for all parallelized components. This figure also gives the execution times for these components in MIRKDC and in PMIRKDC for various numbers of processors. With 1 processor, we see that the matrix construction costs are by far the dominant component in MIRKDC, while in PMIRKDC, the matrix factorization costs dominate. We can also see that, with 1 processor, the factorization costs in MIRKDC are about 4 to 5 times less than those in PMIRKDC, while the backsolve costs are about half. (We recall that MIRKDC uses the sequential ABD solver COLROW and that PMIRKDC uses the parallel ABD solver RSCALE and note that these experimentally observed costs are consistent with the local operation counts for the two ABD solvers presented earlier in this paper.) The component costs in PMIRKDC drop almost linearly with the number of processors and by the time 8 processors are employed the sequential components represent the most significant costs. The slight departures from linear speedup for larger numbers of processors are attributable to the computations associated with the coarser meshes. In such cases the total amount of work performed by each processor is small and thus the overhead associated with invoking the parallel processors is relatively more significant. Similar slight departures from linear speedup are also associated with program segments characterized by light work over many calls. This comment especially applies to the residual evaluation speedups shown in Figure 4. We have observed that residual evaluation speedup can be made arbitrarily close to optimal by artificially inflating the work-per-call (these results are not shown).

7.2.2 Experiment II

We next consider a substantially more difficult problem, E; a parameter continuation strategy is invoked to obtain solutions for the problems in this sequence. Since this is a substantially more time-consuming problem, we employ the Origin 2000 which is up to five times as fast as the Challenge L. The performance statistics reported in this experiment apply to the final, most

computationally-intensive iteration of the continuation loop only.

For the final problem in the continuation sequence for Experiment II the speedup assessment for PMIRKDC and the overall execution times for MIRKDC and PMIRKDC are given in Figure 5. This is quite a difficult problem and thus substantial execution time is required even for the final problem. This problem is also a more relevant test problem since we are of course primarily interested in the application of parallelism in problems with large execution times. We see that PMIRKDC exhibits nearly linear speedups, that on 1 processor the PMIRKDC cost is about 2.5 times that of MIRKDC, and that parallelism pays off after 3 processors. Figure 6 gives the mesh sequence, the total number of calls for the significant computational components, and the percentage of calls to the factorization and backsolve routines for each mesh. We also see that the number of residual evaluation and backsolve calls dominates the number of calls to the other routines, due to the relatively large number of fixed Newton matrix iterations. We also observe that except for one mesh, the number of Newton matrix factorizations and backsolves is quite uniform over the meshes considered. This is more typical of the latter part of a continuation sequence where the meshes are better adapted to the solution behavior. Figure 7 shows nearly optimal linear speedups for all parallelized components. In addition it shows the relative costs of these components within the MIRKDC code and within the PMIRKDC code for various numbers of processors. For 1 processor the matrix setup costs dominate in MIRKDC while in PMIRKDC the matrix backsolve and factorization costs are the highest. The factorization cost in PMIRKDC is about 5 times that of MIRKDC. As the number of processors increases the cost of the parallelized components drops approximately linearly; for 8 processors we see that the factorization and backsolve costs have dropped to almost the same level as the sequential costs.

Cumulative results for this experiment over all meshes considered throughout the entire continuation sequence are provided in Figure 8. This figure gives the total number of calls of each of the significant algorithm components and an indication of the relative number of calls to the matrix factorization and backsolve routines on the various meshes considered. From this figure, we see that the number of residual evaluations and backsolves always substantially outnumbers the calls to the other routines. As well, we see that the percentage of calls to the matrix factorization and backsolve routines is generally higher for the finer meshes, a behavior that is consistent with the fact that such problems are very difficult and that even on finer meshes substantial work is still required to obtain accurate solutions.

7.3 Numerical Results for Sequential Execution

In the experiments considered so far the solution profiles exhibited by MIRKDC and PMIRKDC were identical. That is, both codes solved the problems using the same sequence of meshes having the same number of subintervals and employing the same number of Newton iterations, and producing answers in agreement to approximately round-off level. In this section we consider a problem for which this is not the case.

Experiment III, run sequentially on the Sun Ultra 2, illustrates a potential secondary advantage of RSCALE. Even though there is no parallelism, PMIRKDC greatly outperforms MIRKDC (Figure 9). The reduced execution times in PMIRKDC are explained by comparing the subroutine call and call-per-mesh profiles, in Figures 10-11. From these figures it can be seen that PMIRKDC can use substantially coarser meshes with substantially fewer calls to all computational components,

leading to great savings in execution time. Compared to MIRKDC, PMIRKDC solves this problem with a saving of about 85%. We believe this happens because RSCALE may be acting as a mild "preconditioner" in these problems, producing more accurate ABD system solutions which in turn lead to shorter convergence patterns (i.e., meshes that more quickly adapt to the solution behavior using fewer Newton iterations) in PMIRKDC.

8 Conclusions and Future Work

Difficult BVODEs arise in many applications and can represent a substantial computational cost. It is therefore important to develop numerical methods which can take advantage of the availability of parallel processor architectures in an efficient manner. It has been known for some time that the most significant bottleneck to the effective parallelization of numerical methods for BVODEs is the parallel factorization and solution of the ABD linear systems which arise. While there has been considerable research in the investigation of parallel algorithms for ABD systems and a few high quality software packages have been developed, relatively very little has been done in the development of parallel BVODE codes based on this software.

In this paper we have described in detail the software development effort associated with modifying a sequential BVODE code to allow it to run efficiently in a parallel processor environment; this modification features the parallel treatment of ABD systems using the RSCALE package. The six most computationally intensive components of the BVODE code were targeted for parallelization. Extensive numerical testing demonstrates that almost optimal, linear speedups for all these components are attained, leading to substantial savings in overall execution time. Further reductions in the relative costs of these components would likely be observed on an architecture with more processors. However, even with only eight processors, it is significant to note that, generally, the remaining sequential algorithmic components of the BVODE code become relatively the most expensive. This implies that one area for further research should be in the development of parallel algorithms for some of these components; e.g., parallel mesh selection.

As reported in this paper, it was necessary to provide a very careful implementation of the compiler directives and associated use of memory in order to avoid parallel memory access conflicts. The parallel compiler directives provided by Silicon Graphics within their Fortran compilers furnish a convenient tool for implementing parallelism and managing parallel memory access on a shared memory machine. In fact, the parallel directives within Fortran provided by a number of different vendors are relatively similar and the task of modifying a code to run under a different compiler is sometimes not particularly difficult. However, it would be convenient to employ "portable" compiler directives. For shared memory architectures, the OpenMP Application Program Interface [62] provides this facility and many vendors already provide or are in the process of providing OpenMP implementations. Another area for future work on this project is therefore the replacement of the vendor specific compiler directives with portable OpenMP directives within PMIRKDC.

We have observed significant performance improvements, for certain problems, even on sequential machines, due to the replacement of COLROW by RSCALE. We are currently conducting an investigation of this phenomenon, including a study of the effect of the use of RSCALE on the conditioning of the ABD systems that arise throughout a computation.

Another direction we are currently considering is the possible advantage in using the RSCALE software, even in sequential processor environments, in the direct treatment of BVODEs with nonseparated boundary conditions. Since COLROW cannot handle BABD systems, one must first uncouple the boundary conditions. For fully coupled conditions, this leads to a doubling of n, the number of equations, and thus in some of the most computationally intensive components of a BVODE code (where the costs depend on n^3), costs are increased by a factor of eight. Hence, even though the factorization costs of RSCALE are about four times those of COLROW, the overall savings may favour the use of RSCALE.

References

[1] P.Amodio and M.Paprzycki, Parallel solution of almost block diagonal systems on a hypercube, Lin. Alg. Applic., 241-243, (1996), 85–103.

[2] P. Amodio and M. Paprzycki, A cyclic reduction approach to the numerical solution of boundary value ODEs, SIAM J. Sci. Comput., 18, (1997), 56–68.

[3] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut, and M. Paprzycki, Almost Block Diagonal Linear Systems: Sequential and Parallel Solution Techniques, and Applications, Numer. Linear Algebra Appl., 7, (2000), 275–317.

[4] U.M. Ascher and S.Y.P. Chan, On parallel methods for boundary value ODEs, Computing, 46, (1991), 1–17.

[5] U.M. Ascher, J. Christiansen, and R.D. Russell, A collocation solver for mixed order systems of boundary value problems, Math. Comp., 33, (1979), 659–679.

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

[7] U.M. Ascher, R.M.M. Mattheij, and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Classics in Applied Mathematics Series, SIAM, Philadelphia, 1995.

[8] U.M. Ascher and R.D. Russell, Reformulation of boundary value problems into 'standard' form, SIAM Review, 23, (1981), 238-254.

[9] Association for Computing Machinery, New York, New York, USA, Collected Algorithms, www.acm.org/calgo.

[10] 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), 483-500.

[11] G. Bader and P. Kunkel, Continuation and collocation for parameter-dependent boundary value problems, SIAM J. Sci. Stat. Comput., 10, (1989), 72–88.

[12] Z. Bashir-Ali, J.R. Cash, H.H.M. Silva, Lobatto deferred correction for stiff two-point boundary value problems, Comput. Math. Appl., 36, (1998), 59–69.

[13] K.R. Bennett, Parallel collocation methods for boundary value problems, Ph.D. thesis, Department of Mathematics, University of Kentucky, Tech. Rep. CCS-91-1, University of Kentucky Center for Computational Sciences, 1991.

[14] K.R. Bennett and G. Fairweather, A parallel boundary value ODE code for shared-memory machines, Int. J. High Speed Comput., 4, (1992), 71–86.

[15] C. deBoor, *A practical guide to splines*, Applied Mathematical Sciences, 27, Springer-Verlag, New York-Berlin, 1978.

[16] C. deBoor and R. Weiss, SOLVEBLOK: a package for solving almost block diagonal linear systems, ACM Trans. Math. Softw., 6, (1980), 80–87.

[17] C. de Boor and R. Weiss, Algorithm 546: SOLVEBLOK, ACM Trans. Math. Softw., 6, (1980), 88–91.

[18] R.W. Brankin and I. Gladwell, Codes for almost block diagonal systems, Comput. Math. Appl., 19 (1990), 1–6.

[19] K. Burrage, F.H. Chipman, and P.H. Muir, Order results for mono-implicit Runge-Kutta methods, SIAM J. Numer. Anal., 31, (1994), 876-891.

[20] J.C. Butcher, *The numerical analysis of ordinary differential equations, Runge-Kutta and general linear methods*, John Wiley & Sons, Ltd., Chichester, 1987.

[21] J.R. Cash and A. Singhal, Mono-implicit Runge-Kutta formulae for the numerical integration of stiff differential systems, IMA J. Numer. Anal., 2 (1982), 211–227.

[22] J.R. Cash and M.H. Wright, Implementation issues in solving nonlinear equations for two-point boundary value problems, Computing, 45 (1990), 17–37.

[23] J.R. Cash and M.H. Wright, A deferred correction method for nonlinear two-point boundary value problems: implementation and numerical evaluation, SIAM J. Sci. Stat. Comput., 12, (1991), 971-989.

[24] J.R. Cash, G. Moore, and R.W. Wright, An automatic continuation strategy for the solution of singularly perturbed linear two-point boundary value problems, J. Comput. Phys., 122, (1995), 266–279.

[25] K.L. Chow and W.H. Enright, Distributed Parallel Shooting for BVODEs, Proceedings of High Performance Computing, A. Tentner, ed., The Society for Computer Simulation, Boston, (1998), 203-210.

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

[27] J.C. Diaz, G. Fairweather, and 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. Softw., 9, (1983), 376–380.

[28] AUTO: ftp.cs.concordia.ca/pub/doedel/auto/.

[29] E. Doedel, H.B. Keller, and J.-P. Kernevez, Numerical analysis and control of bifurcation problems. I. Bifurcation in finite dimensions, Internat. J. Bifur. Chaos Appl. Sci. Engrg., 1 (1991), 493–520.

[30] J.J. Dongarra, J. Du Croz, S. Hammarling, and I.S. Duff, A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Softw., 16, (1990), 1–17.

[31] J.J. Dongarra, J. Du Croz, S. Hammarling, and I.S. Duff, Algorithm 679; a set of level 3 basic linear algebra subprograms: model implementation and test programs, ACM Trans. Math. Softw., 16, (1990), 18–28.

[32] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, An extended set of FORTRAN basic linear algebra subprograms, ACM Trans. Math. Softw., 14, (1988), 1–17.

[33] J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, Algorithm 656: an extended set of basic linear algebra subprograms: model implementation and test programs, ACM Trans. Math. Softw., 14, (1988), 18–32.

[34] W.H. Enright and P.H. Muir, A Runge-Kutta Type Boundary Value ODE Solver with Defect Control, Dept. Comp. Sci. Tech. Rep. 93-267, University of Toronto, 1993.

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

[36] I. Gladwell and R.I. Hay, Vector- and parallelization of ODE BVP codes, Parallel Comput., 12, (1989), 343–350.

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

[38] M. Goldmann, Vectorization of the multiple shooting method for the nonlinear boundary value problem in ordinary differential equations, Parallel Comput., 7, (1988), 97–110.

[39] S. Gupta, An adaptive boundary value Runge-Kutta solver for first order boundary value problems, SIAM J. Numer. Anal., 22, (1985), 114-126.

[40] R.I. Hay and I. Gladwell, Solving almost block diagonal linear equations on the CDC Cyber 205, Numerical Analysis Report 98, Department of Mathematics, University of Manchester, 1985.

[41] M. Hegland and M.R. Osborne, Wrap-around partitioning for block bidiagonal linear systems, IMA J. Numer. Anal., 18 (1998), 373–383.

[42] IMSL: Visual Numerics, www.vni.com/products/imsl.

[43] K.R. Jackson and R.N. Pancer, The Parallel Solution of ABD Systems Arising in Numerical Methods for BVPs for ODEs, Technical report 255/91, Dept. Computer Science, Univ. of Toronto, 1992.

[44] M. Lentini and V. Pereyra, An adaptive finite difference solver for nonlinear two-point boundary problems with mild boundary layers, SIAM J. Numer. Anal., 14, (1977), 94–111.

[45] M. Lentini and V. Pereyra, PASVA4: an ODE boundary solver for problems with discontinuous interfaces and algebraic parameters, Mat. Apl. Comput., 2, (1983), 103–118.

[46] L. Liu and R.D. Russell, Linear system solvers for boundary value ODEs, J. Comput. Appl. Math., 45, (1993), 103–117.

[47] C.P. Katti and S. Goel, A parallel mesh chopping algorithm for a class of two-point boundary value problems, Comput. Math. Appl., 35, (1998), 121–128.

[48] H.B. Keller and P. Nelson, A comparison of hypercube implementations of parallel shooting, Mathematics for large scale computing, 49–79, Lecture Notes in Pure and Appl. Math., 120, Dekker, New York, 1989.

[49] H.B. Keller and P. Nelson, Hypercube implementations of parallel shooting, Appl. Math. Comput., 31, (1989), 574–603.

[50] J. Kierzenka and L.F. Shampine, A BVP Solver based on Residual Control and the MATLAB PSE, SMU Math Report 99-001, Department of Mathematics, Southern Methodist University, Dallas, TX, USA, 1999.

[51] G.L. Kraut and I. Gladwell, Cost of stable algorithms for bordered almost block diagonal systems, ICIAM/GAMM 95 (Hamburg, 1995), Z. Angew. Math. Mech., 76, (1996), suppl. 1, 151–154.

[52] MATLAB: The MathWorks Inc., Natick, MA, USA, www.mathworks.com.

[53] R.M.M. Mattheij and S.J. Wright, Parallel stable compactification for ODEs with parameters and multipoint conditions, Appl. Numer. Math., 13, (1993), 305–333.

[54] F. Majaess, P. Keast, and G. Fairweather, The solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions, ACM Trans. Math. Softw., 18, (1992), 193–204.

[55] F. Majaess, P. Keast, G. Fairweather and K.R. Bennett, Algorithm 704: ABDPACK and ABBPACK-FORTRAN programs for the solution of almost block diagonal linear systems arising in spline collocation at Gaussian points with monomial basis functions, ACM Trans. Math. Softw., 18, (1992), 205–210.

[56] A. Miele and T. Wang, Parallel computation of two-point boundary value problems via particular solutions, J.Optim. Theory Appl., 79, (1993), 5–29

[57] P.H. Muir, R.N. Pancer, and K.R. Jackson, Runge-Kutta Software for the Parallel Solution of Boundary Value ODEs, Technical Report 2000-08, Dept. Math. and Comp. Sci., Saint Mary's University, Halifax, N.S., Canada, 2000, or Technical Report, Dept. Comp. Sci., University of Toronto, Toronto, Ont., Canada, 2000.

[58] P.H. Muir and K. Remington, A parallel implementation of a Runge-Kutta code for systems of nonlinear boundary value ODEs, Cong. Numer., 99, (1994), 291–305.

[59] P. Muir and B. Owren, Order barriers and characterizations for continuous mono-implicit Runge-Kutta schemes, Math. Comp., 61, (1993), 675-699.

[60] NAG: Numerical Algorithms Group, OXFORD, UK, www.nag.co.uk.

[61] Netlib: www.netlib.org.

[62] The OpenMP Application Program Interface, www.openmp.org.

[63] M.R. Osborne, Cyclic reduction, dichotomy, and the estimation of differential equations, J. Comput. Appl. Math., 86, (1997), 271–286.

[64] R.N. Pancer, The Parallel Solution of Almost Block Diagonal Linear Systems, Ph.d. Thesis, Dept. of Comp. Sci., University of Toronto, 2000.

[65] R.N. Pancer and K.R. Jackson, The Parallel Solution of Almost Block Diagonal Systems Arising in Numerical Methods for BVPs for ODEs, In the Proceedings of the Fifteenth IMACS World Congress, Berlin, Germany, Volume 2, 57–62, 1997.

[66] M. Paprzycki and I. Gladwell, Solving almost block diagonal systems on parallel computers, Parallel Comput., 17, (1991), 133–153.

[67] M. Paprzycki and I. Gladwell, Using level 3 BLAS to solve almost block diagonal systems, Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing, (Houston, TX, 1991), SIAM, Philadelphia, PA, (1992), 52–62.

[68] M. Paprzycki and I. Gladwell, A parallel chopping algorithm for ODE boundary value problems, Parallel Comput., 19, (1993), 651–666.

[69] H. Pasic and Y. Zhang, Parallel solutions of BVPs in ODEs based on local matching, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia, PA, 1997.

[70] J.K. Reid and A. Jennings, On solving almost block diagonal (staircase) linear systems, ACM Trans. Math. Softw., 10, (1984), 196–201.

[71] R.D. Russell and J. Christiansen, Adaptive mesh selection strategies for solving boundary value problems, SIAM J. Numer. Anal., 15, (1978), 59–80.

[72] R. Weiss, The application of implicit Runge-Kutta and collocation methods to boundary value problems, Math. Comp., 28, (1974), 449-464.

[73] D.E. Womble, R.C. Allen, and L.S. Baca, Invariant imbedding and the method of lines for parallel computers, Parallel Comput., 11, (1989), 263–273.

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

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

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

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

[78] S.J. Wright, Stable parallel elimination for boundary value ODEs, Numer. Math., 67, (1994),521–535.

[79] 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), 425–449.

Figure 2: Overall speedup and execution time of MIRKDC and PMIRKDC in Experiment I. Parallelism begins to pay off at 2 processors.

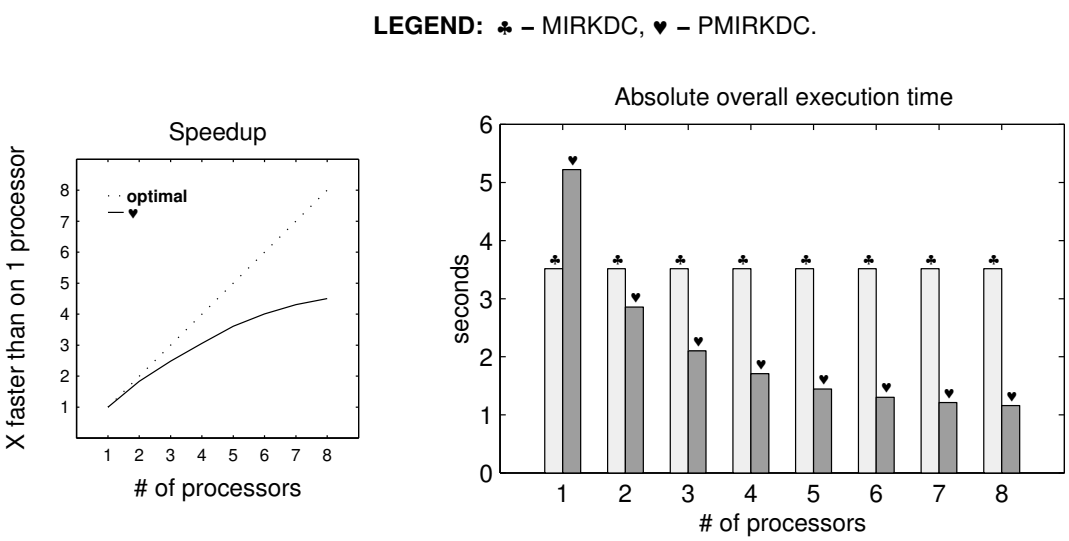

Figure 3: Subroutine call and call-per-mesh profiles of MIRKDC and PMIRKDC in Experiment I.

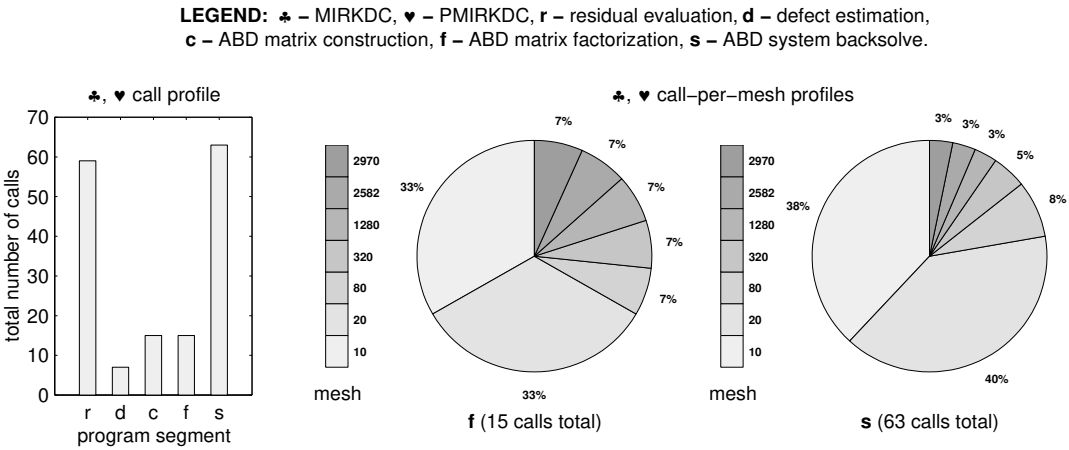

Figure 4: Speedup and execution time of selected program segments of MIRKDC and PMIRKDC in Experiment I.

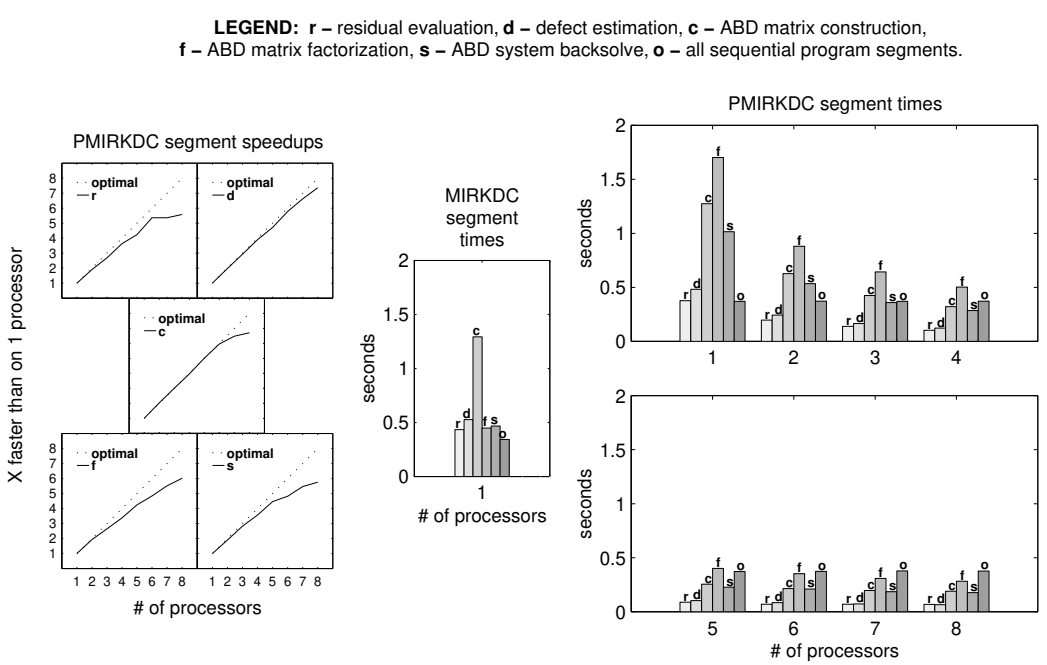

Figure 5: Overall speedup and execution time of MIRKDC and PMIRKDC in Experiment II. Results are shown for the final continuation step only. Parallelism begins to pay off at 3 processors.

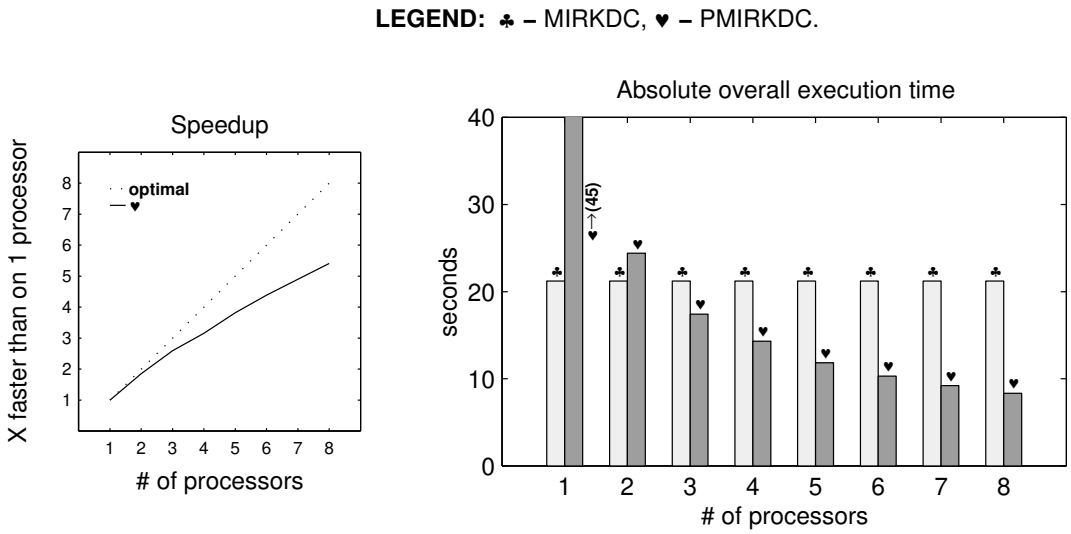

Figure 6: Subroutine call and call-per-mesh profiles of MIRKDC and PMIRKDC in Experiment II. Results are shown for the final continuation step only, which is the most computationally intensive.

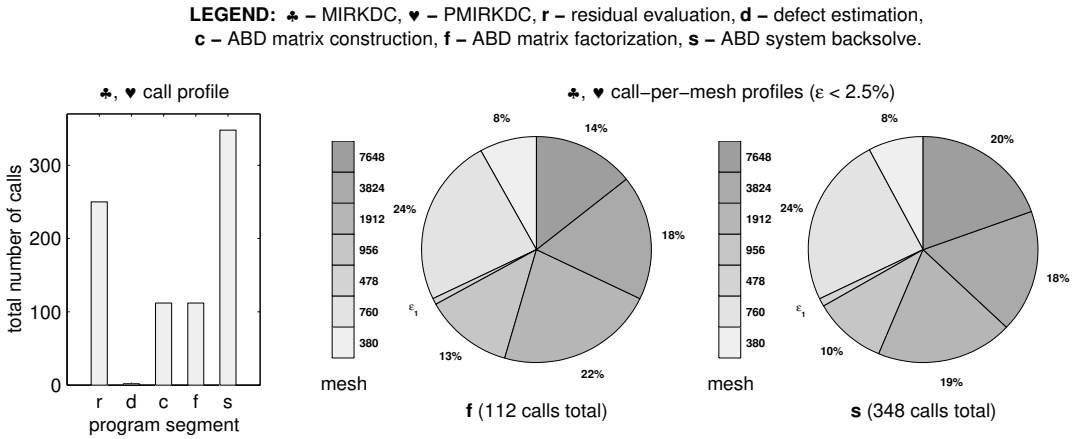

Figure 7: Speedup and execution time of selected program segments of MIRKDC and PMIRKDC in Experiment II. Results are shown for the final continuation step only.

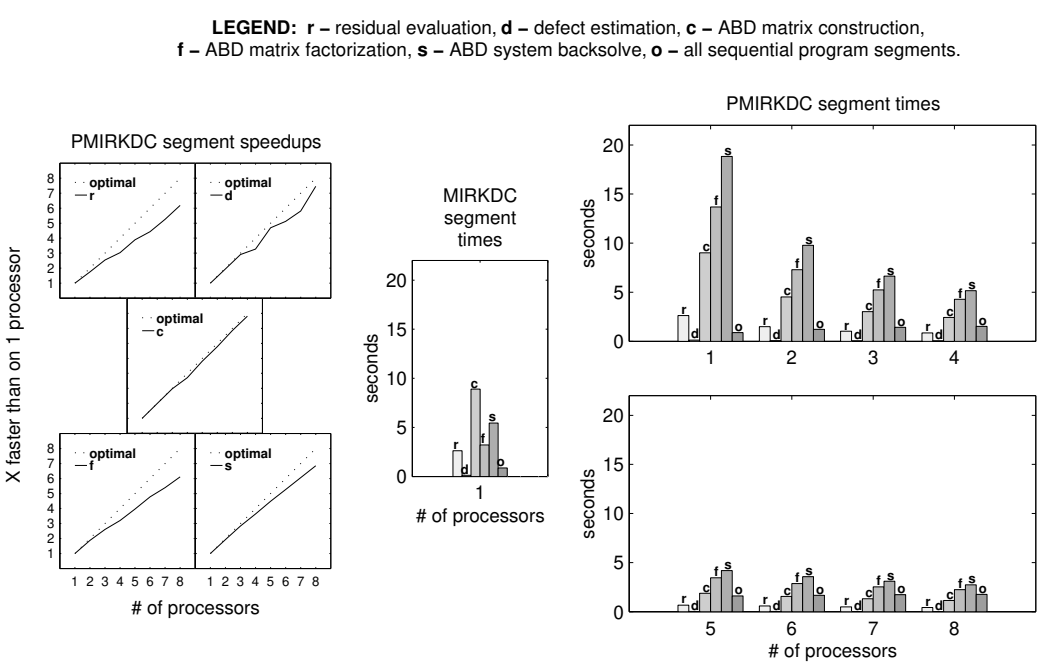

Figure 8: Subroutine call and call-per-mesh profiles of MIRKDC and PMIRKDC in Experiment II. Results are accumulated over all five continuation steps.

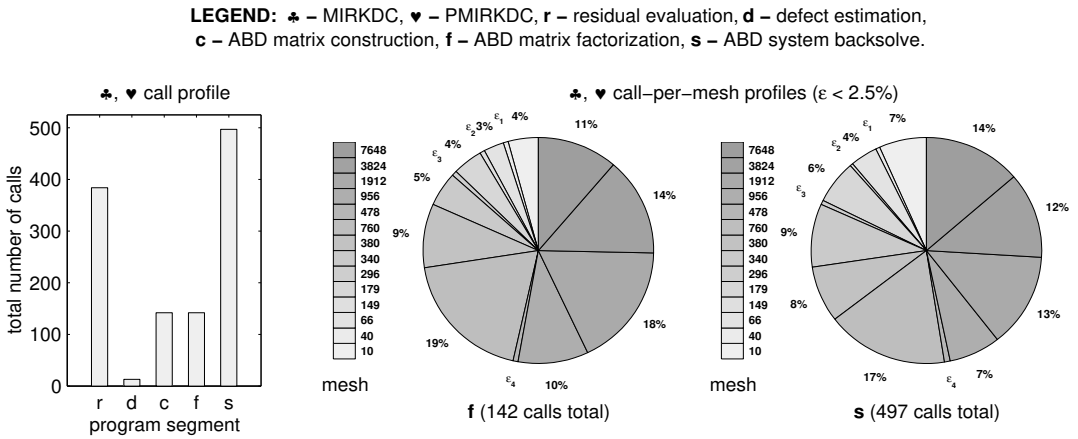

Figure 9: Overall and selected program segment execution times of MIRKDC and PMIRKDC in Experiment III. This experiment is run sequentially on the Ultra 2. The vast difference in execution times is explained by the subroutine call and call-per-mesh profiles shown in Figures 10 and 11.

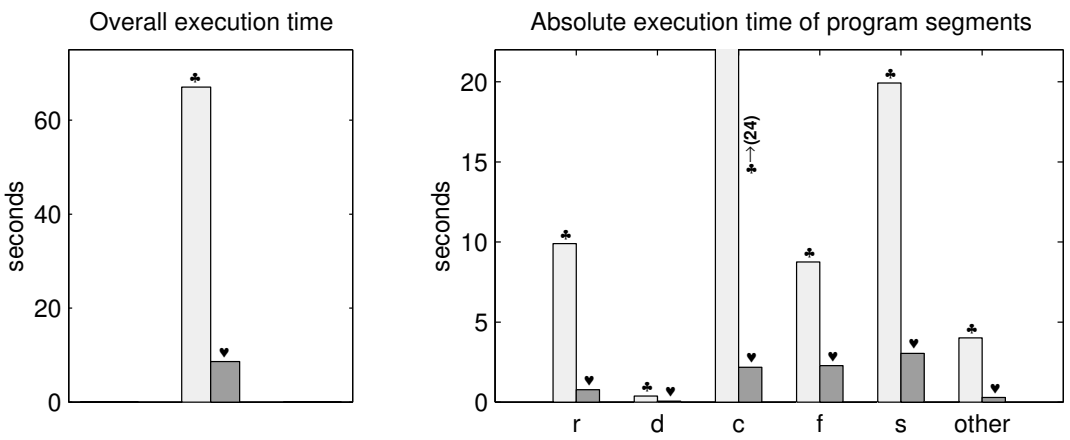

Figure 10: Profiles of MIRKDC in Experiment III. These differ significantly from those of PMIRKDC (Figure 11).

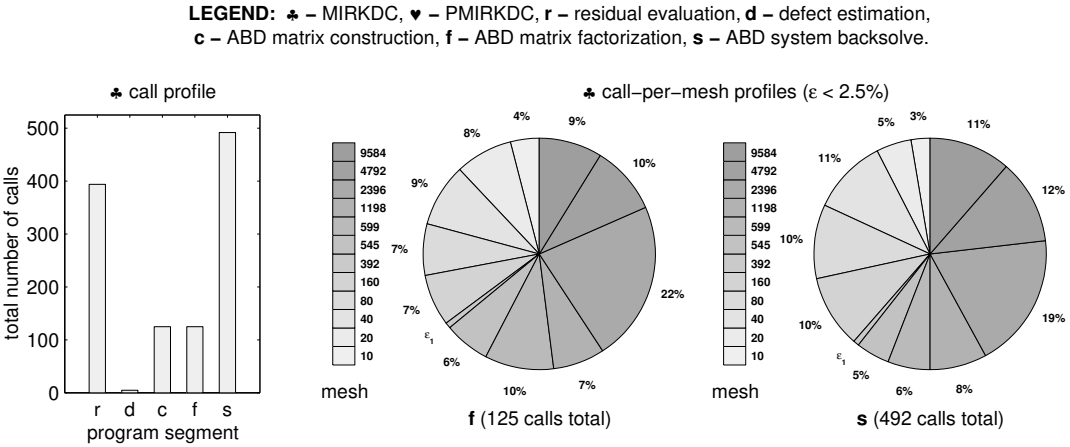
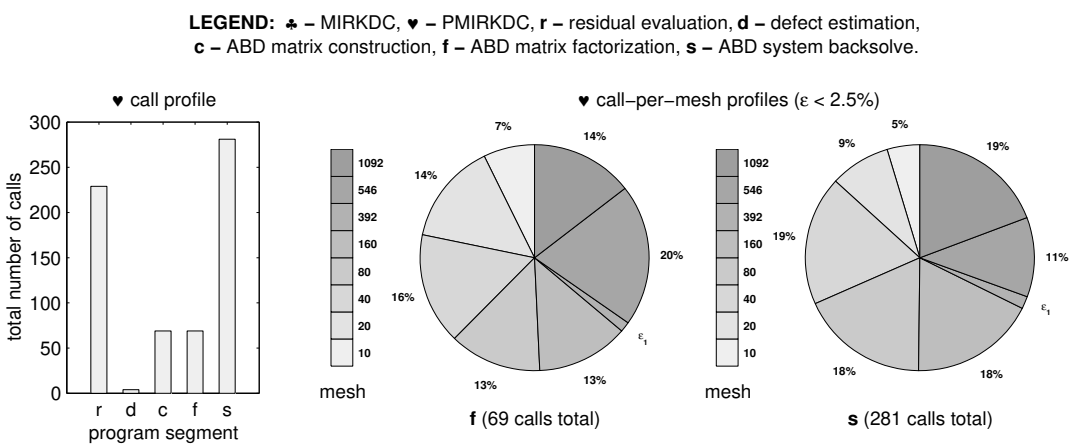
Figure 11: Profiles of PMIRKDC in Experiment III. PMIRKDC computes an acceptable solution using fewer subroutine calls and fewer mesh subintervals than MIRKDC, resulting in substantially reduced execution time (Figure 9).