Distributed parallel shooting for BVODEs

K.L. Chow and W.H. Enright
Department of Computer Science, University of Toronto
E-mail : chow@cs.toronto.edu and enright@cs.toronto.edu

Abstract

Many important scientific problems can be formulated as systems of ordinary differential equations with two-point boundary value constraints (BVODE). Multiple shooting is one of the most widely used numerical techniques for solving BVODE problems. In this work, we present a new distributed parallel numerical algorithm for BVODEs which is based on multiple shooting. We investigate the numerical stability of this new distributed algorithm and identify difficulties that can arise. We propose a new parallel iterative refinement scheme to cope with some specific numerical difficulties identified in our investigation. Computational experience is presented to demonstrate the potential effectiveness of our approach.

Keywords: Distributed parallel algorithm, boundary value problems, multiple shooting

1 Introduction

This paper focuses on effective parallel numerical algorithms for solving first-order systems of ordinary differential equations with two-point boundary value constraints (BVODE) of the form

$$y' = f(x, y(x)) \quad \text{subject to} \quad g(y(a), y(b)) = 0 \quad \text{for } x \in [a, b] \tag{1}$$

where $y(x) \in R^n$, $f : R \times R^n \to R^n$, and $g : R^n \times R^n \to R^n$. BVODEs have many important scientific applications and arise in such diverse areas as the modeling of semiconductor devices and the simulation of the re-entry of space vehicles from orbit.

Multiple shooting [Kel92] is one of the most widely used numerical techniques for solving BVODE problems. It is based on the idea of reformulating the problem as a sequence of initial value problems with continuity constraints. This is done by partitioning the interval $[a, b]$ into a number of subintervals, integrating the corresponding initial value problems over each subinterval, and then simultaneously adjusting all the initial data in order to satisfy all boundary conditions and appropriate continuity conditions at the mesh points.

There is some obvious parallelism in multiple shooting that can be exploited. For instance, the computations on each subinterval can be done independently and hence can be computed by different processors. Thus, if the number of subintervals is large, parallel computation should have a signification influence on the processing time of some components of an algorithm based on multiple shooting. This has been the focus on a number of recent investigations [Len88, WP90, PG90, BF90, AC91, JP92, Wri92] which have been concerned with finding new parallel numerical algorithms for BVODEs. Surveys discussing existing parallel algorithms for BVODE problems can be found in [JP92, ACR+].

A common difficulty discussed in the above investigations is that a fast and stable parallel solution of the large linear systems (that arise when using multiple shooting) is nontrivial. Assume that m (a power of 2) is the number of mesh points used to partition the interval of interest $[a, b]$. The size of the linear systems arising from the application of multiple shooting to the underlying BVODE is mn. The current state of development of parallel solution of such linear systems makes use of an appropriate variant of an odd-even block cyclic reduction technique, see [AC91, JP92, Wri92]. An odd-even cyclic reduction technique or a variant can solve the linear system in $O(\log m)$ parallel steps using $O(m)$ processors and each parallel step requires $O(n^3)$ floating point operations (flops). Thus, this technique requires a total of $O(n^3 \log m)$ arithmetic operations which can be shown to be an optimal bound for the parallel solution of such linear systems.

We have developed our distributed algorithm by taking advantages of the inherent structure of the large linear systems that arise when using multiple shooting. In particular, we employ efficient prefix and fan-in associative operations to solve these systems. Our distributed algorithm requires $O(n^3 \log m)$ flops and is therefore comparable in efficiency to other proposed solvers. We investigate the numerical stability of this new distributed algorithm and identify difficulties that can arise. We propose a new parallel iterative refinement scheme to cope with some specific numerical difficulties identified in our investigation.

We have tested our new approach by modifying an existing widely used multiple shooting algorithm (BVPMS). We have implemented several different versions of this modified multiple shooting algorithm on both serial and distributed memory machines using PVM interfaces. We have chosen test problems from the literature which are expensive to solve by standard boundary value solvers. In particular, the chosen test problems require extensive CPU time and/or considerable storage on a serial machine. Computational experience on these test problems is presented to demonstrate the potential effectiveness of our approach.

The numerical results show that our parallel implementations can achieve considerable speedup on a distributed memory machine. Time savings on the order of 80-95 % over BVPMS for moderate size BVODE problems is observed. We will also discuss how most of the techniques developed in this investigation can be adopted by other widely used numerical approaches such as those based on finite difference or collocation.

This paper is organized as follows. In section 2, we briefly discuss the multiple shooting technique and describe a generic multiple shooting algorithm for solving BVODEs. We discuss how numerical difficulties can arise when a BVODE problem is solved by this multiple shooting algorithm. In section 3, we present a new distributed parallel implementation of this multiple shooting algorithm. Then, we present our parallel iterative refinement scheme to cope with some specific numerical difficulties. In section 4, both serial and parallel computation results are given. Finally, in section 5, we discuss our results and discuss plans for future investigations.

2 Multiple shooting

To solve the first-order BVODE (1) using multiple shooting, we interpret the problem as a sequence of initial value problems with unknown initial conditions and apply a Newton-like iteration to solve for these unknowns to within an accuracy requirement. This requires supplying initial guesses for the unknown initial conditions. Thus, multiple shooting for the approximate solution of BVODE problems can be characterized in terms of variants of an iterative algorithm. For linear problems, a single iteration will generally yield the required set of initial values. Nonlinear BVODE problems can be solved iteratively with each iteration requiring a set of initial value integrations from a to b. Additional mesh points can be inserted in $[a, b]$ to give a larger but often better-conditioned system of equations.

More specifically, a multiple shooting algorithm for solving the BVODE problem (1) will introduce a mesh, $\Pi : a = x_0 < x_1 < \cdots < x_{m-1} = b$, of the problem interval and then determine a corresponding discrete approximation, $Y = (y_0, y_1, \cdots, y_{m-1})^T$ where $y_i \in R^n$ for $i = 0, \cdots, m-1$. Each component of Y is an approximation to the true solution of (1) evaluated at the corresponding mesh point, i.e., $y_i \approx y(x_i)$. A local initial value problem,

$$v_i'(x; y_i) = f(x, v_i(x; y_i)) \quad \text{subject to} \quad v_i(x_i; y_i) = y_i \quad \text{and} \quad x \in [x_i, x_{i+1}], \tag{2}$$

is defined for each mesh subinterval for $i = 0, \cdots, m-2$. Multiple shooting consists of performing the following steps on each iteration. We first approximate $v_i(x_{i+1}; y_i)$ using a reliable initial value solver applied to (2) for $i = 0, \cdots, m-2$. Continuity of the solution at the shooting points implies that $v_i(x_{i+1}; y_i) = y_{i+1}$. The boundary condition requires $g(y_0, y_{m-1}) = 0$. Writing the boundary conditions first and then the $m-1$ continuity conditions at the shooting points x_i, for $i = 0, \cdots, m-2$, we get the system of nm nonlinear equations

$$\Phi(Y) = \begin{pmatrix} g(y_0, y_{m-1}) \\ v_0(x_1; y_0) - y_1 \\ v_1(x_2; y_1) - y_2 \\ \vdots \\ v_{m-2}(x_{m-1}; y_{m-2}) - y_{m-1} \end{pmatrix} = 0$$

where Φ is the residual function. This nonlinear system is generally solved using a Newton-like method where the Jacobian $J(Y)$ of $\Phi(Y)$ has the block matrix form

$$\begin{pmatrix} B_0(y_0, y_{m-1}) & & & & & B_{m-1}(y_0, y_{m-1}) \\ Y_0(x_1) & -I & & & & \\ & Y_1(x_2) & -I & & & \\ & & \ddots & \ddots & & \vdots \\ & & & Y_{m-3}(x_{m-2}) & -I & \\ & & & & Y_{m-2}(x_{m-1}) & -I \end{pmatrix}.$$

For simplicity, we introduce the notation $\Phi_0(y_0, y_{m-1}) = g(y_0, y_{m-1})$ and $\Phi_i(y_{i-1}, y_i) = v_{i-1}(x_i; y_{i-1}) - y_i$, and $L_i = Y_{i-1}(x_i)$ for $i = 1, \cdots, m-1$. The blocks $B_0(y_0, y_{m-1})$ and $B_{m-1}(y_0, y_{m-1})$ are determined by $\frac{\partial \Phi_0(y_0, y_{m-1})}{\partial y_0}$ and $\frac{\partial \Phi_0(y_0, y_{m-1})}{\partial y_{m-1}}$ respectively. The block $Y_i(x_{i+1})$ can be obtained by solving a matrix initial value problem

$$Y_i'(x) = A(x, v_i)Y_i(x) \quad \text{subject to} \quad Y_i(x_i) = I \quad \text{and} \quad x \in [x_i, x_{i+1}]$$

where $A(x, v_i) = \frac{\partial f(x, v_i)}{\partial v_i}$. We summarize the multiple shooting procedure as follows.

Algorithm 2.1 *(Multiple shooting)*

Given $\Pi\ : a = x_0 < x_1 < \cdots < x_{m-1} = b$ *and* Y^0
For $l = 0, 1, \cdots,$ **do until** *convergence criteria satisfied*
  For $i = 0, 1, \cdots, m-2,$ **do** *integrate the IVPs to obtain approximations to* $v_i(x_{i+1}; y_i^l)$ *and* $Y_i(x_{i+1})$
    $v_i'(x; y_i^l) = f(x, v_i(x, y_i^l))$ *subject to* $v_i(x_i; y_i^l) = y_i^l$ *for* $x \in [x_i, x_{i+1}]$
    $Y_i'(x) = A(x, v_i)Y_i(x)$ *subject to* $Y_i(x_i) = I$ *for* $x \in [x_i, x_{i+1}]$
  Endfor
  Assemble $J(Y^l)$ *and compute* $-\Phi(Y^l)$
  Solve the linear system $J(Y^l)S^l = -\Phi(Y^l)$
  Determine $Y^{l+1} = Y^l + S^l$
Endfor

Numerical difficulties may arise when one attempts to solve BVODE problems using multiple shooting. In many cases, these numerical difficulties can be attributed to the conditioning of the linear system arising in the Newton iteration. Even for well-conditioned BVODE problems, the blocks L_i may possess both decaying and growing solution components, see [dHM83, Mat82]. If the length of a mesh interval $[x_{i-1}, x_i]$ is large, one can either incur overflow in the computation of L_i or the columns can become numerically dependent. By restricting the length of mesh intervals, one can control the size of $\parallel L_i \parallel$ and at the same time avoid numerical singular blocks. However, the number of mesh intervals and the corresponding increase in storage may be unacceptable.

In [DB82], it has been shown that there are BVODE problems which are comparatively well-conditioned even though the corresponding initial value problems are ill-conditioned. These well-conditioned BVODEs exhibit both rapidly growing and decaying solution components. The underlying possibly fast propagation of information by solution components may lead to stability difficulties when information from different mesh intervals is combined to form a global approximate solution. In this case the inability to reliable solve the linear system will result in a failure of the Newton iteration to converge.

Even if the linear system arising in the Newton iteration is well-conditioned, the convergence of the outer iteration of Algorithm 2.1 can be very sensitive to the accuracy of the initial approximation to Y. There are two common ways to cope with this difficulty. One is to use a continuation technique [Kel92]. Another alternative is to use a more robust nonlinear equation solver such as damped-Newton iteration [JDS83]. However, we will not address this difficulty in this investigation.

3 Distributed parallel shooting

In this section, we present a distributed parallel implementation of the multiple shooting algorithm described in the previous section. Our implementation is based upon efficient parallel prefix and fan-in associative operations. Recall that, for each iterate Y^l, the computation of the components of the residual function Φ and the blocks of the Jacobian matrix, J, of Φ can be done in parallel. We consider how one can further exploit parallelism in solving the linear system, $JS = -\Phi$. We observe that the special structure of the matrix J enables us to organize the computation of the correction vector S so it only uses its non-zero block components. This organization avoids unnecessary fill-in and thus reduces the memory requirement and associated operation counts.

The idea is to transform the linear system, $JS = -\Phi$, into an equivalent linear system, $\bar{J}S = -\bar{\Phi}$, such that $\bar{J}$ has a block lower bidiagonal structure. We assume that the block components B_0, B_{m-1}, and L_i of J have been computed and the augmented matrix of the original system is given by

$$[J, \Phi] = \left[\begin{array}{ccccccc|c} B_0 & & & & & & B_{m-1} & \Phi_0 \\ L_1 & -I & & & & & & \Phi_1 \\ & L_2 & -I & & & & & \Phi_2 \\ & & L_3 & -I & & & & \Phi_3 \\ & & & \ddots & \ddots & & & \vdots \\ & & & & L_{m-2} & -I & & \Phi_{m-2} \\ & & & & & L_{m-1} & -I & \Phi_{m-1} \end{array}\right].$$

Using straightforward block Gaussian elimination, we can now eliminate the last block component in the first row. The first elimination step is to add B_{m-1} times the m-th row to the first. The second elimination step is to add $B_{m-1}L_{m-1}$ times the $(m-1)$-th row to this new first row. Similarly, we can apply an elimination step to each i-th row accordingly for $i = m-3, \cdots, 2,$. At the end of this block elimination process, we obtain an equivalent augmented matrix

$$\left[\bar{J}, \bar{\Phi}\right] = \left[\begin{array}{ccccccc|c} B & & & & & & & \bar{\Phi}_0 \\ L_1 & -I & & & & & & \Phi_1 \\ & L_2 & -I & & & & & \Phi_2 \\ & & L_3 & -I & & & & \Phi_3 \\ & & & \ddots & \ddots & & & \vdots \\ & & & & L_{m-2} & -I & & \Phi_{m-2} \\ & & & & & L_{m-1} & -I & \Phi_{m-1} \end{array}\right]$$

where

$$\begin{aligned} B &= B_0 + B_{m-1}L_{m-1}L_{m-2}\cdots L_2L_1 \text{ and} \\ \bar{\Phi}_0 &= \Phi_0 + B_{m-1}(\Phi_{m-1} + L_{m-1}\Phi_{m-2} + L_{m-1}L_{m-2}\Phi_{m-3} + \cdots + L_{m-1}L_{m-2}\cdots L_3L_2\Phi_1). \end{aligned}$$

3.1 Revised Newton iteration

Let $S^l = (s_0^l, s_1^l, \cdots, s_{m-1}^l)^T$ where $s_i^l \in R^n$. If we apply Newton's method to solve the nonlinear system $\Phi(Y) = 0$, then the l-th Newton iteration (Block Newton iteration or BN iteration) can be determined by solving

$$Bs_0^l = -\bar{\Phi}_0 \quad \text{and} \quad s_i^l = \Phi_i + L_i s_{i-1}^l, \quad i = 1, 2, \cdots, m-1.$$

We can reformulate Algorithm 2.1 in terms of this BN iteration. A multiple shooting algorithm based upon BN iteration is outlined in Algorithms 3.1.

Algorithm 3.1 *(Multiple shooting using BN iteration)*

Given $\Pi\ : a = x_0 < x_1 < \cdots < x_{m-1} = b$ *and* Y^0
For $l = 0, 1, \cdots,$ **do until** *convergence* **or** $l = lmax$
  1 : *Compute* B_0, B_{m-1}, Φ_0 *at* Y^l
  2 : **For** $i = 1, \cdots, m-1$ **do** *Compute* L_i *and* Φ_i *at* Y^l **Endfor**
  3 : *Compute* B *and* $\bar{\Phi}_0$
  4 : *Solve the linear system* $Bs_0^l = -\bar{\Phi}_0$ *and set* $y_0^{l+1} = y_0^l + s_0^l$
  5 : **For** $i = 1, \cdots, m-1$ **do** *Compute* $s_i^l = L_i s_{i-1}^l + \Phi_i$ **Endfor**
  6 : **For** $i = 1, \cdots, m-1$ **do** *Compute* $y_i^{l+1} = y_i^l + s_i^l$ **Endfor**
Endfor

Consider the computational cost required for a single BN iteration. Suppose that the residual function and its Jacobian (B_0, B_{m-1}, Φ_0 and L_i, Φ_i for $i = 1, \cdots, m-1$) are evaluated at the current iterate Y^l and take $T_I(m, n)$ flops. The quantity of $T_I(m, n)$ will depend on the underlying IVP solver used and the specific accuracy requirement. This cost is generally $\Omega(n^3 m)$. The computation of B and $\bar{\Phi}_0$ requires $(m-1)(n^3+n^2+n)+n^2$ flops on a sequential machine. The cost of solving the linear system in Step 4 of the BN iteration (using standard Gaussian elimination with partial pivoting) is $(\frac{1}{3}n^3+n^2)$ flops. The cost of updating y_0^{l+1} is n flops. Finally, the total cost of obtaining s_i^l and y_i^{l+1}, for $i = 1, \cdots, m-1$, is $(m-1)(n^2+2n)$ flops. By taking all of these costs into account, a single BN iteration on a sequential machine requires

$$T_I(m, n) + m(n^3 + O(n^2))\ flops.$$

We observe that a Block-Newton iteration is similar to an iteration of the compactification algorithm [AMR88] or an iteration of the condensation algorithm [DB82]. We now proceed to discuss a parallelization of this BN iteration. This approach employs parallel prefix and fan-in associative operations ([KR] and [J92]). Other existing approaches are discussed in [Wri92, Wri94] and [JP92].

3.2 Exploiting parallelization

We observe that the BN iteration results in first-order linear recurrences involving vector and matrix operations. We realize that such linear recurrences can be solved effectively in parallel using the prefix operation, see [J92]. Referring to the Algorithm 3.1, we must first construct B and $\bar{\Phi}_0$ in Step 3 before we can determine s_0^l in Step 4 and solve a linear recurrence for s_i^l in Step 5. This is a major drawback that makes the execution of Steps 3 and 4-5 inherently sequential. However, we observe that both Steps 3 and 4-5 involve matrix products of L_i's and these matrix products can be computed efficiently in parallel using fast parallel prefix matrix multiplications. After these matrix products are computed, we can determine the transformation and solve the linear recurrence efficiently. This observation enables us to reformulate the BN iteration and results in a new parallel iteration scheme. We will refer to this parallel reformulation of the BN iteration as the Parallel Block-Newton (PBN) iteration. Algorithm 3.2 is a direct derivation from the serial BN iteration, Algorithm 3.1.

Algorithm 3.2 *(Multiple shooting using PBN iteration I)*

Given $\Pi\ : a = x_0 < x_1 < \cdots < x_{m-1} = b$ *and* Y^0
For $l = 0, 1, \cdots,$ **do until** *convergence* **or** $l = lmax$
 1 : *Compute* B_0, B_{m-1}, Φ_0 *at* Y^l *and set* $P_0 = I$
 2 : **For** $i = 1, \cdots, m-1$ **pardo** *compute* $L_i,\ \Phi_i$ *at* Y^l **Endfor**
 3 : **For** $i = 1, \cdots, m-1$ **pardo using prefix multiplcations**
  Compute $P_i = L_i L_{i-1} \cdots L_1$ *and* $\bar{P}_i = L_{m-1} L_{m-2} \cdots L_{m-i}$
  Endfor
 4 : **For** $i = 1, \cdots, m-2$ **pardo** *Compute* $\rho_i = \bar{P}_i \Phi_{m-i-1}$ **Endfor**
 5 : **Pardo using parallel fan-in additions** *Compute* $\tau = \Phi_{m-1} + \sum_{i=1}^{m-2} \rho_i$
 6 : *Compute* $B = B_0 + B_{m-1} P_{m-1},\ \bar{\Phi}_0 = \Phi_0 + B_{m-1}\tau$
 7 : *Solve linear system* $B\beta_0 = -\bar{\Phi}_0$
 8 : **For** $i = 1, \cdots, m-1$ **pardo** *Solve linear system* $P_i\beta_i = \Phi_i$ **Endfor**
 9 : **For** $i = 1, \cdots, m-1$ **pardo using prefix additions**
  Compute $z_i = \sum_{j=0}^{i-1} \beta_j$
  Endfor
 10 : **For** $i = 0, \cdots, m-1$ **pardo** *Compute* $s_i^l = P_i z_{i+1}$ **Endfor**
 11 : **For** $i = 0, \cdots, m-1$ **pardo** *Compute* $y_i^{l+1} = y_i^l + s_i^l$ **Endfor**
Endfor

Indeed, Algorithms 3.2 and 3.1 will compute the same Y^{l+1} if exact arithmetic is used. This fact is assured by the following theorem.

Theorem 3.1 *The PBN and the BN formulations of multiple shooting are algorithmically equivalent under an exact arithmetic assumption.*

Proof : Assume that exact arithmetic is used. It suffices to show that both iterations will compute the same correction vector S^l. Consider the BN iteration and let

$$\sigma_i = \begin{pmatrix} s_i^l \\ 1 \end{pmatrix} \quad \text{and} \quad A_i = \begin{pmatrix} L_i & \Phi_i \\ 0 & 1 \end{pmatrix}.$$

Then, the BN iteration can be rewritten as $\sigma_i = A_i \sigma_{i-1}$ or $\sigma_i = A_i A_{i-1} \cdots A_1 \sigma_0$. Explicitly, we obtain $A_i A_{i-1} \cdots A_1 = \begin{pmatrix} P_i & Q_i \\ 0 & 1 \end{pmatrix}$ where $P_i = L_i L_{i-1} \cdots L_1$ and $Q_i = L_i L_{i-1} \cdots L_2 \Phi_1 + L_i L_{i-1} \cdots L_3 \Phi_2 + \cdots + L_i \Phi_{i-1} + \Phi_i$. By extracting s_i^l from σ_i, we have

$$\begin{aligned} s_i^l &= P_i s_0^l + L_i L_{i-1} \cdots L_2 \Phi_1 + L_i L_{i-1} \cdots L_3 \Phi_2 + \cdots + L_i \Phi_{i-1} + \Phi_i \\ &= P_i s_0^l + P_i P_1^{-1} \Phi_1 + P_i P_2^{-1} \Phi_2 + \cdots + P_i P_{i-1}^{-1} \Phi_{i-1} + P_i P_i^{-1} \Phi_i \\ &= P_i (s_0^l + P_1^{-1} \Phi_1 + P_2^{-1} \Phi_2 + \cdots + P_{i-1}^{-1} \Phi_{i-1} + P_i^{-1} \Phi_i) \end{aligned}$$

Now, we let $\beta_0 = s_0^l$ and $\beta_k = P_k^{-1} \Phi_k$, $k = 1, \cdots, i$, and $z_i = \beta_0 + \cdots + \beta_{i-1}$. Thus, we have $s_i^l = P_i z_{i+1}$. These formulas show the equivalence between the PBN and BN iterations. $\odot$

Consider the computation of s_i^l for $i = 1, \cdots, m-1$. We observe that the PBN iteration (Steps 8-11 of Algorithm 3.2) requires a total of $O(n^3 m)$ operations. In contrast, the corresponding steps (Steps 5-6 of Algorithm 3.1) in the BN iteration requires a total of $O(n^2 m)$ operations. Thus, the PBN iteration requires more operations than the BN iteration. However, the proof of Theorem 3.1 further suggests an alternate formulation of the PBN iteration which is given in Algorithm 3.3. This new PBN formulation requires only a total of $O(n^2 m)$ operations in obtaining s_i^l for $i = 1, \cdots, m-1$ (see Steps 8-10 of Algorithm 3.3).

Algorithm 3.3 *(Multiple shooting using PBN iteration II)*

Given Π : $a = x_0 < x_1 < \cdots < x_{m-1} = b$ *and* Y^0
For $l = 0, 1, \cdots,$ **do until** *convergence* **or** $l = lmax$

1 : *Compute* B_0, B_{m-1}, Φ_0 *at* Y^l *and set* $A_0 = I$ *and* $\bar{P}_0 = I$

2 : **For** $i = 1, \cdots, m-1$ **pardo** *Compute* L_i, Φ_i *at* Y^l *and construct* $A_i = \begin{pmatrix} L_i & \Phi_i \\ 0 & 1 \end{pmatrix}$ **Endfor**

3 : **For** $i = 1, \cdots, m-1$ **pardo using prefix multiplcations**
Compute $\mathcal{A}_i = A_i A_{i-1} \cdots A_1$ *and* $\bar{P}_i = L_{m-1} L_{m-2} \cdots L_{m-i}$
Endfor

4 : **For** $i = 1, \cdots, m-2$ **pardo** *Compute* $\rho_i = \bar{P}_i \Phi_{m-i-1}$ **Endfor**

5 : **Pardo using parallel fan-in additions** *Compute* $\tau = \Phi_{m-1} + \sum_{i=1}^{m-2} \rho_i$

6 : *Compute* $B = B_0 + B_{m-1} P_{m-1}$, $\bar{\Phi}_0 = \Phi_0 + B_{m-1} \tau$

7 : *Solve* $B \beta_0 = -\bar{\Phi}_0$ *and construct* $\sigma_0 = \begin{pmatrix} s_0^l \\ 1 \end{pmatrix}$

8 : **For** $i = 1, \cdots, m-1$ **pardo** *Compute* $\sigma_i = \mathcal{A}_i \sigma_0$ **Endfor**

9 : **For** $i = 0, \cdots, m-1$ **pardo** *Extract* s_i^l *from* $\sigma_i = \begin{pmatrix} s_i^l \\ 1 \end{pmatrix}$ **Endfor**

10 : **For** $i = 0, \cdots, m-1$ **pardo** *Compute* $y_i^{l+1} = y_i^l + s_i^l$ **Endfor**
Endfor

Before analysing the cost of the PBN iteration, we will describe in detail the distributed parallel prefix and fan-in operations. In a distributed memory machine, processors are connected using a message passing interconnection network. Each processor has its own local memory that is accessible only by that processor. Also, each processor can be uniquely identified by an integer k which is referred to as the processor number. Processors can interact and cooperate only by passing messages. In order to describe the distributed parallel fan-in and prefix algorithms on a distributed memory machine, we introduce the following terminology.

Let k be any natural number and $k^i = k \otimes 2^i$ where $\otimes$ is the binary exclusive-or operation. The relationship between k and k^i is that their binary representations differ only in the i-th position. By convention, the rightmost bit of binary representation of k is the zero bit position and the bit position number will increase from right to left. Also, one can show $k = (k^i)^i$. Let $*$ be any associative operation over a domain D. Given a set of $m > 1$ elements from D, suppose there are m processors available in a machine where m is a power of 2. Note that all logarithms are base 2.

The fan-in operation on $m > 1$ elements from D will result in another element $d_{0,m-1} = d_0 * d_1 * \cdots * d_{m-1}$. The fan-in operation can be done using the following distributed parallel algorithm (Algorithm 3.4). Assume that processor p_k holds an element d_k initially for $k = 0, \cdots, m-1$. When the distributed parallel fan-in algorithm terminates, processor p_{m-1} holds the final result $d_{0,m-1}$.

Algorithm 3.4 *(Distributed fan-in operation)*

For $i = 1, \cdots, \log m$ **do**
  For $k = 0, \cdots, m-1$ **active processor** p_k **pardo**
    If $k > k^i$ **then** *recv (* p_{k^i}, d *) and compute* $d_k = d * d_k$
        else *send (* p_{k^i}, d_k *) and stop*
    Endif
  Endfor
Endfor

Referring to Algorithm 3.4, a processor executing the 'stop' instruction becomes inactive after the execution of this statement. A processor p_k executing the recv(p, d) instruction blocks the execution of its program until data from processor p is received. It then stores the data in d and resumes the execution of its program. A processor p_k executing the send(p, d) instruction sends a copy of d to processor p, then resumes execution of the next instruction immediately.

The distributed parallel prefix operation on $m > 1$ elements from D will result in another m elements $d_{0,i} = d_0 * d_1 * \cdots * d_i$ for $i = 0, \cdots, m-1$. The prefix operation can be performed using Algorithm 3.5. Assume that processor p_i holds an element d_i initially for $i = 0, \cdots, m-1$. When the distributed parallel prefix algorithm terminates, processor p_i holds the result $d_{0,i}$ for $i = 0, 1, \cdots, m-1$.

Algorithm 3.5 *(Distributed prefix operation)*

Perform a distributed fan-in operation on D and re-activate all processors
For $i = \log m - 1, \cdots, 1$ **do**
  For $j = 2^i - 1, \cdots, m - 2^{i-1} - 1$ **step** 2^i **pardo**
    Processor p_j **do** *send ($p_{j+2^{i-1}}, d_j$)*
    Processor $p_{j+2^{i-1}}$ **do** *recv (p_j, d) and compute* $d_{j+2^{i-1}} = d * d_{j+2^{i-1}}$
  Endfor
EndFor

Note that Algorithm 3.5 involves applying Algorithm 3.4 once, which takes $\log m$ parallel steps. In addition, it requires an extra $\log m - 1$ parallel steps to complete the prefix operation. Thus, Algorithm 3.5 takes $2\log m - 1$ parallel steps. It can be shown that, if the operation $*$ is commutative over D, one can construct the prefix operation using only $\log m$ parallel steps. Since in our application (Algorithm 3.3) we are concerned with matrix multiplication which is not commutative, we cannot exploit this observation.

We can determine the parallel arithmetic complexity of the PBN iteration (Algorithm 3.3) based on a distributed memory model. Suppose that $T_{I_0}(n)$ is the time required to compute the matrices B_0, B_{m-1}, and vector Φ_0, and $T_{I_i}(n)$ is the time required to computed the matrices L_i, and vector Φ_i for $i = 1, \cdots, N$. Recall that $T_I(m,n)$ is the total time required to compute all these matrices and vectors on a sequential machine. Thus, we have $T_I(m,n) = \sum_{i=0}^{m-1} T_{I_i}(n)$.

The maximum cost over all processors for Step 1 and 2 of Algorithm 3.3 is $\max_{0\le i\le m-1} T_{I_i}(n)$. Step 3 takes $2\log m - 1$ parallel steps and each parallel step requires one $(n+1)\times(n+1)$ matrix-matrix multiplication and one $n \times n$ matrix-matrix multiplication. Thus, the total arithmetic cost is $(2\log m - 1)((n+1)^3 + n^3)$ flops for Step 3. Step 4 requires n^2 flops. Step 5 requires $n \log m$ flops. Step 6 takes $(n^3 + 2n^2 + 2n)$ flops. The cost of solving an $n \times n$ linear system in Step 7 of PBN iteration is $(\frac{1}{3}n^3 + n^2)$ flops (if Gaussian elimination with partial pivoting is used). Step 8 requires $(n+1)^2$ flops. The cost of updating y_i^{l+1} is n flops. By taking all of these costs into account, the PBN iteration requires $(\max_{0\le i\le m-1} T_{I_i}(n) + (4n^3 + 6n^2 + 7n + 2)\log m - (\frac{2}{3}n^3 - 2n^2 - n))$ flops. This implies the parallel arithmetic cost of the PBN iteration is

$$\max_{0\le i\le m-1} T_{I_i}(n) + (\log m - 1)(4n^3 + O(n^2)) \quad flops.$$

Note that this expression does not include the data communication cost required by the processors. The communication complexity of a parallel algorithm for a distributed memory model is defined as the worst-case bound on the amount of message passing between any two processors. This worst-case bound depends the actual mapping of a parallel algorithm onto a distributed memory model (with its associated communication paths and bandwidths).

We now turn to the actual mapping of the PBN iteration onto a distributed memory machine model. Referring to Algorithm 3.3, the computation of Φ_i and the corresponding Jacobian matrix components; the subsequent forming of $A_i, \mathcal{A}_i, \bar{P}_i, \rho_i, \sigma_i, s_i^l$, and y_i^{l+1} $(i = 0, \cdots, m-1)$ corresponding to each subinterval of the mesh Π can be done using a single processor. Thus, we can assign the computations associated with each mesh subinterval to a processor. Suppose that there are m processors available and m is a power of 2. Processor p_i performs the computations corresponding to mesh subinterval $[x_{i-1}, x_i]$ for $i = 1, \cdots, m-1$ and p_0 performs the computation related to the boundary conditions.

In order to study the communication complexity of the PBN iteration, we need to know the underlying communication routing technique used in the distributed memory machine model. Suppose that the store-and-forward routing technique [KGGK94] is employed. The total communication time of a message of size b units to traverse c_d communication links (diameter) between two processors is characterized by the equation $T(b, c_d) = t_s + c_d t_w b$ where t_s is the constant overhead for any message passing and t_w is the per-unit transfer time. Notice that the unit measurement of time and size depend on the nature of the data elements. For example, the size unit can be a double precision floating point number and the time unit can be a millisecond. The communication diameter, c_d, of a distributed memory machine is the maximum distance (in terms of number of links) between any two processors. For example, $c_d = 1$ for a fully interconnected network, $c_d = \log m$ for a hypercube interconnected network, and $c_d = \frac{m}{2}$ for a ring interconnected network where m is the number of processors. Similarly, one can derive the formula, $T_a(b, c_d) = c_d(t_s + t_w b)$, of the total communcaiton time required for broadcasting a message from one processor to all processors.

Consider a single PBN iteration in Algorithm 3.3. Initially, we assume that $x_0, y_0, x_{m-1}, y_{m-1}$ and $x_{i-1}, y_{i-1}, x_i, y_i$ have been distributed to processor p_0 and processor p_i respectively for $i = 1, \cdots, m-1$. We observe that there is no communication required by Steps 1, 2, 6, 7, 9 and 10. In Step 3, the message passing associated with the computation of $\mathcal{A}_i$ requires $2\log m - 1$ parallel steps and each step requires either sending or receiving a message of size $(n+1)^2$. Thus, the computation of $\mathcal{A}_i$ takes $(2\log m - 1)T((n+1)^2, c_d) = (2\log m - 1)(t_s + c_d t_w (n+1)^2)$ time units. To compute $\bar{P}_i$, processors p_i and p_{m-i} must exchange the $n \times n$ matrices L_i and L_{m-i} for $i = 1, \cdots, m-1$. This data exchange can be done in a single parallel step and hence costs $T(n^2, c_d) = (t_s + c_d t_w n^2)$ time units. Again, we compute $\bar{P}_i$ for $i = 1, \cdots, N$ using parallel prefix multiplication which requires $(2\log m - 1)T(n^2, c_d) = (2\log m - 1)(t_s + c_d t_w n^2)$ time units.

In step 4, processors p_i and p_{m-i-1} must exchange Φ_i and Φ_{m-i-1}. This data exchange can also be done in a single parallel step and costs $T(n, c_d) = (t_s + c_d t_w n)$ time units. In step 5, a parallel fan-in vector addition is required to obtain τ and requires $\log m$ parallel steps. Each parallel step either sends or receives a vector of size n. Thus, Step 5 costs

$T(n, c_d) \log m = (t_s + c_d t_w n) \log m$. Finally, in Step 8, each processor p_i must receive s_0^l from p_0 in order to compute s_i^l. This can be done in a single parallel broadcasting step and costs $T_a(n, c_d) = c_d(t_s + t_w n)$. By taking all the above costs into account, the total communication time is $(5 \log m)t_s + c_d t_w((4n^2 + 5n + 2) \log m - (n^2 + 1))$. If the condition $nt_w \gg t_s > 0$ holds, then the total commmunication time required by a single PBN iteration is bounded by

$$(1 + c_d)O(n^2 \log m) \tag{3}$$

time units.

Numerical difficulties may arise when using any implementation of multiple shooting. We now discuss a special iterative refinement scheme which can be used with our parallel shooting algorithm to help cope with such difficulties.

3.3 Distributed iterative refinement scheme

We consider two sources of error that can arise and affect the solutions of BVODE problems, namely truncation and rounding errors. With a multiple shooting algorithm, truncation errors arise in obtaining approximations to L_i and $v_{i-1}(x_i; y_{i-1})$. A major numerical difficulty can arise in the propagation of truncation and rounding errors when the Newton correction vector S^l is formed. As a consequence, the vector S^l can lose its significance. Referring to Algorithm 3.1, truncation and rounding errors can both contribute to the determination of Φ_i in Steps 1-2 and may be magnified in Steps 3-4. Then, these errors can be propagated through Step 5 and are accumulated in each component of S^l.

The question now is how stability properties with respect to these errors can be improved. One way to improve the stability of the computation of S^l would simply be to apply an iterative refinement scheme to the linear system arising from the Newton iteration. Iterative refinement is a well-known technique for improving a computed approximation, x, to a given linear system, $A\bar{x} = b$. A single refinement step consists of three stages. The first stage computes the error residual, $r = Ax - b$. The second stage then solves the linear system, $A\bar{y} = r$, to determine a correction, y, of x. The final stage corrects x to form $z = x + y$. An 'iterative' refinement scheme allows us to repeat this refinement step with x replaced by z and to continue this iteration until some accuracy requirement is satisfied.

The idea behind iterative refinement is that some improvement in the accuracy of the approximate solution will be obtained on each step if r and y are computed accurately enough. If there are no rounding errors associated with the computation of r, y, and z, then z would be the exact solution to the given system. In floating point arithmetic, the computation of r can be done in higher precision. However, it is also common to carry out a single refinement step at the standard working precision. The use of iterative refinement at working precision [JW77, Ske80] has been advocated and shown to be effective. One iterative refinement step will usually be enough to improve the accuracy of the computed solution, see [Hig96]. Now, we proceed to derive a working precision iterative refinement scheme that can be applied in our parallel approach.

Consider the solution of the linear system, $\bar{J}S^l = -\bar{\Phi}$. We examine the stability of the exact solution, $\bar{S}^l$, relative to a perturbation, $d\bar{\Phi}$, of the right side, $\bar{\Phi}$. In block form, we let $d\bar{\Phi}_i$ be the perturbation of the ith block component of $\bar{\Phi}$ for $i = 0, \cdots, m-1$. We denote $d\bar{s}_i^l$ as the ith block component of the induced perturbations, $d\bar{S}^l$, of the exact solution, $\bar{S}^l$, for $i = 0, \cdots, m-1$. The perturbation, $d\bar{\Phi}$, of the right side, $\bar{\Phi}$, can be interpreted as arising from the inherent propagation effect of the recursion associated with Step 4 and 5 of Algorithm 3.1. To illustrate how large the perturbation, $d\bar{S}^l$, of $\bar{S}^l$ can be, we assume that L_i has been well approximated for $i = 1, \cdots, m-1$. In Step 4 of Algorithm 3.1, the block component $\bar{s}_0^l + d\bar{s}_0^l$ of the Newton correction vector is obtained from the perturbated system, $Bs_0^l = -(\bar{\Phi}_0 + d\bar{\Phi}_0)$. This induced perturbation, $d\bar{s}_0^l$, will be propagated (and possibly amplified) by the recursion scheme and accumulated in $d\bar{s}_i^l$ of $\bar{s}_i^l$, for $i = 1, \cdots, m-1$. The perturbations introduced in this way can be viewed as block-wise perturbations of $\bar{\Phi}$ and $\bar{J}$.

Theorem 3.2 *Let $\bar{s}_0^l$ and $\bar{s}_0^l + d\bar{s}_0^l$ be the exact solutions of the linear systems, $Bu = -\bar{\Phi}_0$ and $Bu = -(\bar{\Phi}_0 + d\bar{\Phi}_0)$, respectively. If $\bar{s}_i^l + d\bar{s}_i^l$ are determined by applying Algorithm 3.1 in exact arithmetic, then the induced perturbation, $d\bar{s}_i^l$, of $\bar{s}_i^l$ due to the perturbation, $d\bar{\Phi}$, of $\bar{\Phi}$ satisfies*

$$\| d\bar{s}_i^l \| \leq K^i D(\kappa + i)$$

where $i = 1, \cdots, m-1$, $K = \max\{\max_{1\leq j\leq m-1} \| L_j \|, 1\}$, $\kappa = \| B^{-1} \|$, and $D = \max_{0\leq j\leq m-1} \| d\bar{\Phi}_j \|$.

Proof : The induced perturbations, $d\bar{s}_i^l$ for $i = 0, \cdots, m-1$, will satisfy the following equations

$$Bd\bar{s}_0^l = -d\bar{\Phi}_0 \quad \text{and} \quad d\bar{s}_i^l = d\bar{\Phi}_i + L_i d\bar{s}_{i-1}, \quad i = 1, \cdots, m-1.$$

The proof follows using a straightforward induction argument. $\odot$

Referring to Theorem 3.2, the number κ is a conditioning constant of the BVODE and the number K is a constant related to the condition number of J (see [DB82, AC91]) and the constant K^{m-1} reflects the condition of the associated initial value problems. We observe that even if a BVODE problem is well-conditioned, i.e. κ is of moderate size, but the corresponding initial value problems are ill-conditioned, i.e., $K \gg 1$, then the induced error bound is large, and hence the propagated error, $d\bar{s}_i^l$, can dominate the underlying solution of the recursion. However, the size of the errors introduced in this way can be monitored by estimating the value of $K^{m-1}D(\kappa + m - 1)$.

In order to apply a single step of iterative refinement to improve a computed solution, S^l, we need to determine the associated error residual, r_i for $i = 0, \cdots, m-1$. To determine an appropriate vector r_0, we determine the error residual of the boundary conditions using the residual function, Φ, and then determine the effect of the block elimination process. We compute

$$\gamma = B_1 s_0^l + B_{m-1} s_{m-1}^l + \Phi_0 \tag{4}$$

and then determine r_0 according to the block elimination process where Φ_0 is replaced by γ. For $i = 1, \cdots, m-1$, r_i is determined using

$$r_i = L_i s_{i-1}^l + \Phi_i - s_i^l. \tag{5}$$

After all the r_i's have been determined, we solve the linear system, $Bds_0^l = -r_0$, for the induced error ds_0^l. Applying the same recursion principle in Step 5 of Algorithm 3.1, one can obtain the successful induced error term ds_i^l for $i = 1, \cdots, m-1$ and update the Newton correction vector by setting $s_i^l = s_i^l + ds_i^l$ for $i = 0, \cdots, m-1$. A serial version of the iterative refinement step is described as follows.

Algorithm 3.6 *(Serial iterative refinement)*

5.1 : *Compute* $\gamma = B_1 s_0^l + B_{m-1} s_{m-1}^l + \Phi_0$
5.2 : **For** $i = 1, \cdots, m-1$ **do** $r_i = L_i s_{i-1}^l + \Phi_i - s_i^l$ **Endfor**
5.3 : *Compute* $r_0 = \gamma + B_{m-1}(r_{m-1} + L_{m-1} r_{m-2} + \cdots + L_{m-1} L_{m-2} \cdots L_2 r_1)$
5.4 : *Solve the linear system* $Bds_0^l = -r_0$
5.5 : **For** $i = 1, \cdots, m-1$ **do** *Compute* $ds_i^l = r_i + L_i ds_{i-1}^l$ **Endfor**
5.6 : **For** $i = 0, \cdots, m-1$ **do** *Compute* $s_i^l = s_i^l + ds_i^l$ **Endfor**

The arithmetic complexity of a single refinement step is $O(n^2 m)$. Algorithm 3.6 can be parallelized easily by using the same techniques described in section 3.2. A parallel extension of the serial refinement step is given in Algorithm 3.7.

Algorithm 3.7 *(Parallel iterative refinement)*

9.1 : *Compute* $\gamma = B_1 s_0^l + B_{m-1} s_{m-1}^l + \Phi_0$
9.2 : **For** $i = 1, \cdots, m-1$ **pardo** $r_i = L_i s_{i-1}^l + \Phi_i - s_i^l$ **Endfor**
9.3 : **For** $i = 1, \cdots, m-2$ **pardo** *Compute* $\rho_i = \bar{P}_i r_{m-i-1}$ **Endfor**
9.4 : **Pardo using parallel fan-in additions** *Compute* $\tau = \gamma + \sum_{i=1}^{m-2} \rho_i$
9.5 : *Compute* $r_0 = \gamma + B_{m-1}\tau$
9.6 : *Solve the linear system* $Bds_0^l = -\sigma_0$ *and construct* $d\sigma_0 = \begin{pmatrix} ds_0^l \\ 1 \end{pmatrix}$
9.7 : **For** $i = 1, \cdots, m-1$ **pardo** *Compute* $d\sigma_i = \mathcal{A}_i d\sigma_0$ **Endfor**
9.8 : **For** $i = 0, \cdots, m-1$ **pardo**

Extract ds_i^l *from* $d\sigma_i = \begin{pmatrix} ds_i^l \\ 1 \end{pmatrix}$ *and compute* $s_i^l = s_i^l + ds_i^l$

Endfor

The parallel time complexity of a single parallel refinement step is $O(n^2 + n \log m)$.

4 Computational Results

We have tested our approach by modifying an existing widely used multiple shooting algorithm (BVPMS). We have implemented several different versions of this multiple shooting algorithm on both serial and distributed memory machines using PVM interfaces. The acronyms for the different implementations of the multiple shooting algorithm are given in Table 1. All modifications of the multiple shooting algorithm employ dverk as the initial

Acronym	IVP Integrator	Equation Solver	Iterative Refinement	Execution mode
j-bvpms	dverk	Jepson	$\times$	Serial
gir-j-bvpms	dverk	Jepson	$\surd$	Serial
bn-bvpms	dverk	Block-Newton	$\times$	Serial
gir-bn-bvpms	dverk	Block-Newton	$\surd$	Serial
pibn-bvpms	dverk	Block-Newton	$\times$	Parallel
pbn-bvpms	dverk	Block-Newton	$\times$	Parallel
gir-pbn-bvpms	dverk	Block-Newton	$\surd$	Parallel

Table 1: Description of Algorithms

value problem solver. dverk is a routine based on Verner's fifth and sixth order pair of Runge-Kutta formulas [HEJ76] for finding approximations to the solution of an initial value problem. The prefix 'gir' is used to distinguish the algorithms that apply a single step of iterative refinement. All implementations are written in Fortran in double precision. These implementations are executed either on a serial machine (a SUN SPARCstation) or on a distributed memory parallel machine (a cluster of SUN SPARCstations). The complete set of test problems is given in Appendix A. Computational experience on this set of BVODE problems is presented in Appendix B to demonstrate the potential effectiveness of the numerical approach we have developed.

4.1 Serial implementations and results

We first present results of serial implementations of the modified multiple shooting algorithms on a SUN SPARCstation. These modified multiple shooting algorithms employ the BN iteration. We compare our results with an existing multiple shooting algorithm, namely j-bvpms . We will refer to our serial modification of the multiple shooting algorithm as bn-bvpms . Recall that applying these variants of the Newton iteration will reduce to solving systems of linear equations perhaps of different sizes. j-bvpms employs the Jepson's banded linear system solver [Jep] to directly solve the almost block bidiagonal system, $JS^l = -\Phi$, within each Newton iteration. This solution technique is designed to be as stable as possible. Gaussian elimination with column pivoting is used in the solution of the system. bn-bvpms is an implementation of Algorithm 3.1 and solves the reduced system, $Bs_0^l = -\bar{\Phi}_0$, using the

IMSL dense linear system solver that solves a system of linear equations with LU factorization.

We have applied j-bvpms and bn-bvpms to solve the set of test problems specified in Appendix A. We tested these algorithms using a variety of shooting points with different error tolerances in the range 10^{-5} and 10^{-10}. Shooting points were equally distributed within the problem interval. We now discuss some of the results we observed. j-bvpms and bn-bvpms could solve Problems A.1 and A.2 using the same number of iterations. The results are recorded in Tables 2 and 3. These tables give the required number of iterations (no. iter), the absolute maximum component of the residual function at convergence (max. error), and the required processing time (t) for the Newton iteration to converge. We observe that bn-bvpms consistently requires less processing time than j-bvpms . The last column in both Tables 2 and 3 give the percent of savings in time, S_T, for bn-bvpms relative to j-bvpms . From the system execution profile, we observed that the memory storage requirement for the bn-bvpms is much less than j-bvpms . For example, when we employ a mesh size of $m = 1024$, j-bvpms requires approximately 5.5 Mbytes, but bn-bvpms requires only 1.5 Mbytes. Both processing time and space improvements are expected since bn-bvpms does not need to construct, store, and solve an $mn \times mn$ linear system as j-bvpms does for each iteration. This savings become more significant when m increases. The computational results in Tables 4-10 and 19 show that bn-bvpms generally outperforms j-bvpms . In many of the test problems, bn-bvpms requires fewer iterations to converge. The reason for this improvement is not obvious since the algorithms differ only in the linear system solver that is used in the Newton iteration. One would not expect such a difference in the required number of iterations. We observe that the only possible cause of this difference is related to the numerical stability of the associated linear system solvers. The stability of j-bvpms and bn-bvpms are closely related to the condition of the linear systems solved within the Newton iteration.

The prefix 'gir' is used to distinguish the algorithm that applies a single (global) iterative refinement step. For example, gir-j-bvpms uses the j-bvpms algorithm with the application of a single step of iterative refinement after each iteration. For gir-bn-bvpms , we have implemented an iterative refinement scheme based on the refinement step given in Algorithm 3.6. Tables 7-11 show that an iterative refinement step can significantly improve the performance of j-bvpms . Also, gir-bn-bvpms improves upon bn-bvpms for some difficult test problems (see Tables 11-16). A single iterative refinement step, however, does not always improve the performance of j-bvpms and bn-bvpms . We observe that this scenario will particularly occur when solving stiff BVODE problems such as Problem A.11. Table 19 gives the computational result for this test problem. Problem A.11 is difficult because it contains a boundary layer near the left boundary. Both j-bvpms and bn-bvpms could solve this difficult test problem. bn-bvpms outperforms j-bvpms for all meshes. The results of Table 19 show that a single step of iterative refinement could degrade the performance of either j-bvpms or bn-bvpms on such stiff problems.

4.2 Parallel implementations and results

We have designed and implemented two different parallel versions of BVPMS on a distributed memory parallel machine (a cluster of SUN SPARCstations) using the parallel virtural machine (PVM) interface. PVM is a well-known software system developed at Oak Ridge National Labs that allows a network of serial and/or parallel computers to appear as a single distributed memory parallel machine. Different parallel programming paradigms such as a single program multiple data (SPMD) model or a master-slave (MS) model can be used with PVM. Parallel constructs are implemented using the PVM Fortran interfaces. We will refer to these two parallel implementations as pibn-bvpms and pbn-bvpms . Again, we will use the prefix gir to distinguish the parallel algorithms that include a single step of parallel iterative refinement. pibn-bvpms is a parallel implementation of bn-bvpms in which only the numerical integrations associated with the mesh subintervals have been parallelized. pbn-bvpms is a parallel implementation of bn-bvpms which is implemented using the PBN iteration (Algorithm 3.3), and parallel prefix and fan-in operations (Algorithms 3.4 and 3.5). For gir-pbn-bvpms , we have implemented Algorithm 3.7 with a single step of iterative refinement.

We have applied these parallel algorithms to some of the test problems using different numbers of processors. Tables 20-29 report the computational results. They show that these parallel algorithms can attain a significant speedup and considerable savings in time when solving BVODE problems. A lot of information is presented in a compact form. We acknowledge that this form is not easy to comprehend. Note that Tables 20-29 are all presented in the same format. The upper table named (a) gives the PVM timing (in seconds) for each iteration, per-iteration speedup, and the overall speedup for different number of processors. In these tests, the number of processors is always equal to the number of shooting points (see Column 1). The last column indicates the iteration number. For the different iterations, we report the time required for the parallel numerical integrations in Column 3 associated with each subinterval (labelled IVPs). This time is determined by measuring the maximum time over all processes used to complete the parallel numerical integrations. We also recorded the maximum, average, and minimum number of successful steps required by dverk to complete the numerical integrations over all mesh subintervals (see Columns 6-8). Note that these statistics allow us to infer how well the workload within the IVP Phase was balanced. We have also recorded the time required for the complete iteration in Columns 2 and 4. This time is determined by measuring the maximum time over all processes required to complete the iteration. Per-iteration speedup and overall speedup are measured relative to bn-bvpms (see Columns 9 and 10). The percentage of time required in the numerical integration part of an iteration is shown in Column 5. The lower table named (b) gives the speedup and time savings relative to j-bvpms . Column 1 reports the number of shooting points that is equal to the number of processes used in our parallel executions. Column 2 reports the total time required by j-bvpms to converge. Similarly, Column 3 gives the total time required by pbn-bvpms or gir-pbn-bvpms to converge. The second last column shows the overall speedup relative to j-bvpms . The last column shows the % time savings relative to j-bvpms .

Let us consider the results in Table 20. The upper table (a) gives a summary of the computational results of bn-bvpms and pbn-bvpms on the test problem A.2. The dimension of Problem A.2 is 5 and the error tolerance for the residual function components is set to 10^{-5}. Problem A.2 can be solved by bn-bvpms and pbn-bvpms on uniform meshes with sizes 8, 16, 32, 64, and 128. Consider the 8-point mesh, both bn-bvpms and pbn-bvpms required 6 iterations to converge. For the first iteration, bn-bvpms required 8.58445D-02 seconds (see Column 2) whereas pbn-bvpms required 1.51800D-02 seconds (see Column 4). The time for performing the numerical integrations within the first iteration of pbn-bvpms is 9.88550D-03 seconds (see Column 3). Columns 6-8 revealed that the numerical integration requires the same amount of work (i.e., 4 successful steps) to complete the numerical integrations associated with each subinterval. After 6 iterations, we observe that both bn-bvpms and pbn-bvpms converged to the same discrete approximate solution on the given mesh. Column 10 gives the average per-iteration speedup ratio of pbn-bvpms relative to bn-bvpms . This average ratio is determined by averaging all the numbers in Column 9. For Problem A.2 with a mesh size of 8, the average per-iteration speedup is 5.50 (see Column 10).

The last two entries in Column 2 show the overall processing time and the overall numerical integrations (IVPs) time required by bn-bvpms . Comparing these two values, the numerical integrations account for approximately 93% of the overall processing time required by bn-bvpms . Recall that, for pbn-bvpms , the numerical integrations account for approximately 65% of the time in each iteration. This difference can be explained since one incurs a '$\log m$' penalty when parallelizing the linear algebra component of the algorithm. The last entry in Column 3 gives the overall processing time required by pibn-bvpms . Recall that pibn-bvpms is a parallel modification of bn-bvpms in which only the numerical integrations have been parallelized. Similarly, the last entry in Column 4 gives the overall processing time required by pbn-bvpms . The overall processing time for pibn-bvpms and pbn-bvpms using 8 processes is 0.122168 and 9.41265D-02 seconds, respectively. Knowing the overall processing time of bn-bvpms, pbn-bvpms, and pibn-bvpms, we can determine the overall speedup for pbn-bvpms and pibn-bvpms relative to bn-bvpms , see the last two entries of Columns 9 and 10. For Problem A.3 with a mesh size of 8, the overall speedup ratio of pbn-bvpms and pibn-bvpms is 5.37 and 4.14, respectively. We observe from Tables 24 and 25 that, if the problem size is small and the number of mesh points is relatively large, the parallelization of the solution of a linear system may not be beneficial. gir-pbn-bvpms could also achieve significant speedup and time savings over j-bvpms even when a single step of parallel iterative refinement was required to solve the underlying BVODE problems (see Tables 26-29).

5 Concluding remarks

We have investigated the numerical stability of a modified multiple shooting algorithm and identified difficulties that can arise. In many cases, these numerical difficulties can be attributed to the conditioning of the linear equations arising in the Newton iteration and/or to the failure of the Newton iteration to converge. We developed a parallel iterative refinement technique to help cope with these difficulties. We have implemented some of the techniques we have designed and carried out computational experiments to investigate and quantify the improvements. Experimental results show that our modifications can lead to significant improvements in processing time, storage requirements, and the required number of Newton iterations. Most of the techniques developed in this investigation can be adopted by other widely used numerical algorithms such as those based on finite difference, collocation, or implicit Runge-Kutta formulas. We plan to consider such implementations in the future. In general, it is important to balance the workload among different processors in our distributed parallel shooting algorithm. The load balancing issue associated with the initial value problem phase of the algorithm is directly related to the underlying mesh selection scheme. For multiple shooting, most mesh selection schemes are only concerned with 'equidistribution of error' and 'numerical stability' issues. We are currently investigating how one can incorporate 'load balancing' into existing mesh selection heuristics. So far, our analysis and parallel implementation of the multiple shooting are mainly carried out on a distributed memory machine model. However, most of our work can easily be extended to a shared memory machine model.

A BVODE Problems

A set of BVODE problems were chosen from the literatures [Ren76, MS88, EHL75, Hol64, KM93, AMR88, DB82]. Some problems were created by modifying the existing BVODE problems. Modifications were made by either coupling with different boundary conditions or coupling two or more BVODEs to form a larger BVODEs for testing purposes. All BVODE problems have been converted to their equivalent first-order forms. The high-order forms of BVODE problems can be found in the original source of references.

A.1 Superconductivity problem

The Ginzburg-Landau equations can be found in [Ren76].

Superconductivity problem : Ginzburg-Landau Equations	
ODEs	BCs
$y_1' = y_2$ $y_2' = -\frac{1}{x}y_2 + \frac{1}{x^2}y_2 + k^2y_1^3 + k^2y_1y_3^2 - \frac{2k}{x}y_1y_3 - k^2y_1$ $y_3' = y_4$ $y_4' = -\frac{1}{x}y_4 + \frac{1}{x^2}y_3 + y_1^2y_3 - \frac{1}{kx}y_1^2$ where $k = 1.0$ and $R = 0.5$	$y_1(0.5) = 0.0$ $y_2(10) = 0.0$ $y_3(0.5) = 0.0$ $y_4(10) - R = 0.0$
Initial Estimate	
$y_1(x) = \frac{x(15.0-x)}{50.0},\ y_2(x) = \frac{(15.0-2.0x)}{50.0},\ y_3(x) = \frac{Rx}{10.0},\ y_4(x) = \frac{R}{10.0}$	

A.2 MUS test problem

Problem A.2 can be found in the software package MUS by [MS88].

MUS test problem I	
ODEs	BCs
$y_1' = \frac{0.5y_1(y_3-y_1)}{y_2}$ $y_2' = -0.5(y_3 - y_1)$ $y_3' = \frac{0.9-1000.0(y_3-y_5)-0.5y_3(y_3-y_1)}{y_4}$ $y_4' = 0.5(y_3 - y_1)$ $y_5' = -100.0(y_5 - y_3)$	$y_1(0) - 1.0 = 0.0$ $y_2(0) - 1.0 = 0.0$ $y_3(0) - 1.0 = 0.0$ $y_4(0) + 10.0 = 0.0$ $y_3(1) - y_5(1) = 0.0$
Initial Estimate	
$y_1(x) = 1.0,\ y_2(x) = 1.0,\ y_3(x) = -4.5x^2 + 8.91x + 1.0,\ y_4(x) = -10.0,$ $y_5(x) = -4.5x^2 + 9.0x + 0.91$	

A.3

Problem A.3 is same as Problem A.2 but coupled with different boundary conditions.

MUS test problem II	
ODEs	BCs
$y_1' = \frac{0.5y_1(y_3-y_1)}{y_2}$	$y_1(0) - 1.0 = 0.0$
$y_2' = -0.5(y_3 - y_1)$	$y_2(0) - 1.0 = 0.0$
$y_3' = \frac{0.9-1000.0(y_3-y_5)-0.5y_3(y_3-y_1)}{y_4}$	$y_3(0) - 1.0 + y_3(1) + 0.208 * y_4(1) = 0.0$
$y_4' = 0.5(y_3 - y_1)$	$y_4(0) + 10.0 + y_4(1) + 4.806 * y_5(1) = 0.0$
$y_5' = -100.0(y_5 - y_3)$	$y_5(0) - 0.968 + y_3(1) - y_5(1) = 0.0$
Initial Estimate	
$y_1(x) = 1.0,\ y_2(x) = 1.0,\ y_3(x) = -4.5x^2 + 8.91x + 1.0,\ y_4(x) = -10.0,$ $y_5(x) = -4.5x^2 + 9.0x + 0.91$	

A.4

Problem A.4 is same as Problem A.2 but coupled with different boundary conditions.

MUS test problem III	
ODEs	BCs
$y_1' = \frac{0.5y_1(y_3-y_1)}{y_2}$	$y_1(0) - 1.0 = 0.0$
$y_2' = -0.5(y_3 - y_1)$	$y_2(0) - 1.0 + y_2(1) - 0.353 * y_3(1) = 0.0$
$y_3' = \frac{0.9-1000.0(y_3-y_5)-0.5y_3(y_3-y_1)}{y_4}$	$y_3(0) - 1.0 + y_3(1) + 0.208 * y_4(1) = 0.0$
$y_4' = 0.5(y_3 - y_1)$	$y_4(0) + 10.0 + y_4(1) + 4.806 * y_5(1) = 0.0$
$y_5' = -100.0(y_5 - y_3)$	$y_5(0) - 0.968 + y_3(1) - y_5(1) = 0.0$
Initial Estimate	
$y_1(x) = 1.0,\ y_2(x) = 1.0,\ y_3(x) = -4.5x^2 + 8.91x + 1.0,\ y_4(x) = -10.0,$ $y_5(x) = -4.5x^2 + 9.0x + 0.91$	

A.5

Problem A.5 is same as Problem A.2 but coupled with different boundary conditions.

MUS test problem IV	
ODEs	BCs
$y_1' = \frac{0.5y_1(y_3-y_1)}{y_2}$	$y_1(0) - 1.0 + y_1(1) - 1.966 * y_2(1) = 0.0$
$y_2' = -0.5(y_3 - y_1)$	$y_2(0) - 1.0 + y_2(1) - 0.353 * y_3(1) = 0.0$
$y_3' = \frac{0.9-1000.0(y_3-y_5)-0.5y_3(y_3-y_1)}{y_4}$	$y_3(0) - 1.0 + y_3(1) + 0.208 * y_4(1) = 0.0$
$y_4' = 0.5(y_3 - y_1)$	$y_4(0) + 10.0 + y_4(1) + 4.806 * y_5(1) = 0.0$
$y_5' = -100.0(y_5 - y_3)$	$y_5(0) - 0.968 + y_3(1) - y_5(1) = 0.0$
Initial Estimate	
$y_1(x) = 1.0,\ y_2(x) = 1.0,\ y_3(x) = -4.5x^2 + 8.91x + 1.0,\ y_4(x) = -10.0,$ $y_5(x) = -4.5x^2 + 9.0x + 0.91$	

A.6 Chemistry problem

Problem A.6 is an initial value problem which can be found in [EHL75].

Chemistry problem I	
ODEs	BCs
$y_1' = y_3 - 100.0\ y_1 y_2$	$y_1(0) - 1.0 = 0.0$
$y_2' = y_3 - 100.0\ y_1 y_2 - y_2^2 + 2y_4$	$y_2(0) - 1.0 = 0.0$
$y_3' = -y_3 + 100.0\ y_1 y_2$	$y_3(0) - 1.0 = 0.0$
$y_4' = y_2^2 - y_4$	$y_4(0) - 1.0 = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0$	

A.7

Problem A.7 is same as Problem A.6 but coupled with different boundary conditions.

Chemistry problem II	
ODEs	BCs
$y_1' = y_3 - 100.0\ y_1 y_2$	$y_1(0) - 1.0 = 0.0$
$y_2' = y_3 - 100.0\ y_1 y_2 - y_2^2 + 2y_4$	$y_2(0) - 1.0 = 0.0$
$y_3' = -y_3 + 100.0\ y_1 y_2$	$y_3(0) - 1.0 = 0.0$
$y_4' = y_2^2 - y_4$	$y_4(0) - 1.0 + y_4(2) - 0.619 = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0$	

A.8 Hydrodynamic problem

Problem A.8 can be found in [Hol64].

Hydrodynamic problem I : Swirling flow	
ODEs	BCs
$y_1' = y_2$	$y_1(0) = 0.0$
$y_2' = y_3$	$y_2(0) + y_2(5) = 0.0$
$y_3' = \frac{(t-3)}{2} y_1 y_3 - t y_2^2 + 1 - y_4^2 + s y_2$	$y_3(0) + 0.978297 = 0.0$
$y_4' = y_5$	$y_4(0) + y_4(5) - 1.0 = 0.0$
$y_5' = \frac{(t-3)}{2} y_1 y_5 - (t-1) y_2 y_4 + s(y_4 - 1)$	$y_5(0) - 0.646786 = 0.0$
where $t = -0.1,\ s = 0.2$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0,\ y_5(x) = 0.0$	

A.9

Problem A.9 is a modification of Problem A.8 and can be found in [KM93].

Hydrodynamic problem II : Swirling flow	
ODEs	BCs
$y_1' = y_2$	$y_1(0) = 0.0$
$y_2' = y_3$	$y_2(0) + y_2(1) = 0.0$
$y_3' = \frac{(t-3)}{2} y_1 y_3 - t y_2^2 + 1 - y_4^2 + s y_2$	$y_3(0) = 0.0$
$y_4' = y_5$	$y_4(0) + y_4(1) - 1.0 = 0.0$
$y_5' = \frac{(t-3)}{2} y_1 y_5 - (t-1) y_2 y_4 + s(y_4 + 1)$	$y_5(0) = 0.0$
where $t = s = 0.2$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0,\ y_5(x) = 0.0$	

A.10 Fluid injection problem

Problem A.10 can be found in [AMR88].

Fluid injection problem I : Flow in a channel	
ODEs	BCs
$y_1' = y_2$	$y_1(0) = 0.0$
$y_2' = y_3$	$y_2(0) = 0.0$
$y_3' = R(y_2^2 - y_1 y_3 - y_4)$	$y_1(1) - 1.0 = 0.0$
$y_4' = 0.0$	$y_2(1) = 0.0$
$y_5' = y_6$	$y_5(0) = 0.0$
$y_6' = -R y_1 y_6 - 1$	$y_5(1) = 0.0$
$y_7' = y_8$	$y_7(0) = 0.0$
$y_8' = -0.7 y_1 y_8$	$y_7(1) - 1.0 = 0.0$
where $R = 10.0$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0,$ $y_5(x) = 0.0,\ y_6(x) = 0.0,\ y_7(x) = 0.0,\ y_8(x) = 0.0$	

A.11

Problem A.11 can be found in [AMR88]. This is a difficult problem because there is a boundary layer near $x = 0$.

Fluid injection problem II : Flow in a channel	
ODEs	BCs
$y_1' = y_2$	$y_1(0) = 0.0$
$y_2' = y_3$	$y_2(0) = 0.0$
$y_3' = y_4$	$y_1(1) - 1.0 = 0.0$
$y_4' = R(y_2 y_3 - y_1 y_4)$	$y_2(1) = 0.0$
where $R = 10000.0$	
Initial Estimate	
$y_1(x) = -2.0x^3 + 3.0x^2,\ y_2(x) = -6.0x^2 + 6.0x,$ $y_3(x) = -12.0x + 6.0,\ y_4(x) = -12.0$	

A.12

Problem A.12 is just two independent specifications of problem A.10. This artificial problem is used to test the speedup efficiency of the new parallel multiple shooting codes for large value of n.

Fluid injection problem III : Flow in a channel	
ODEs	BCs
$y_1' = y_2$	$y_1(0) = 0.0$
$y_2' = y_3$	$y_2(0) = 0.0$
$y_3' = R(y_2^2 - y_1 y_3 - y_4)$	$y_1(1) - 1.0 = 0.0$
$y_4' = 0.0$	$y_2(1) = 0.0$
$y_5' = y_6$	$y_5(0) = 0.0$
$y_6' = -Ry_1 y_6 - 1$	$y_5(1) = 0.0$
$y_7' = y_8$	$y_7(0) = 0.0$
$y_8' = -0.7 y_1 y_8$	$y_7(1) - 1.0 = 0.0$
$y_9' = y_{10}$	$y_9(0) = 0.0$
$y_{10}' = y_{11}$	$y_{10}(0) = 0.0$
$y_{11}' = R(y_{10}^2 - y_9 y_{11} - y_{12})$	$y_9(1) - 1.0 = 0.0$
$y_{12}' = 0.0$	$y_{10}(1) = 0.0$
$y_{13}' = y_{14}$	$y_13(0) = 0.0$
$y_{14}' = -Ry_9 y_{14} - 1$	$y_{13}(1) = 0.0$
$y_{15}' = y_{16}$	$y_{15}(0) = 0.0$
$y_{16}' = -0.7 y_9 y_{16}$	$y_{15}(1) - 1.0 = 0.0$
where $R = 10.0$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0,$ $y_5(x) = 0.0,\ y_6(x) = 0.0,\ y_7(x) = 0.0,\ y_8(x) = 0.0$ $y_9(x) = 0.0,\ y_{10}(x) = 0.0,\ y_{11}(x) = 0.0,\ y_{12}(x) = 0.0,$ $y_{13}(x) = 0.0,\ y_{14}(x) = 0.0,\ y_{15}(x) = 0.0,\ y_{16}(x) = 0.0$	

A.13 Artificial test problem

The artificial BVODE problem possesses both rapidly decaying and growing solution components and can be found in [DB82]. This problem is considered to be difficult for $\lambda = 20$ and will become more difficult as λ increases.

Artifical test problem	
ODEs	BCs
$y_1' = y_2$ $y_2' = \lambda^2(y_1 + \cos^2(\pi x)) + 2\pi^2\cos(2\pi x)$ where $\lambda = 20, 50, 80$	$y_1(0) = 0.0$ $y_1(1) = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0$	
True Solution	
$y(x) = \frac{e^{-\lambda}}{1+e^{-\lambda}}e^{\lambda x} + \frac{1}{1+e^{-\lambda}}e^{-\lambda x} - \cos^2(\pi x)$	

A.14

Problem A.14 posses one decaying and one growing solution components and can be found in [AC91, JP92]. This problem will become more difficult as λ increases.

Standard test problem I	
ODEs	BCs
$y_1' = a_{11}y_1 + a_{12}y_2 + (1.0 - a_{11} - a_{12})e^x$ $y_2' = a_{21}y_1 + a_{22}y_2 + (1.0 - a_{21} - a_{22})e^x$ where $a_{11} = -\lambda\cos(2\omega x),\ a_{12} = \omega + \lambda\sin(2\omega x)$ $a_{21} = -\omega + \lambda\sin(2\omega x),\ a_{22} = \lambda 2\cos(2\omega x)$ $\lambda = 80,\ \omega = 1.0$	$y_1(0) = 0.0$ $y_1(1) - e = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0$	
True Solution	
$y_1(x) = e^x,\ y_2(x) = e^x$	

A.15

Problem A.15 possesses both decaying and growing solution components and can be found in [Wri94, JP92].

Standard test problem II	
ODEs	BCs
$y_1' = a_{11}y_1 + a_{12}y_2 + a_{13}y_3 + (1.0 - a_{11} - a_{12} - a_{13})e^x$ $y_2' = a_{21}y_1 + a_{22}y_2 + a_{23}y_3 + (1.0 - a_{21} - a_{22} - a_{23})e^x$ $y_3' = a_{31}y_1 + a_{32}y_2 + a_{33}y_3 + (1.0 - a_{31} - a_{32} - a_{33})e^x$ where $a_{11} = 1.0 - 19.0 * \cos(2x),\ a_{12} = 0.0$ $a_{13} = 1.0 + 19.0 * \sin(2x),\ a_{21} = 0.0$ $a_{22} = 19.0,\ a_{23} = 0.0$ $a_{31} = -1.0 + 19.0 * \sin(2x),\ a_{32} = 0.0$ $a_{33} = 1 + 19\cos(2x)$	$y_1(0) - 1.0 = 0.0$ $y_2(\pi) - e^{\pi} = 0.0$ $y_1(\pi) + 3.0 * y_3(\pi) - 4.0 * e^{\pi} = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0$	
True Solution	
$y_1(x) = e^x,\ y_2(x) = e^x,\ y_3(x) = e^x$	

A.16

Problem A.16 is same as Problem A.15 but coupled with different boundary conditions.

Standard test problem III	
ODEs	BCs
$y_1' = a_{11}y_1 + a_{12}y_2 + a_{13}y_3 + (1.0 - a_{11} - a_{12} - a_{13})e^x$ $y_2' = a_{21}y_1 + a_{22}y_2 + a_{23}y_3 + (1.0 - a_{21} - a_{22} - a_{23})e^x$ $y_3' = a_{31}y_1 + a_{32}y_2 + a_{33}y_3 + (1.0 - a_{31} - a_{32} - a_{33})e^x$ where $a_{11} = 1.0 - 19.0 * \cos(2x),\ a_{12} = 0.0$ $a_{13} = 1.0 + 19.0 * \sin(2x),\ a_{21} = 0.0$ $a_{22} = 19.0,\ a_{23} = 0.0$ $a_{31} = -1.0 + 19.0 * \sin(2x),\ a_{32} = 0.0$ $a_{33} = 1 + 19\cos(2x)$	$y_1(0) - 1.0 = 0.0$ $y_2(0) + y_2(\pi) - 1.0 - e^{\pi} = 0.0$ $y_3(0) + y_3(\pi) - 1.0 - e^{\pi} = 0.0$
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0$	
True Solution	
$y_1(x) = e^x,\ y_2(x) = e^x,\ y_3(x) = e^x$	

A.17

Problem A.17 possesses both decaying and growing solution components and can be found in [Wri94, JP92].

Standard test problem IV	
ODEs	BCs
$y_1' = a_{11}y_1 + a_{13}y_3 + (1.0 - a_{11} - a_{13})e^x$	$y_1(0) - 1.0 = 0.0$
$y_2' = a_{22}y_2 + a_{24}y_4 + (1.0 - a_{22} - a_{24})e^x$	$y_2(0) + 4.0 * y_5(0) - 5.0 = 0.0$
$y_3' = a_{31}y_1 + a_{33}y_3 + (1.0 - a_{31} - a_{33})e^x$	$y_1(1) - e = 0.0$
$y_4' = a_{42}y_2 + a_{44}y_4 + (1.0 - a_{42} - a_{44})e^x$	$y_4(1) - y_3(1) = 0.0$
$y_5' = a_{55}y_5 + (1.0 - a_{55})e^x$	$5.0 * y_5(1) - 4.0 * y_2(1) - e = 0.0$
where $a_{11} = -\lambda_1 \cos(2\omega_1 x)$	
$a_{13} = \omega_1 + \lambda_1 \sin(2\omega_1 x)$	
$a_{22} = -\lambda_2 \cos(2\omega_2 x)$	
$a_{24} = \omega_2 + \lambda_2 \sin(2\omega_2 x)$	
$a_{13} = -\omega_1 + \lambda_1 \sin(2\omega_1 x)$	
$a_{33} = \lambda_1 \cos(2\omega_1 x)$	
$a_{42} = -\omega_2 + \lambda_2 \sin(2\omega_2 x)$	
$a_{44} = \lambda_2 \cos(2\omega_2 x)$	
$a_{55} = \lambda_3$	
$\lambda_1 = 40.0,\ \lambda_2 = 40.0,\ \lambda_3 = 10.0$	
$\omega_1 = 1.0,\ \omega_2 = 25.0$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0,\ y_4(x) = 0.0,\ y_5(x) = 0.0$	
True Solution	
$y_1(x) = e^x,\ y_2(x) = e^x,\ y_3(x) = e^x,\ y_4(x) = e^x,\ y_5(x) = e^x$	

A.18

Problem A.18 possesses both decaying and growing solution components and can be found in [JP92].

Standard test problem V	
ODEs	BCs
$y_1' = a_{11}y_1 + a_{12}y_2 + (1.0 - 10a_{11} - 10a_{12})e^{\frac{x}{10}}$	$y_1(0) - 10.0 = 0.0$
$y_2' = a_{21}y_1 + a_{22}y_2 + (1.0 - 10a_{21} - 10a_{22})e^{\frac{x}{10}}$	$y_1(10) - 10.0 * e = 0.0$
where $a_{11} = a(10b - a) - 2b^2$	
$a_{12} = a(b + 10a) - 2ab$	
$a_{21} = -b(10b - a) - 2ab$	
$a_{22} = -b(b + 10a) - 2a^2$	
$a = \cos(2\pi x),\ b = \sin(2\pi x)$	
Initial Estimate	
$y_1(x) = 0.0,\ y_2(x) = 0.0,\ y_3(x) = 0.0$	
True Solution	
$y_1(x) = 10e^{\frac{x}{10}},\ y_2(x) = 10e^{\frac{x}{10}}$	

B BVODE Results

B.1 Serial results

Notation		
m	:	Number of mesh points
No. iter	:	Number of Newton iterations
Max. error	:	The maximum component of the residual function, i.e., $\parallel \Phi \parallel_\infty = \max_{0 \leq i \leq m-1} \{ \parallel \Phi_i \parallel_\infty \}$
t	:	Processing time measured in Second
S_T	:	% of savings in time relative to j-bvpms $= \frac{(t_1 - t_2)}{t_1} \times 100\%$ where t_1 and t_2 are processing time required by j-bvpms and bn-bvpms respectively
$*$	:	No record because the related problem couldn't be solved

m	j-bvpms	bn-bvpms	S_T
1-8	*	*	*
16	6 0.492741 0.10837D-05	6 0.462837 0.10837D-05	6.51
32	6 0.696969 0.69639D-05	6 0.649334 0.69639D-05	6.83
64	6 1.37058 0.11523D-08	6 1.22796 0.11523D-08	10.47
128	6 1.72776 0.43756D-09	6 1.44087 0.43756D-09	16.60
256	5 2.84554 0.37922D-05	5 2.54196 0.37922D-05	10.67
512	5 5.68994 0.18851D-05	5 5.00204 0.18851D-05	12.09
1024	5 11.3706 0.94182D-06	5 10.0588 0.94182D-06	11.54

Table 2: No. iter, t, and Max. error for Problem A.1 (Problem size $n = 4$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	bn-bvpms	S_T
2	11 109.458 0.70726D-06	11 107.489 0.70726D-06	1.8
3	8 0.797084 0.46445D-05	8 0.707624 0.46445D-05	11.23
4	*	*	*
5	7 0.477734 0.48597D-09	7 0.406553 0.48597D-09	14.90
6	6 0.439622 0.34119D-05	6 0.393257 0.34119D-05	10.55
7	6 0.448174 0.33638D-06	6 0.392062 0.33638D-06	12.52
8	6 0.521294 0.53997D-07	6 0.440077 0.53997D-07	15.59
16	6 0.847643 0.24272D-09	6 0.726644 0.24272D-09	14.27
32	5 1.01040 0.26080D-05	5 0.845366 0.26080D-05	16.33
64	5 2.18829 0.12256D-06	5 1.72445 0.12256D-06	21.20
128	5 2.48923 0.61418D-06	5 1.96749 0.61418D-06	20.96
256	5 5.13722 0.30984D-06	5 3.92040 0.30984D-06	23.69
512	5 10.1276 0.15590D-06	5 7.71573 0.15590D-06	23.81
1024	5 20.7086 0.78234D-07	5 15.5597 0.78234D-07	24.86

Table 3: No. iter, t, and Max. error for Problem A.2 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	3 0.195871 0.44409D-15	3 0.196581 0.44409D-15	3 0.190287 0.44409D-15	3 0.191384 0.44409D-15
3	4 0.319636 0.11102D-15	4 0.327433 0.22204D-15	4 0.315289 0.22204D-15	4 0.315454 0.22204D-15
4	4 0.387151 0.56797D-07	4 0.398760 0.56797D-07	4 0.352857 0.56797D-07	4 0.355565 0.56797D-07
5	4 0.411698 0.359589D-05	4 0.429913 0.359589D-05	4 0.386375 0.359589D-05	4 0.387736 0.359589D-05
6	5 0.529570 0.19044D-06	5 0.594330 0.19044D-06	5 0.508260 0.19044D-06	5 0.509162 0.19044D-06
7	5 0.581363 0.13112D-06	5 0.618312 0.13112D-06	5 0.539666 0.13112D-06	5 0.542677 0.13112D-06
8	5 0.616908 0.68239D-07	5 0.642857 0.68239D-07	5 0.590944 0.68239D-07	5 0.591726 0.68239D-07
16	5 1.15158 0.19228D-06	5 1.21026 0.19228D-06	5 0.885189 0.19228D-06	5 0.917009 0.19228D-06
32	5 1.86461 0.95253D-06	5 1.93800 0.95253D-06	5 1.38981 0.95253D-06	5 1.40675 0.95253D-06
64	5 2.29564 0.22827D-05	5 2.99726 0.22827D-05	5 2.18248 0.22827D-05	5 2.20908 0.22827D-05
128	6 3.99094 0.58375D-07	6 6.87576 0.58375D-07	6 3.70943 0.58375D-07	6 4.95855 0.58375D-07
256	6 6.29283 0.49170D-05	6 15.8975 0.49170D-05	6 5.26134 0.49170D-05	6 5.62506 0.49170D-05

Table 4: No. iter, t, and Max. error for Problem A.6 (Problem size $n = 4$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	4 0.289430 0.20279D-05	4 0.296617 0.20279D-05	4 0.281707 0.20279D-05	4 0.284497 0.20279D-05
3	5 0.420727 0.18218D-05	5 0.424213 0.18218D-05	5 0.412420 0.18218D-05	5 0.418460 0.18218D-05
4	5 0.477843 0.15664D-06	5 0.491527 0.15664D-06	5 0.462572 0.15664D-06	5 0.467755 0.15664D-06
5	5 0.522597 0.22246D-07	5 0.540261 0.22246D-07	5 0.498952 0.22246D-07	5 0.499886 0.22246D-07
6	5 0.597016 0.10599D-06	5 0.603676 0.10599D-06	5 0.545698 0.10599D-06	5 0.552628 0.10599D-06
7	5 0.623935 0.25720D-06	5 0.652613 0.25720D-06	5 0.583107 0.25720D-06	5 0.593675 0.25720D-06
8	5 0.663434 0.62601D-06	5 0.698517 0.62601D-06	5 0.639877 0.62601D-06	5 0.640798 0.62601D-06
16	6 1.13489 0.16229D-07	6 1.32093 0.16229D-07	6 1.05180 0.16229D-07	6 1.10025 0.16229D-07
32	6 1.76525 0.31361D-06	6 3.00080 0.31361D-06	6 1.70593 0.31361D-06	6 1.70891 0.31361D-06
64	6 2.78540 0.44707D-07	6 3.74956 0.44707D-07	6 2.54379 0.44707D-07	6 2.58780 0.44707D-07
128	6 4.32260 0.14898D-06	6 7.49565 0.14898D-06	6 4.02469 0.14898D-06	6 4.25228 0.14898D-06
256	7 7.38524 0.25633D-09	7 19.0623 0.25633D-09	7 6.14436 0.25633D-09	7 6.59377 0.25633D-09

Table 5: No. iter, t, and Max. error for Problem A.7 (Problem size $n = 4$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	7 1.06990 0.26674D-09	7 1.13835 0.26673D-09	7 1.06506 0.26673D-09	7 1.37538 0.26674D-09
3	6 1.02278 0.18222D-09	6 1.10067 0.18222D-09	6 1.00662 0.18223D-09	6 1.04109 0.18223D-09
4	5 0.896596 0.99489D-06	5 1.00256 0.99489D-06	5 0.869251 0.99489D-06	5 0.903150 0.99489D-06
5	5 1.02218 0.52147D-07	5 1.15284 0.52147D-07	5 1.00527 0.52147D-07	5 1.03872 0.52147D-07
6	5 1.13767 0.48449D-08	5 1.30325 0.48449D-08	5 1.08706 0.48449D-08	5 1.13349 0.48449D-08
7	5 1.25537 0.10923D-08	5 1.52214 0.10923D-08	5 1.17140 0.10923D-08	5 1.22615 0.10923D-08
8	5 1.29798 0.78357D-09	5 1.54444 0.78357D-09	5 1.19292 0.78358D-09	5 1.31765 0.78357D-09
16	5 2.52113 0.81040D-11	5 2.54005 0.81042D-11	5 1.80773 0.81040D-11	5 2.01644 0.81042D-11
32	5 3.30935 0.47661D-11	5 4.25568 0.47656D-11	5 2.61861 0.47657D-11	5 2.91318 0.47656D-11
64	5 7.08016 0.56508D-11	5 10.37003 0.56507D-11	5 5.46785 0.56507D-11	5 5.95798 0.56509D-11
128	5 8.23247 0.35225D-11	5 18.7189 0.35225D-11	5 5.98271 0.35226D-11	5 6.98286 0.35226D-11
256	4 12.2378 0.42079D-05	4 42.8391 0.42079D-05	4 9.70212 0.42079D-05	4 11.2046 0.42079D-05

Table 6: No. iter, t, and Max. error for Problem A.10 (Problem size $n = 8$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	5 0.158123 0.49129D-07	4 0.141135 0.46138D-07	4 0.124592 0.46138D-07	4 0.134407 0.46138D-07
3	6 0.233928 0.37321D-08	4 0.182810 0.34222D-07	4 0.152618 0.34222D-07	4 0.16741 0.34222D-07
4	5 0.224794 0.12066D-05	4 0.214988 0.33844D-07	4 0.171877 0.33844D-07	4 0.184075 0.33844D-07
5	5 0.253239 0.90830D-07	4 0.250405 0.57204D-07	4 0.187764 0.57204D-07	4 0.208247 0.57204D-07
6	5 0.309837 0.31649D-06	4 0.313701 0.62212D-07	4 0.234381 0.62212D-07	4 0.251036 0.62212D-07
7	5 0.311473 0.34124D-06	4 0.323288 0.60929D-07	4 0.226712 0.60929D-07	4 0.250302 0.60929D-07
8	5 0.356518 0.43671D-06	4 0.369553 0.57616D-07	4 0.262584 0.57616D-07	4 0.285939 0.57616D-07
16	5 0.602482 0.20030D-05	4 0.820354 0.33952D-07	4 0.430273 0.33952D-07	4 0.469159 0.33952D-07
32	5 0.907889 0.27696D-06	4 1.13544 0.17597D-07	4 0.628569 0.17597D-07	4 0.714487 0.17597D-07
64	6 2.24820 0.44197D-08	4 2.48904 0.88894D-08	4 1.23082 0.88894D-08	4 1.43996 0.88894D-08
128	6 2.82560 0.44677D-08	4 5.24178 0.44659D-08	4 1.44242 0.44659D-08	4 1.74250 0.44659D-08
256	6 6.01263 0.51258D-08	4 14.6692 0.22374D-08	4 3.31252 0.22374D-08	4 3.75682 0.22374D-08

Table 7: No. iter, t, and Max. error for Problem A.9 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	*	*	*	*
3	*	9 0.758699 0.28414D-07	9 0.662892 0.13941D-10	9 0.696631 0.13906D-10
4	*	*	*	*
5	9 0.457389 0.51567D-07	7 0.487893 0.25613D-08	6 0.301960 0.54526D-06	6 0.322278 0.54526D-06
6	7 0.427671 0.28122D-05	6 0.462673 0.22511D-05	6 0.348511 0.13429D-07	6 0.373425 0.13429D-07
7	7 0.442262 0.11018D-05	6 0.480177 0.86388D-06	6 0.345478 0.81923D-09	6 0.371049 0.81924D-09
8	7 0.678619 0.56315D-06	6 0.744767 0.42327D-06	6 0.505714 0.13425D-09	6 0.530247 0.13424D-09
16	8 0.954879 0.91739D-08	6 0.976473 0.47118D-07	5 0.543250 0.45928D-06	5 0.595769 0.45928D-06
32	8 1.47341 0.10467D-06	6 1.69814 0.20303D-07	5 0.784624 0.68833D-07	5 0.892908 0.68833D-07
64	8 2.88210 0.13948D-06	6 3.70939 0.14336D-07	5 1.60771 0.17677D-07	5 1.78331 0.17677D-07
128	8 3.69986 0.10956D-06	6 7.94730 0.12257D-07	5 1.81819 0.62838D-08	5 2.21920 0.62838D-08
256	8 7.97856 0.17854D-06	6 21.8718 0.11379D-07	5 4.25970 0.25997D-08	5 4.44801 0.25997D-08

Table 8: No. iter, t, and Max. error for Problem A.3 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	*	*	*	*
3	*	*	*	*
4	*	*	*	*
5	*	7 0.456093 0.18076D-09	6 0.306288 0.51471D-07	6 0.327653 0.51471D-07
6	*	6 0.482014 0.37937D-05	6 0.354377 0.10791D-07	6 0.377589 0.10791D-07
7	*	6 0.480502 0.32809D-05	6 0.344189 0.75315D-09	6 0.371166 0.75315D-09
8	8 0.560708 0.53687D-06	6 0.542072 0.19408D-05	6 0.394682 0.13357D-09	6 0.426377 0.13357D-09
16	10 1.18628 0.67215D-06	10 1.68367 0.23746D-05	5 0.537267 0.44757D-06	5 0.583894 0.44757D-06
32	13 2.30914 0.13552D-05	*	5 0.761031 0.64604D-07	5 0.882393 0.64604D-07
64	13 4.65669 0.29115D-05	*	5 1.53261 0.17516D-07	5 1.78089 0.17516D-07
128	*	*	5 1.75243 0.62475D-08	5 2.23833 0.62475D-08
256	*	*	5 3.42762 0.25895D-08	5 4.43307 0.25895D-08

Table 9: No. iter, t, and Max. error for Problem A.4 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2	*	*	*	*
3	*	*	*	*
4	*	*	*	*
5	*	7 0.466509 0.11591D-05	6 0.308688 0.32877D-06	6 0.326555 0.32877D-06
6	19 1.16696 0.18831D-06	7 0.553737 0.11729D-06	6 0.349635 0.46980D-08	6 0.374409 0.46980D-08
7	13 0.810028 0.45773D-05	7 0.570814 0.26783D-07	6 0.345654 0.36255D-09	6 0.372139 0.36255D-09
8	18 1.30150 0.96200D-07	7 0.648592 0.15557D-07	6 0.394491 0.81355D-10	6 0.425793 0.81358D-10
16	11 1.33810 0.13011D-05	6 0.997471 0.24528D-05	5 0.533593 0.41984D-06	5 0.587113 0.41984D-06
32	12 2.21504 0.80541D-07	6 1.83062 0.23204D-05	5 0.764077 0.65584D-07	5 0.880867 0.65584D-07
64	12 4.46881 0.15165D-06	6 3.83152 0.26318D-05	5 1.53615 0.18645D-07	5 1.77062 0.18645D-07
128	12 5.86922 0.23554D-06	6 8.04125 0.28880D-05	5 1.75458 0.68165D-08	5 2.19615 0.68165D-08
256	12 12.1691 0.31158D-06	6 22.2930 0.30437D-05	5 3.40527 0.28635D-08	5 4.46803 0.28635D-08

Table 10: No. iter, t, and Max. error for Problem A.5 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-8	*	*	*	*
16	*	*	*	*
32	*	*	*	*
64	29 12.1758 0.309740D-07	22 15.7790 0.11219D-05	*	22 9.08483 0.11219D-05
128	13 6.49497 0.65990D-07	12 15.6481 0.42947D-05	14 6.31821 0.37265D-08	12 5.32535 0.42947D-05
256	11 10.38948 0.61936D-10	9 33.1834 0.26554D-06	9 6.36359 0.96694D-07	9 7.39311 0.25654D-06
512	14 27.2394 0.90221D-09	10 79.5512 0.15780D-10	23 31.8552 0.34340D-05	10 16.2120 0.15780D-10
1024	26 99.1892 0.35458D-06	7 148.7778 0.36610D-05	7 20.1384 0.40392D-05	7 23.3874 0.36610D-05

Table 11: No. iter, t, and Max. error for Problem A.8 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
8	3 0.182204 0.78614D-07	3 0.190208 0.78613D-07	4 0.298345 0.15484D-06	3 0.232771 0.78611D-07
16	3 0.218874 0.96616D-09	3 0.220792 0.96620D-09	4 0.312703 0.38030D-09	3 0.254946 0.96616D-09
32	2 0.189358 0.37008D-05	2 0.207933 0.37008D-05	4 0.371532 0.13411D-09	2 0.217732 0.37014D-05
64	2 0.213366 0.21931D-05	2 0.302019 0.21931D-05	4 0.430407 0.82419D-11	2 0.262562 0.21927D-05
128	2 0.282088 0.11416D-05	2 0.531271 0.11416D-05	4 0.509779 0.14615D-10	2 0.329537 0.11399D-05
256	2 0.373443 0.14211D-13	2 1.24602 0.71054D-14	4 0.821608 0.19130D-10	2 0.465437 0.10424D-08

Table 12: No. iter, t, and Max. error for Problem A.13 (Problem size $n = 2$, $\lambda = 50$, Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
8	3 0.284110 0.14542D-05	3 0.292060 0.14542D-05	*	4 0.454533 0.49780D-06
16	3 0.323088 0.59451D-08	3 0.338638 0.59445D-08	*	4 0.478726 0.86031D-09
32	3 0.366710 0.68894D-10	3 0.391620 0.68853D-10	*	4 0.578172 0.18091D-10
64	3 0.391482 0.40390D-12	3 0.530671 0.20373D-12	*	4 0.653508 0.11084D-11
128	2 0.383185 0.37709D-06	2 0.656743 0.37709D-06	*	3 0.618028 0.16731D-05
256	2 0.394888 0.14211D-13	2 1.30133 0.14211D-13	*	3 0.701883 0.60012D-07

Table 13: No. iter, t, and Max. error for Problem A.13 (Problem size $n = 2$, $\lambda = 80$, Error tolerance $= 10^{-5}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-4	*	*	*	*
8	3 2.13483 0.23886D-10	3 2.34133 0.23869D-10	*	4 3.21093 0.18169D-10
16	3 2.19770 0.48628D-13	3 2.45512 0.51736D-13	*	4 3.10774 0.15543D-12
32	3 2.31044 0.44409D-14	3 2.68265 0.44409D-14	*	4 3.14526 0.75939D-13
64	3 2.16144 0.22204D-14	3 2.73048 0.17764D-14	*	4 3.40082 0.66169D-13
128	3 2.48243 0.13323D-14	3 3.15616 0.88818D-15	*	4 3.83151 0.10436D-12
256	3 2.93891 0.17764D-14	3 4.53519 0.88818D-15	*	4 4.53544 0.24425D-13

Table 14: No. iter, t, and Max. error for Problem A.14 (Problem size $n = 2$, $\lambda = 80$, Error tolerance $= 10^{-10}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-4	*	*	*	*
8	3 2.83170 0.14193D-10	3 3.15817 0.54206D-11	*	3 2.87817 0.11381D-10
16	3 3.01142 0.28777D-12	3 3.72272 0.28777D-12	*	3 3.35178 0.14921D-12
32	3 3.17212 0.31974D-13	3 3.75997 0.42633D-13	*	3 3.61879 0.99476D-13
64	3 3.59665 0.14211D-13	3 4.10896 0.28422D-13	*	3 4.07925 0.28422D-13
128	3 3.97235 0.28422D-13	3 5.27595 0.28422D-13	*	3 5.02946 0.54001D-12
256	3 4.54788 0.14211D-13	3 8.02360 0.14211D-13	*	3 5.38928 0.42633D-13

Table 15: No. iter, t, and Max. error for Problem A.15 (Problem size $n = 3$ and Error tolerance $= 10^{-10}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-4	*	*	*	*
8	3 2.85005 0.14751D-10	3 3.10012 0.13788D-10	*	3 2.86277 0.13124D-10
16	3 3.39485 0.18119D-12	3 3.67921 0.15277D-12	*	3 3.31806 0.32330D-12
32	3 3.20766 0.23093D-13	3 3.69764 0.49738D-13	*	3 3.60449 0.85265D-13
64	3 3.57184 0.14211D-13	3 4.05900 0.10658D-13	*	3 3.93047 0.35527D-13
128	3 4.02601 0.10658D-13	3 5.19049 0.10658D-13	*	3 4.63454 0.21316D-13
256	3 4.52088 0.71054D-14	3 7.88079 0.71054D-14	*	3 5.29845 0.17764D-13

Table 16: No. iter, t, and Max. error for Problem A.16 (Problem size $n = 3$ and Error tolerance $= 10^{-10}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-4	*	*	*	*
8	13 12.5105 0.13580D-10	21 23.694 0.20056D-10	21 20.8244 0.20164D-10	21 20.9328 0.20056D-10
16	13 12.8935 0.13580D-10	13 14.3477 0.13580D-10	13 12.3490 0.20811D-10	13 13.3240 0.13577D-10
32	11 12.3480 0.48801D-11	11 13.6011 0.48805D-11	11 12.2456 0.12783D-10	11 12.3980 0.48801D-11
64	10 12.6940 0.70104D-11	10 15.0269 0.70108D-11	10 12.5538 0.20554D-10	10 12.7896 0.70108D-11
128	9 14.7829 0.39545D-10	9 22.3665 0.39545D-10	10 16.4982 0.12916D-10	9 15.6088 0.39545D-10
256	9 20.1096 0.16711D-10	9 45.6365 0.16711D-10	9 18.6198 0.35342D-10	9 19.6828 0.16711D-10

Table 17: No. iter, t, and Max. error for Problem A.17 (Problem size $n = 5$, $\lambda_1 = 40, \lambda_2 = 40, \lambda_3 = 10, \omega_1 = 1, \omega_2 = 25$, Error tolerance $= 10^{-10}$)

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-4	*	*	*	*
8	*	*	14 10.56824 0.32742D-10	20 14.7484 0.32742D-10
16	4 2.90523 0.18872D-10	4 3.3779 0.25921D-10	7 5.43750 0.12733D-10	4 3.42627 0.15902D-10
32	4 3.07953 0.25466D-10	4 3.60359 0.10459D-10	4 3.03436 0.36380D-10	4 3.55602 0.17963D-10
64	4 3.29351 0.72760D-11	4 4.14227 0.15248D-10	4 3.22950 0.10914D-10	4 3.94753 0.14552D-10
128	4 3.91839 0.90949D-11	4 4.80750 0.18190D-10	4 3.90754 0.97948D-11	4 4.04235 0.10914D-10
256	4 4.71787 0.11823D-10	4 7.47436 0.72760D-11	4 4.70851 0.72760D-11	4 4.57321 0.13642D-10

Table 18: No. iter, t, and Max. error for Problem A.18 (Problem size $n = 2$ and Error tolerance = 10^{-10})

m	j-bvpms	gir-j-bvpms	bn-bvpms	gir-bn-bvpms
2-12	*	*	*	*
13	92 215.804 0.25060D-07	135 328.100 0.22344D-06	65 149.333 0.17124D-05	143 192.855 0.15668D-05
14	26 63.6855 0.15300D-06	26 65.8353 0.25204D-06	26 60.7153 0.14812D-05	26 63.2740 0.27858D-05
15	40 96.7772 0.14826D-06	39 96.2278 0.29347D-05	38 89.8636 0.87836D-07	40 95.9305 0.31704D-07
16	21 57.0518 0.11289D-06	21 58.5994 0.12021D-06	21 52.9870 0.11290D-06	21 53.9987 0.18632D-06
17	17 41.4001 0.14708D-05	17 42.9064 0.14606D-05	17 41.0463 0.14816D-05	17 41.3131 0.14563D-05
18	13 31.9113 0.60116D-06	13 32.6688 0.60112D-06	13 31.5647 0.60069D-06	13 31.9249 0.61380D-06
19	13 31.8833 0.67760D-06	13 32.3654 0.67760D-06	13 31.8124 0.67760D-06	13 32.0570 0.67760D-06
20	18 43.7704 0.12980D-05	15 37.8062 0.91218D-06	15 37.1206 0.21345D-06	17 42.0828 0.54335D-06
21	12 29.4920 0.24954D-06	12 30.3778 0.24910D-06	12 29.3432 0.24950D-06	12 30.0665 0.25044D-06
22	17 42.1894 0.27256D-05	17 48.3789 0.26923D-05	17 40.2834 0.27105D-05	17 42.7169 0.26980D-05

Table 19: No. iter, t, and Max. error for Problem A.11 (Problem size $n = 4$ and Error tolerance $= 10^{-5}$)

B.2 Parallel results

Notation		
m	:	Number of mesh points
t_1, t_2	:	Computational time measured in Second
$S_T(t_1, t_2)$	:	% of savings in time relative to t_1
	:	$= \frac{(t_1 - t_2)}{t_1} \times 100\%$
$S_p(t_1, t_2)$	:	Speedup relative to j-bvpms using p processors

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p = 1$	$p = 8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	8.58445D-02	9.88550D-03	1.51800D-02	65.12	4	4.00	4	5.66	5.50	1
	8.22665D-02	9.88150D-03	1.51015D-02	65.43	4	4.00	4	5.45		2
	8.21510D-02	9.83700D-03	1.51795D-02	64.80	4	4.00	4	5.41		3
	8.22445D-02	9.88050D-03	1.52070D-02	64.97	4	4.00	4	5.41		4
	8.19330D-02	9.81150D-03	1.51010D-02	64.97	4	4.00	4	5.43		5
	8.20780D-02	9.89100D-03	1.46030D-02	67.73	4	4.00	4	5.62		6
Overall (IVPs)	0.505714 (0.470122)	0.122168	9.41265D-02					5.37 (4.14)		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	0.107461	7.32600D-03	1.83495D-02	39.92	3	3.00	3	5.85	5.83	1
	0.107682	7.19450D-03	1.80300D-02	39.90	3	3.00	3	5.97		2
	0.106396	7.51450D-03	1.87335D-02	40.11	3	3.00	3	5.68		3
	0.106657	7.60750D-03	1.86440D-02	40.80	3	3.00	3	5.72		4
	0.106622	7.51750D-03	1.80480D-02	41.65	3	3.00	3	5.91		5
Overall (IVPs)	0.543250 (0.493944)	0.163126	0.118917					4.57 (3.33)		
m	$p = 1$	$p = 32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	0.156007	5.50200D-03	1.97005D-02	27.93	2	2.00	2	7.92	7.55	1
	0.154930	5.48850D-03	2.03630D-02	26.95	2	2.00	2	7.61		2
	0.154982	5.46000D-03	2.17520D-02	25.10	2	2.00	2	7.21		3
	0.155151	5.52200D-03	2.06215D-02	26.78	2	2.00	2	7.52		4
	0.154947	5.32900D-03	2.06945D-02	25.75	2	2.00	2	7.49		5
Overall (IVPs)	0.784624 (0.701807)	0.284804	0.204960					3.83 (2.75)		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	0.325512	6.71550D-03	2.16100D-02	31.07	2	2.00	2	15.06	14.64	1
	0.318878	6.43449D-03	2.15720D-02	29.83	2	2.00	2	14.78		2
	0.318465	6.65000D-03	2.23010D-02	29.82	2	2.00	2	14.28		3
	0.318889	6.58400D-03	2.23015D-02	29.52	2	2.00	2	14.30		4
	0.318926	6.67500D-03	2.15585D-02	30.96	2	2.00	2	14.47		5
Overall (IVPs)	1.60771 (1.41344)	0.560412	0.384855					4.18 (2.87)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	0.367202	7.19851D-03	3.95065D-02	18.22	2	2.00	2	9.29	9.12	1
	0.361059	6.88950D-03	4.01290D-02	17.16	2	2.00	2	9.00		2
	0.360595	6.91400D-03	3.90340D-02	17.71	2	2.00	2	9.24		3
	0.360725	7.00650D-03	3.98740D-02	17.57	2	2.00	2	9.05		4
	0.367202	7.38000D-03	4.00495D-02	18.43	2	2.00	2	9.02		5
Overall (IVPs)	1.81819 (1.52722)	1.12046	0.759989					2.39 (1.62)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
8	0.678619	9.41265D-02	7.31	86.31
16	0.954879	0.118917	8.03	87.55
32	1.47341	0.204960	7.19	86.09
64	2.88210	0.384855	7.49	86.65
128	3.69986	0.759989	4.87	79.46

Table (b) : Speedup and Time savings relative to j-bvpms

Table 20: Parallel computational results for Problem A.3 (Problem size $n = 5$ and Error tolerance $= 10^{-5}$)

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	IVPs Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p = 1$	$p = 8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	0.226393	3.16350D-02	4.51625D-02	70.05	4	4.00	4	5.01	4.60	1
	0.244464	3.87155D-02	5.29995D-02	73.05	5	4.43	4	4.61		2
	0.236773	3.85640D-02	5.21520D-02	73.95	5	4.29	4	4.54		3
	0.236917	3.83570D-02	5.30370D-02	72.32	5	4.29	4	4.47		4
	0.238224	3.85500D-02	5.43700D-02	70.90	5	4.29	4	4.38		5
Overall (IVPs)	1.19292 (1.14331)	0.323828	0.273783					4.26 (3.68)		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	0.365865	2.49455D-02	4.23065D-02	58.96	3	3.00	3	12.93	8.61	1
	0.357473	2.47435D-02	4.17065D-02	59.33	3	3.00	3	13.00		2
	0.357260	2.49125D-02	4.16395D-02	59.83	3	3.00	3	13.02		3
	0.357900	2.38325D-02	4.15250D-02	57.39	3	3.00	3	13.06		4
	0.357219	2.43375D-02	4.13890D-02	58.80	3	3.00	3	13.10		5
Overall (IVPs)	1.80773 (1.67674)	0.310432	0.230332					7.85 (5.82)		
m	$p = 1$	$p = 32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	0.530332	2.13045D-02	3.87835D-02	54.93	2	2.00	2	13.47	13.53	1
	0.520364	1.96950D-02	3.81470D-02	51.63	2	2.00	2	13.64		2
	0.528811	2.10090D-02	3.76040D-02	55.87	2	2.00	2	13.80		3
	0.520319	2.25550D-02	3.83425D-02	58.83	2	2.00	2	13.57		4
	0.518845	2.05040D-02	3.94075D-02	52.03	2	2.00	2	13.17		5
Overall (IVPs)	2.61861 (2.38062)	0.506900	0.214520					12.21 (5.17)		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	1.10752	2.01685D-02	4.23175D-02	47.66	2	2.02	2	26.17	25.57	1
	1.08953	2.01975D-02	4.31415D-02	46.82	2	2.00	2	25.25		2
	1.08831	1.96605D-02	4.30660D-02	45.65	2	2.00	2	25.27		3
	1.08519	2.02985D-02	4.32765D-02	46.90	2	2.00	2	25.08		4
	1.08688	1.96680D-02	4.17010D-02	47.16	2	2.00	2	26.06		5
Overall (IVPs)	5.46785 (4.93767)	0.917932	0.400110					13.67 (5.96)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	1.20571	1.08750D-02	7.25595D-02	14.99	1	1.00	1	16.62	16.4	1
	1.21313	1.06995D-02	7.35350D-02	14.55	1	1.00	1	16.48		2
	1.18628	1.11800D-02	7.08570D-02	15.78	1	1.00	1	16.74		3
	1.18408	1.05615D-02	7.19885D-02	14.67	1	1.00	1	16.45		4
	1.18303	1.10310D-02	7.40815D-02	14.89	1	1.00	1	15.97		5
Overall (IVPs)	5.98271 (4.92880)	1.84234	0.792728					7.55 (3.25)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
8	1.29798	0.273783	4.74	78.91
16	2.52112	0.230332	10.94	90.86
32	3.30935	0.214520	15.43	93.52
64	7.08016	0.400110	17.70	94.35
128	8.23247	0.792728	10.38	90.37

Table (b) : Speedup and Time savings relative to j-bvpms

Table 21: Parallel computational results for Problem A.10 (Problem size $n = 8$ and Error tolerance $= 10^{-5}$)

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p=1$	$p=8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	0.334534	4.78705D-02	6.78080D-02	70.59	6	6.00	6	4.93	4.72	1
	0.747592	0.135868	0.163534	83.08	18	14.00	10	4.56		2
	0.665664	0.115643	0.148752	77.74	15	12.43	11	4.47		3
	0.694416	0.113396	0.146668	77.31	15	13.00	11	4.73		4
	0.695397	0.116097	0.142940	81.22	15	13.00	11	4.86		5
	0.697192	0.112105	0.145507	77.04	15	13.00	11	4.79		6
Overall (IVPs)	3.87483 (3.81002)	1.03959	0.981204					3.95 (3.73)		
m	$p=1$	$p=16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	0.604077	4.60550E-02	6.24765E-02	73.72	5	5.00	5	9.67	9.55	1
	0.903330	7.09385E-02	9.42025E-02	75.30	9	7.53	6	9.59		2
	0.824800	6.44045E-02	8.74600E-02	73.64	8	7.00	6	9.43		3
	0.834065	6.37675E-02	9.08640E-02	70.18	8	7.20	6	9.18		4
	0.859671	6.30075E-02	8.69905E-02	72.43	8	7.20	6	9.88		5
Overall (IVPs)	4.04957 (3.77846)	0.616204	0.460125					8.80 (6.57)		
m	$p=1$	$p=32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	1.03942	3.41505E-02	5.83680E-02	58.51	4	4.00	4	17.81	16.73	1
	1.12385	4.47450E-02	6.63745E-02	67.41	5	4.55	4	16.93		2
	1.07854	4.05365E-02	6.70730E-02	60.44	5	4.26	4	16.08		3
	1.10630	4.06630E-02	6.64745E-02	61.17	5	4.32	4	16.64		4
	1.10702	4.02730E-02	6.83335E-02	58.94	5	4.32	4	16.20		5
Overall (IVPs)	5.47795 (5.08297)	0.618705	0.341668					16.03 (8.85)		
m	$p=1$	$p=64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	1.61363	2.63830E-02	5.58435E-02	47.24	3	3.00	3	28.90	28.64	1
	1.58622	2.66615E-02	5.49190E-02	48.55	3	3.00	3	28.88		2
	1.58515	2.61150E-02	5.49320E-02	47.74	3	3.00	3	28.86		3
	1.58529	2.79760E-02	5.50245E-02	50.84	3	3.00	3	28.81		4
	1.58458	2.59270E-02	5.71065E-02	45.40	3	3.00	3	27.75		5
Overall (IVPs)	7.97868 (7.35946)	0.578288	0.285533					27.94 (13.80)		
m	$p=1$	$p=128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	2.27231	1.83855E-02	5.18700E-02	35.45	2	2.00	2	43.81	43.75	1
	2.20302	1.78925E-02	5.02685E-02	35.59	2	2.00	2	43.83		2
	2.25825	1.87545E-02	5.08205E-02	36.90	2	2.00	2	44.44		3
	2.24714	1.77665E-02	5.06095E-02	35.11	2	2.00	2	44.40		4
	2.23247	1.89855E-02	5.28230E-02	35.94	2	2.00	2	42.26		5
Overall (IVPs)	11.2371 (9.76937)	1.56266	0.264115					42.55 (7.19)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1,t_2)$	Time Savings $S_T(t_1,t_2)$
8	3.94385	0.981204	4.02	75.12
16	4.09178	0.460125	8.89	88.75
32	6.02668	0.341668	17.64	94.33
64	8.49652	0.285533	29.76	96.64
128	12.2771	0.264115	46.48	97.85

Table (b) : Speedup and Time savings relative to j-bvpms

Table 22: Parallel computational results for Problem A.10 (Problem size $n = 8$ and Error tolerance $= 10^{-10}$)

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	IVPs Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	2.41326	0.154297	0.217253	71.02	3	3.00	3	11.11	10.99	1
	2.39805	0.153821	0.218420	70.42	3	3.00	3	10.98		2
	2.39698	0.154456	0.218032	70.84	3	3.00	3	10.87		3
	2.39674	0.152587	0.218791	69.74	3	3.00	3	10.95		4
	2.39764	0.155197	0.218366	71.07	3	3.00	3	10.98		5
Overall (IVPs)	12.0157 (10.9652)	1.474188	1.27127					9.45 (8.15)		
m	$p = 1$	$p = 32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	3.53689	0.101693	0.216195	47.04	2	2.00	2	16.36	16.28	1
	3.51838	0.101428	0.217614	46.61	2	2.00	2	16.17		2
	3.51336	0.104511	0.214900	48.63	2	2.00	2	16.35		3
	3.51385	0.102763	0.214905	47.82	2	2.00	2	16.35		4
	3.51541	0.101211	0.215886	46.88	2	2.00	2	16.28		5
Overall (IVPs)	17.6112 (15.7444)	2.08197	1.35478					13.00 (8.46)		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	6.89747	0.121292	0.245693	49.37	2	2.00	2	28.07	28.14	1
	6.84858	0.121170	0.244187	49.62	2	2.00	2	28.05		2
	6.84981	0.120456	0.243627	49.44	2	2.00	2	28.12		3
	6.85589	0.120879	0.243243	49.69	2	2.00	2	28.19		4
	6.85151	0.119808	0.242524	49.40	2	2.00	2	28.25		5
Overall (IVPs)	34.3168 (30.6650)	4.15368	1.70879					20.08 (8.26)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	10.4999	0.273408	0.337008	81.13	1	1.00	1	31.16	31.04	1
	10.4731	0.272885	0.340708	80.09	1	1.00	1	30.74		2
	10.4380	0.275458	0.339545	81.13	1	1.00	1	30.74		3
	10.4354	0.270698	0.336899	80.35	1	1.00	1	31.27		4
	10.4482	0.272941	0.333753	81.78	1	1.00	1	31.31		5
Overall (IVPs)	52.3104 (45.0537)	8.32589	2.54831					20.53 (6.28)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
16	13.2183	1.27127	10.40	90.38
32	25.9103	1.35478	19.13	94.77
64	41.2638	1.70879	24.15	95.86
128	50.1808	2.54831	19.69	94.92

Table (b) : Speedup and Time savings relative to j-bvpms

Table 23: Parallel computational results for Problem A.12 (Problem size $n = 16$ and Error tolerance $= 10^{-5}$)

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	IVPs Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p = 1$	$p = 8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	3.67580D-02	6.26100D-03	1.14865D-02	52.69	8	7.86	7	3.20		1
	3.18685D-02	5.88250D-03	1.08550D-02	54.19	8	6.71	6	2.94	3.07	2
Overall (IVPs)	8.24252D-02 (6.90205D-02)	5.09700D-02	4.65975D-02					1.77 (1.62)		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	4.03005D-02	3.70450D-03	1.20725D-02	30.69	4	4.00	4	3.34		1
	3.53765D-02	3.44350D-03	1.17150D-02	29.39	4	3.53	3	3.02	3.18	2
Overall (IVPs)	8.95780D-02 (7.71900D-02)	6.92580D-02	6.10970D-02					1.47 (1.29)		
m	$p = 1$	$p = 32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	5.42500D-02	2.96500D-03	1.29610D-02	22.88	3	2.65	2	4.19		1
	4.35690D-02	2.76150D-03	1.29825D-02	21.27	3	2.10	2	3.36	3.78	2
Overall (IVPs)	0.111916 (9.80930D-02)	0.108011	9.61245D-02					1.16 (1.04)		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	8.53350D-02	2.85000D-03	1.46285D-02	19.48	2	2.00	2	5.83		1
	8.50600D-02	2.83950D-03	1.51120D-02	18.79	2	2.00	2	5.63	5.73	2
Overall (IVPs)	0.184359 (0.168068)	0.190479	0.171272					1.08 (0.97)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	1.01044D-02	3.28750D-03	2.83570D-02	11.59	1	1.00	1	3.56		1
	1.00476D-02	3.69350D-03	2.69545D-02	13.70	1	1.00	1	3.73	3.63	2
Overall (IVPs)	0.216111 (0.184384)	0.363744	0.314772					0.69 (0.59)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
8	8.45640D-02	4.65975D-02	1.81	44.90
16	9.83990D-02	6.10970D-02	1.61	37.91
32	0.119084	9.61245D-02	1.23	19.28
64	0.178077	0.171272	1.04	3.82
128	0.216620	0.314772	0.69	-45.31

Table (b) : Speedup and Time savings relative to j-bvpms

Table 24: Parallel computational results for Problem A.13 (Problem size $n = 2$, $\lambda = 20$, Error tolerance $= 10^{-5}$)

	bn-bvpms	pbn-bvpms								
	BN Time	IVPs Time	PBN Time	Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p = 1$	$p = 8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	8.52105D-02	1.27355D-02	1.84300D-02	69.10	18	17.43	17	4.62	3.71	1
	6.53085D-02	1.19205D-02	2.00065D-02	59.58	17	13.29	9	3.26		2
	5.83425D-02	1.11045D-02	1.64230D-02	67.62	17	11.57	9	3.55		3
	5.67305D-02	1.12080D-02	1.66695D-02	67.24	17	11.57	9	3.40		4
Overall (IVPs)	0.298345 (0.266800)	9.65280D-02	8.66685D-02					3.44 (3.10)		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	9.10330D-02	7.25500D-03	1.56565D-02	46.34	9	8.40	7	4.62	5.06	1
	7.03045D-02	8.14950D-03	1.37950D-02	59.08	9	6.67	4	3.26		2
	5.98640D-02	6.20350D-03	1.28615D-02	48.23	9	5.73	4	3.55		3
	5.93420D-02	6.26050D-03	1.27005D-02	49.29	9	5.73	4	3.40		4
Overall (IVPs)	0.312703 (0.291206)	0.131445	0.115558					2.71 (2.38)		
m	$p = 1$	$p = 32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	1.11237D-01	5.05200D-03	1.49385D-02	33.82	5	4.52	4	7.52	5.79	1
	7.94520D-02	4.43550D-03	1.46015D-02	30.38	5	3.45	2	5.44		2
	7.32590D-02	4.38850D-03	1.44030D-02	30.47	5	3.26	2	5.09		3
	7.29290D-02	4.39600D-03	1.43075D-02	30.73	5	3.26	2	5.10		4
Overall (IVPs)	0.371532 (0.333765)	0.195423	0.180350					2.06 (1.90)		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	1.33714D-01	4.09050D-03	1.45560D-02	28.10	3	2.67	2	9.19	6.56	1
	8.75325D-02	2.74150D-03	1.62575D-02	16.86	3	2.02	2	5.38		2
	8.77960D-02	3.00400D-03	1.53515D-02	19.57	3	2.03	2	5.72		3
	8.81910D-02	3.05350D-03	1.47880D-02	20.65	3	2.03	2	5.96		4
Overall (IVPs)	0.430407 (0.390283)	0.350090	0.320831					1.34 (1.23)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	1.83858D-01	3.37750D-03	2.75525D-02	12.26	2	1.57	1	6.67	4.94	1
	9.76500D-02	1.96700D-03	2.08460D-02	9.44	1	1.00	1	4.68		2
	9.74180D-02	2.04450D-03	2.28245D-02	8.96	1	1.00	1	4.27		3
	9.63650D-02	1.90100D-03	2.32090D-02	8.19	1	1.00	1	4.15		4
Overall (IVPs)	0.509779 (0.458393)	0.679459	0.62301					0.82 (0.75)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
8	0.182204	8.66685D-02	2.10	52.43
16	0.218874	0.115558	1.89	10.33
32	0.189358	0.180350	1.05	4.76
64	0.213366	0.320831	0.67	-50.37
128	0.282088	0.623010	0.45	-120.86

Table (b) : Speedup and Time savings relative to j-bvpms

Table 25: Parallel computational results for Problem A.13 (Problem size $n = 2$, $\lambda = 50$, Error tolerance $= 10^{-5}$)

	gir-bn-bvpms	gir-pbn-bvpms								
	GIR-BN Time	IVPs Time	GIR-PBN Time	IVPs Cost	IVPs No. Successful Steps			GIR-PBN Speedup		
m	$p = 1$	$p = 64$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	0.279985	7.17150D-03	3.31025D-02	21.66	2	2.00	2	8.46	9.39	1
	0.395605	9.74900D-03	3.70765D-02	26.29	3	2.48	2	10.67		2
	0.408569	1.95310D-02	3.95125D-02	49.43	7	2.67	2	10.34		3
	⋮	⋮						⋮		⋮
	0.277365	6.24600D-03	3.30800D-02	18.88	2	2	2	8.38		21
	0.277771	6.35100D-03	3.33895D-02	19.02	2	2	2	8.32		22
Overall (IVPs)	9.08483 (7.87348)	3.17341	1.67344					5.43 (2.86)		
m	$p = 1$	$p = 128$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	0.313897	7.01255D-03	0.101090	6.94	1	1.00	1	3.11	3.12	1
	0.469918	9.14500D-03	0.113871	8.03	2	1.30	1	4.13		2
	0.308761	6.81750D-03	0.112624	6.05	1	1.00	1	2.74		3
	⋮	⋮						⋮		⋮
	0.308146	6.54400D-03	0.104247	6.28	1	1.00	1	2.96		11
	0.307806	6.56300D-03	0.099938	6.57	1	1.00	1	3.08		12
Overall (IVPs)	5.32535 (3.79419)	3.38052	1.81565					2.93 (1.58)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	gir-pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
64	12.1758	1.67344	7.28	86.26
128	6.49497	1.81565	3.58	72.05

Table (b) : Speedup and Time savings relative to j-bvpms

Table 26: Parallel computational results for Problem A.8 (Problem size $n = 5$, Error tolerance $= 10^{-5}$)

	gir-bn-bvpms	gir-pbn-bvpms								
	GIR-BN Time	IVPs Time	GIR-PBN Time	IVPs Cost	IVPs No. Successful Steps			GIR-PBN Speedup		
m	$p = 1$	$p = 16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	2.28627	0.275770	0.291615	94.5665	179	97.20	11	7.84	9.32	1
	2.57753	0.269269	0.283805	94.8782	174	110.33	20	9.08		2
	2.61739	0.253223	0.279087	90.7326	168	110.87	23	9.38		3
	⋮			⋮				⋮		⋮
	2.47609	0.252150	0.273513	92.1894	170	106.4	25	9.05		20
	2.45054	0.242383	0.259059	93.5629	162	105.33	25	9.46		21
Overall (IVPs)	52.9870 (52.4736)	6.84117	6.32702					8.37 (7.75)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	gir-pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1, t_2)$	Time Savings $S_T(t_1, t_2)$
16	57.0518	6.32702	9.02	88.91

Table (b) : Speedup and Time savings relative to j-bvpms

Table 27: Parallel computational results for Problem A.11 (Problem size $n = 5$, Error tolerance $= 10^{-5}$)

	gir-bn-bvpms	gir-pbn-bvpms								
	BN	IVPs	PBN	IVPs				PBN		
	Time	Time		Cost	No. Successful Steps			Speedup		
m	$p=1$	$p=8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	0.266038	3.74190D-02	4.96660D-02	75.34	25	23.14	22	5.36	4.24	1
	9.26570D-02	1.28995D-02	2.55025D-02	50.58	9	8.43	8	3.63		2
	9.28705D-02	1.28800D-02	2.47990D-02	51.94	9	8.43	8	3.74		3
Overall (IVPs)	0.545763 (0.413048)	0.195914	0.121118					4.51 (2.79)		
m	$p=1$	$p=16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	0.319720	2.16965D-02	3.86080D-02	56.19	13	11.73	11	7.82	5.46	1
	9.537400D-02	7.30700D-03	2.16555D-02	33.72	4	4.00	4	4.40		2
	9.518800D-02	6.59600D-03	2.29100D-02	28.79	4	4.00	4	4.15		3
Overall (IVPs)	0.583333 (0.467088)	0.173272	0.145799					4.00 (3.37)		
m	$p=1$	$p=32$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	0.385872	1.44615D-02	3.28510D-02	44.02	7	6.39	5	11.75	7.96	1
	0.153736	5.84950D-03	2.51835D-02	23.23	3	3.00	3	6.10		2
	0.153879	6.16900D-03	2.55035D-02	24.19	3	3.00	3	6.03		3
Overall (IVPs)	0.791781 (0.638974)	0.231049	0.195217					4.06 (3.43)		
m	$p=1$	$p=32$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	0.433850	8.50750D-03	3.14070D-02	27.09	4	3.44	3	13.81	9.68	1
	0.210706	4.76200D-03	2.80240D-02	16.99	2	2.00	2	7.51		2
	0.209918	4.36500D-03	2.71390D-02	16.08	2	2.00	2	7.73		3
Overall (IVPs)	0.951995 (0.843075)	0.376786	0.298061					3.19 (2.53)		
m	$p=1$	$p=64$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	0.615983	6.38450D-03	3.41695D-02	18.68	2	2.00	2	18.03	11.11	1
	0.257760	3.17650D-03	3.11390D-02	10.20	1	1.00	1	8.28		2
	0.260850	3.23300D-03	3.07150D-02	10.53	1	1.00	1	7.02		3
Overall (IVPs)	1.45344 (1.00823)	0.681986	0.518270					2.80 (2.13)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	gir-pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1,t_2)$	Time Savings $S_T(t_1,t_2)$
8	0.479705	0.121118	3.96	74.75
16	0.592574	0.145799	4.06	75.39
32	0.556736	0.195217	2.85	64.94
64	0.667350	0.298061	2.24	55.34
128	0.803491	0.518270	1.55	64.50

Table (b) : Speedup and Time savings relative to j-bvpms

Table 28: Parallel computational results for Problem A.16 (Problem size $n = 3$ and Error tolerance $= 10^{-5}$)

	gir-bn-bvpms	gir-pbn-bvpms								
	BN Time	IVPs Time	PBN Time	IVPs Cost	IVPs No. Successful Steps			PBN Speedup		
m	$p=1$	$p=8$		%	Max	Aver	Min	Ratio	Aver	Iteration
8	1.62297	0.215358	0.251091	85.65	160	152.57	148	6.46	6.23	1
	0.563691	7.25325D-03	9.23555D-02	78.54	54	53.14	52	6.10		2
	0.564415	7.13385D-03	9.22770D-02	77.58	54	53.14	52	6.12		3
Overall (IVPs)	2.86277 (2.50006)	0.721939	0.457359					6.26 (3.97)		
m	$p=1$	$p=16$		%	Max	Aver	Min	Ratio	Aver	Iteration
16	1.68291	0.110807	0.134700	82.26	81	74.67	71	12.49	10.88	1
	0.566028	3.78125D-02	5.73620D-02	65.92	26	25.40	25	9.87		2
	0.566029	3.71320D-02	5.50075D-02	67.50	26	25.40	25	10.29		3
Overall (IVPs)	3.31806 (2.64379)	0.860020	0.492487					6.74 (3.86)		
m	$p=1$	$p=16$		%	Max	Aver	Min	Ratio	Aver	Iteration
32	1.91742	6.43970D-02	8.08995D-02	79.60	45	38.68	35	23.70	18.34	1
	0.624425	1.94875D-02	4.01535D-02	48.53	13	12.68	12	15.55		2
	0.624913	1.89445D-02	3.96455D-02	47.78	13	12.68	12	15.76		3
Overall (IVPs)	3.60449 (2.77738)	0.929439	0.530524					6.79 (3.88)		
m	$p=1$	$p=32$		%	Max	Aver	Min	Ratio	Aver	Iteration
64	2.08004	3.79985D-02	5.59705D-02	67.89	25	20.43	18	37.16	25.15	1
	0.657323	1.14305D-02	3.43865D-02	33.24	7	6.79	6	19.12		2
	0.662848	1.09995D-02	3.45580D-02	31.83	7	6.79	6	19.18		3
Overall (IVPs)	3.93047 (3.23541)	0.755489	0.633599					6.20 (5.20)		
m	$p=1$	$p=64$		%	Max	Aver	Min	Ratio	Aver	Iteration
128	2.48014	2.66140D-02	4.85445D-02	54.32	13	10.88	9	51.09	32.96	1
	0.827632	8.37500D-03	3.43265D-02	24.40	4	4.00	4	24.11		2
	0.827398	8.86000D-03	3.49605D-02	25.34	4	4.00	4	23.67		3
Overall (IVPs)	4.63454 (3.71627)	1.03370	0.848311					5.46 (4.48)		

Table (a) : PVM Timing (Sec.), Per-iteration speedup, and Total speedup

m	j-bvpms Time t_1 (sec)	gir-pbn-bvpms Time t_2 (sec)	Speedup $S_m(t_1,t_2)$	Time Savings $S_T(t_1,t_2)$
8	2.85005	0.457359	6.23	83.95
16	3.39485	0.492487	6.89	85.49
32	3.20766	0.530524	6.05	83.46
64	3.57184	0.633599	5.64	82.26
128	4.02601	0.848310	4.75	78.93

Table (b) : Speedup and Time savings relative to j-bvpms

Table 29: Parallel computational results for Problem A.16 (Problem size $n = 3$ and Error tolerance $= 10^{-10}$)

References

[AC91] U.M. Ascher and S.Y.P. Chan. On parallel methods for boundary value ODEs. *Computing*, 46:1–17, 1991.

[ACR+] 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. *To appear in SIAM Review.*

[AMR88] U.M. Ascher, R.M.M. Mattheij, and R.D. Rusell. *Numerical solution of boundary value problems for ordinary differential equations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[BF90] K.R. Bennett and G. Fairweather. Pcolnew : A parallel boundary-value ODE code for shared-memory machines. Technical Report CCS-90-8, University of Kentucky, Center for Computational Sciences, 1990.

[DB82] P. Deuflhard and G. Bader. Multiple shooting techniques revisited. Technical Report Preprint 163, Universitat Heidelberg, Institut fur Angewandte Mathematik, Im Neuenheimer Feld 293, 6900 Heidelberg, Federal Republic of Germany, June 1982.

[dHM83] F.R. de Hoog and R.M.M. Mattheij. On dichotomy and well-conditioning in BVP. Technical Report Report 8356, Katholieke Universiteit Nijmegen, Department of Maths., 1983.

[EHL75] W.H. Enright, T.E. Hull, and B. Lindberg. Comparing numerical methods for stiff system of O.D.E:s. *BIT*, 15:10–48, 1975.

[HEJ76] T.E. Hull, W.H. Enright, and K.R. Jackson. User's guide for DVERK - a subroutine for solving non-stiff ODE's. Technical Report 100, Department of Computer Science, University of Toronto, Toronto, Canada, 1976.

[Hig96] N. J. Higham. *Accuracy and stabilty of numerical algorithms.* SIAM Press, 1996.

[Hol64] J. F. Holt. Numerical soluiton of nonlinear two-point boundary problems by finite difference methods. *CACM*, 7(6):366–373, 1964.

[J9́2] J. JáJá. *An introduction to parallel algorithms.* Addison-Wesley, New York, 1992.

[JDS83] Jr J.E. Dennis and R.B. Schnabel. *Numerical methods for unconstrained optimiziation and nonlinear equations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[Jep] A.D. Jepson. An solver for matrices with banded, multi-value structure that uses Gaussian elimination with column pivoting. *Private Communication.*

[JP92] 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, Department of Computer Science, University of Toronto, 1992.

[JW77] M. Jankowski and H. Woźniakowski. Iterative refinement implies numerical stability. *BIT*, 17:303–311, 1977.

[Kel92] H.B. Keller. *Numerical methods for two-point boundary-value problems.* Dover, New York, 1992.

[KGGK94] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Intoduction to parallel computing : Design and analysis of algorithms.* Benjamin/Cummings, Redwood City, California, 1994.

[KM93] M.E. Kramer and R.M.M. Matheij. Application of global methods in parallel shooting. *SIAM J. Numer. Anal.*, 30(6):1723–1739, 1993.

[KR] R.M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report EECS CSD Report 88/408, Computer Science Division, UC Berkeley, California, 1988 (also appeared in Handbook of Theortical Computer Science : Volumn A : Algorithm and complexity, MIT press, 1990).

[Len88] M. Lentini. The potential for parallel algorithms in ordinary differential equations : boundary value problems. Technical Report 5/88, Lab. Nacional de Computacao Cientifica LNCC, Rio de Janeiro, 1988.

[Mat82] R.M.M. Mattheij. The conditioning of linear boundary value problems. *SIAM J. Numer. Anal.*, 19(5):963–978, 1982.

[MS88] R.M.M. Mattheij and G.W.M. Staarink. MUS : A package for solving two-point BVPs. Technical report, Economisch Institunt, Thomas van Aquinostraat 6, 6526, GD Nijmegen, The Netherlands, 1988.

[PG90] M. Paprzycki and I. Gladwell. A parallel chopping algorithm for ODE boundary value problem. Technical Report SMU MATH REPT 90-9, Department of Mathematics, South Methodist University, 1990.

[Ren76] P. Rentrop. Numerical solution of the singular Ginzburg-Landau equations by multiple shooting. *Computing*, 16:61–67, 1976.

[Ske80] R.D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Mathematics of Computation*, 35(151):817–832, 1980.

[WP90] S. Wright and V. Pereyra. Adaptation of a two-point boundary value problem solver to a vector-multiprocessor environment. *SIAM J. Sci. Statist. Comput.*, 11:425–449, 1990.

[Wri92] S. Wright. Stable parallel algorithms for two-point boundary value problems. *SIAM J. Sci. Statist. Comput.*, 13:742–764, 1992.

[Wri94] S. Wright. Stable parallel elimination for boundary value ODEs. *Numerische Mathematik*, 67(4):521–535, 1994.