ON PARALLEL METHODS FOR BOUNDARY VALUE ODES

by

Uri M. Ascher and S.Y. Pat Chan

Technical Report 89-19

September 1989

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5 Canada

On parallel methods for boundary value ODEs

Uri M. Ascher and S.Y. Pat Chan

September 1989

Department of Computer Science, University of British Columbia,

Vancouver, B.C. Canada V6T 1W5

Abstract

Some of the traditional methods for boundary value ODEs, such as standard multiple shooting, finite difference and collocation methods, lend themselves well to parallelization in the independent variable: the first stage of the construction of a solution approximation is performed independently on each subinterval of a mesh. However, the underlying possibly fast bidirectional propagation of information by fundamental modes brings about stability difficulties when information from the different subintervals is combined to form a global solution. Additional difficulties occur when a very stiff problem is to be efficiently and stably solved on a parallel architecture.

In this paper parallel shooting and difference methods are examined, a parallel algorithm for the stable solution of the resulting algebraic system is proposed and evaluated, and a parallel algorithm for stiff boundary value problems is proposed.

This research was supported in part by NSERC Canada Grant A4306.

1. Introduction

Recent advances in parallel computer architecture are allowing larger and larger scientific computations to be performed at a more and more affordable expense. One major condition for an efficient use of a parallel architecture is, however, the utilization of an appropriate algorithm, capable of taking advantage of such an architecture. A large volume of recent numerical literature has been devoted to the task of finding new parallel algorithms for various problems. Here we concentrate on such algorithms for solving a system of n ordinary differential equations (ODEs)

$$\mathbf{y}' = \mathbf{f}(t,\mathbf{y}), \qquad a < t < b \tag{1}$$

subject to n independent boundary conditions ,

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

A special case of this boundary value problem (BVP) is an initial value problem (IVP), where $\mathbf{y}(a)$ is given,

$$\mathbf{y}(a) = \alpha. \tag{3}$$

The ease with which one can find highly parallelizable algorithms for a given problem, and the extent to which this can be done, depend on the type of problem at hand. In case of ODEs one often encounters a fast propagation of information across the interval of integration in a way often depending on the traversed domain. This tends to suggest the use of marching algorithms which are sequential in nature. Questions of robustness, reliability and stability arise when one attempts to design a parallel algorithm for such a problem, as we shall see.

The design of a parallel algorithm is also strongly affected by the type of parallel architecture used. The most basic distinction among the variety of computing machines available is between those with a shared memory, where usually the number of processors is not very large (up to 32, say) and distributed memory, message passing machines, where the number of processors can reach thousands, and communication time among processors becomes important. Here, we wish to avoid getting to the specifics of machine architecture for most of the time, and we therefore imagine, unless otherwise specified, an ideal parallel machine with as many processors as required but with a negligible communication cost (i.e. a PRAM model, see e.g. [17]).

A general word of caution is regarding the reliability of proposed algorithms. In particular, questions of stability of a given algorithm often appear at a casual glance to be regarded as less central in recent literature. Of course, the property of stability of a given algorithm is as important for parallel algorithms as for serial ones. In fact, with the more powerful parallel computers carelessness in this respect may easily lead one to a situation where more meaningless numbers are obtained faster. A case in point arises in §3.

A sequential method for solving (1) subject to (2) or (3) usually proceeds by discretizing the ODE on a mesh in t. An algebraic relation is obtained, relating solution values at neighboring mesh points. Gear [8] has classified parallel algorithms for ODEs into two classes: i) "across the system", where each ODE or a group of ODEs is assigned to a different processor; for stiff IVPs and for BVPs this involves parallel techniques for solving large systems of algebraic equations, and ii) "across the method", where an algorithm is designed so that different parts of it can be executed on different processors in parallel. The basic advantage of an algorithm across the system would occur if an explicit difference scheme can be effectively used or if the system of ODEs has a special structure that can be utilized to advantage. For example, in circuit simulation in VLSI one obtains a very large set of ODEs, yielding a large algebraic system of equations to be solved at each time step. This system is, however, sparse, since each electric device on a chip has a direct connection to only a few neighbors. The "across the method" approach, on the other hand, is the more promising one when a robust, general purpose software is contemplated. We concentrate for the remainder of this paper on algorithms where work on a number of steps in t can be performed in parallel (cf. [21]).

Traditional numerical methods for BVPs in a way should lend themselves more easily than methods for IVPs to parallelization in t ("across the method"), because those for BVPs are *global:* the solution is obtained "everywhere" simultaneously. This is in contrast to methods for IVPs which are inherently local, in that the entire solution is known at the initial point and can be advanced locally; the latter is a sequential process by its nature. Of course IVPs, being a special case of BVPs, cannot be more difficult to solve numerically, regardless of machine architecture, but some of the more important advantages of IVP methods are among the method aspects which are most difficult to parallelize.

In BVPs, some of the most well-known methods for linear problems are obviously parallelizable: In standard multiple shooting, finite difference, collocation and finite element methods, the first stage of the construction of a solution approximation is done independently on each subinterval of a mesh and so can be assigned to different processors for different subintervals. Only at a later stage, that involving matching the solution pieces through a system of algebraic equations, is the parallelization question nontrivial. Moreover, a global (damped) Newton method (quasilinearization) yields at each iteration a linear BVP which can be solved by segments. Allowing more iterations in one segment than in another is possible, though perhaps not simple, but even without this the advantage of using parallel processors with a parallel shooting scheme, for instance, is clear. And yet, not everything is straightforward here either: The best shooting algorithms on a sequential machine are marching algorithms [4, Ch. 4], and problems arise also regarding the stability of prospective linear system solvers, as we further explore below.

A basic reason why a stable parallelization of BVP algorithms is not straightforward, particularly for stiff problems, is that even though the methods are global, the ODE still yields an underlying varying propagation of information, which may be fast for a stiff problem. Indeed, the situation is much more complicated here than for IVPs in this respect, because a well-conditioned (stable) BVP may support both rapidly increasing and rapidly decreasing fundamental solution modes. A stable solution algorithm must decouple these modes to a sufficient extent in some way [23], [14], [22].

Algorithms which attempt to achieve this decoupling explicitly, e.g. reorthogonalization, stabilized march or the Riccati method (cf. [10], [5], [6], [19], [4, §§4.4-4.5]) appear to be inherently sequential in t. There, unlike in the parallel shooting algorithm described in §2 below, information from the previous shooting subinterval is used to start integration on the next one. On the other hand, the standard multiple shooting method is a parallel shooting scheme. This method is also equivalent in some sense to the radically different finite difference approach (see, e.g., [4, Thm 5.38]) for nonstiff problems. For stiff problems the standard multiple shooting method requires too many shooting points, and becomes highly inefficient.

Other discretization methods exist which do not perform the decoupling of modes explicitly, but rather implicitly, for instance symmetric finite difference schemes. But this does not cause the issue to disappear: it resurfaces in the solution of the discretized ODEs. Questions of stability of various algorithms for solving linear algebraic equations arise, where a given algorithm must appropriately decouple the increasing and the decreasing discrete modes. Since an appropriate decoupling appears to be a sequential process when the problem does not have constant coefficients, the question of finding a parallel algorithm dealt with in §3 is in a way nontrivial indeed. We propose and evaluate such an algorithm.

In §4 we consider stiff BVPs. The obvious candidate methods are based on symmetric difference and collocation schemes, and we demonstrate their potential. However, symmetric schemes also have well-known drawbacks, both theoretical and practical (the latter mainly when layers resolution is not desired). We are then led to consider parallelizable methods which involve more work per mesh interval. The algebraic difference schemes approach of Schmidt [27] could be promising; a complete development and investigation of such schemes is at the time of this writing still underway for a sequential computer. Here we develop an algorithm based on the theoretical multiple shooting framework of [3].

2. Parallel shooting

As mentioned before, we apply for a nonlinear BVP (1),(2) some variant of Newton's method. This leaves us to deal, at each iteration, with the solution of a linear ODE

$$\mathbf{y}' = A(t)\mathbf{y} + \mathbf{q}(t), \quad a < t < b, \tag{4}$$

subject to linear boundary conditions

$$B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \boldsymbol{\beta}. \tag{5}$$

We concentrate first on multiple shooting [18], [26].

To recall, the well-known "standard" multiple shooting method was first introduced as a means for reducing the instability of the single shooting method for solving BVPs. The single shooting method attempts to solve BVPs by solving corresponding IVPs and matching the boundary conditions, but it becomes unstable essentially when these IVPs are unstable (ill-conditioned) even though the given BVP is well-conditioned. This instability of the single shooting method may grow exponentially with the size of the interval of integration, so in the multiple shooting method this interval is partitioned into a set of smaller intervals, and on each subinterval an IVP solution scheme is carried out.

However, the appeal of this approach here is its inherent parallelism. Here the integration on different shooting intervals is done independently, followed by an assembly of the solution pieces. An idea like this in the IVP context appears first in Nievergelt [24]. We partition the interval $[a,b]$ as follows

$$a = t_1 < t_2 < t_3 < \cdots < t_N < t_{N+1} = b. \tag{6}$$

Let $\mathbf{y}_i$ denote our approximation of $\mathbf{y}(t_i)$, $1 \le i \le N+1$. On each subinterval we now solve IVPs to approximate a fundamental solution $Y(t;t_i)$ and a particular solution $\mathbf{v}(t;t_i)$ of (4) on it, which in turn allows us to express $\mathbf{y}(t_{i+1})$ in terms of $\mathbf{y}(t_i)$, viz.

$$\mathbf{y}(t_{i+1}) = Y(t_{i+1};t_i)\mathbf{y}(t_i) + \mathbf{v}(t_{i+1};t_i). \tag{7}$$

This exact relationship is approximated in the integration by a "one step" relation of the form

$$R_i\mathbf{y}_{i+1} = S_i\mathbf{y}_i + \mathbf{q}_i, \quad 1 \le i \le N, \tag{8}$$

or more conveniently,

$$\mathbf{y}_{i+1} = \Gamma_i\mathbf{y}_i + \mathbf{r}_i, \quad 1 \le i \le N, \tag{9}$$

(a form which can be achieved in parallel for each i from (8)). Here $\Gamma_i := R_i^{-1}S_i$ approximates at $t=t_{i+1}$ the fundamental solution $Y(t;t_i)$ satisfying $Y(t_i;t_i)=I$. The form (8) (or (9)) is supplemented by the given boundary conditions

$$B_a\mathbf{y}_1 + B_b\mathbf{y}_{N+1} = \hat{\beta}. \tag{10}$$

A special case of this procedure is when (8) is just a usual one-step finite difference scheme, which corresponds to approximating the fundamental solution and particular solution on $[t_i,t_{i+1}]$ by just one integration step. Using collocation or implicit Runge-Kutta scheme in this way involves a more complex one integration step. More generally, it is important to realize that we have a *two-level mesh.* The coarse level is the mesh of shooting points (6). At the fine level mesh

$$a = x_1 < x_2 < x_3 < \cdots < x_M < x_{M+1} = b, \tag{11}$$

we have in addition to the points t_i the points used in discretizing the ODE on each subinterval $[t_i, t_{i+1}]$. For a simple collocation method the fine mesh includes all the collocation points (as well as the coarse mesh points (6)). Instead we may have a sequence of integration steps to cover $[t_i, t_{i+1}]$ using say some Runge-Kutta scheme (corresponding to a composite quadrature rule). The point is that *the fine level integrations, resulting in the construction of* Γ_i *and* $\mathbf{r}_i$, *can be done in parallel for different* i. The number N relates to the number of parallel processors available. If (8) stands for a simple (e.g. trapezoidal) one-step scheme then little has been gained as yet, because the fine mesh is equal to the coarse mesh. But if we couple a few simple steps in obtaining (8), with the result that the subinterval lengths $t_{i+1}-t_i$ are not necessarily small for a high accuracy, then we have a higher degree of parallelism.

To obtain one approximate solution for $\mathbf{y}(t)$ we write the $N+1$ relations (8),(10) as one system of algebraic equations

$$\mathbf{A}\mathbf{y}_\pi = \mathbf{g}_\pi \tag{12a}$$

for the vector of unknowns

$$\mathbf{y}_\pi := (\mathbf{y}_1^T, \mathbf{y}_2^T, \ldots, \mathbf{y}_{N+1}^T)^T. \tag{12b}$$

The matrix of this system, $\mathbf{A}$, is sparse. Let us denote by A_{ij} the i,j^{th} $n \times n$ block of $\mathbf{A}$. Then there are nonzero blocks in A_{ii}, $A_{i,i+1}$, $i=1,\ldots,N$ (these blocks are also nonsingular), $A_{N+1,1}$ and $A_{N+1,N+1}$:

$$\mathbf{A} = \begin{bmatrix} S_1 & R_1 & & & \\ & S_2 & R_2 & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & S_N & R_N \\ B_a & & & & B_b \end{bmatrix} \tag{13}$$

To recall (see, e.g., [4, §4.3]), if Γ_i of (8) approximates $Y(t_{i+1};t_i)$ sufficiently well for each i, then with

$$K := \max(\max_{1 \le i \le N} ||\Gamma_i|| \ , 1)$$

and κ the stability constant of the BVP (4),(5) the condition number of $\mathbf{A}$ satisfies

$$cond(\mathbf{A}) \le K\kappa N. \tag{14}$$

Thus, if the shooting points are chosen so that K is of moderate size and the linear algebraic system (12) is solved, say by Gaussian elimination with full row-partial pivoting, then the obtained algorithm is stable.

3. Solution of linear systems

Let us consider the case for IVPs first. Here $\mathbf{y}_1$ is given and in $\mathbf{A}$ we put the boundary equations as the first rows. Given (8) or (9), the solution may be obtained by recursion (in the form of iteration) on i, starting with $\mathbf{y}_1$. This recursion is stable under very mild conditions, namely, if for $1 \le i \le N$, Γ_i does not give rise to rapidly increasing (discrete) modes (see [4, Ch. 6]). This is the case if the IVP is stable and Γ_i is a reasonably good approximation to $Y(t_{i+1};t_i)$, all i, or if a damping difference scheme like a backward differentiation formula (BDF) is used. The algorithm requires $O(N)$ operations when done in series. (The time complexity is generally proportional here also to n^3 of course, but we concentrate on the dependence on N for a fixed n.)

However, using an appropriate variant of the odd/even reduction algorithm (cf. [7],[9]) the solution of the linear system can be done in an $O(logN)$ time complexity if $O(N)$ processors are utilized. The algorithm is illustrated in Fig. 1, for the case $N=7$. The serial recursion described before involves Gaussian elimination of the blocks $A_{i+1,i}$ using A_{ii}. The algorithm consists of a sequence of steps each involving a series of block Gauss elimination operations with restricted partial pivoting. Each of these steps is done in parallel. Thus, assuming for convenience that $N=2^m-1$ for some positive integer m, the first step consists of elimination of $A_{2j,2j-1}$ using $A_{2j-1,2j-1}$, $1 \le j \le \frac{N+1}{2}=2^{m-1}$. This introduces new nonzero blocks in $A_{2j,2j-2}$. In the next

$$\underset{(a)}{\begin{bmatrix} \times &&&&&&& \\ \times & \times &&&&&& \\ & \times & \times &&&&& \\ && \times & \times &&&& \\ &&& \times & \times &&& \\ &&&& \times & \times && \\ &&&&& \times & \times & \\ &&&&&& \times & \times \end{bmatrix}} \rightarrow \underset{(b)}{\begin{bmatrix} \times &&&&&&& \\ 0 & \times &&&&&& \\ & \times & \times &&&&& \\ & + & 0 & \times &&&& \\ &&& \times & \times &&& \\ &&& + & 0 & \times && \\ &&&&& \times & \times & \\ &&&&& + & 0 & \times \end{bmatrix}} \rightarrow$$

$$\rightarrow \underset{(c)}{\begin{bmatrix} \times &&&&&&& \\ & \times &&&&&& \\ & 0 & \times &&&&& \\ & 0 && \times &&&& \\ &&& \times & \times &&& \\ &&& \times && \times && \\ &&& + && 0 & \times & \\ &&& + && 0 && \times \end{bmatrix}} \rightarrow \underset{(d)}{\begin{bmatrix} \times &&&&&&& \\ & \times &&&&&& \\ && \times &&&&& \\ &&& \times &&&& \\ &&& 0 & \times &&& \\ &&& 0 && \times && \\ &&& 0 &&& \times & \\ &&& 0 &&&& \times \end{bmatrix}}$$

Figure 1 - odd/even reduction for IVPs.
In this example, $N=2^m-1$, $m=3$. The $\times$'s denote nonzero $n\times n$ blocks, 0 denotes a currently eliminated block, and $+$ denotes a new nonzero block generated during the current step. Note that operation within each of the m block Gaussian elimination steps leading from (a) to (d) can be done simultaneously.

next step, the blocks $A_{4j-2,4j-2}$ are used to eliminate $A_{4j-1,4j-2}$ and $A_{4j,4j-2}$, $1 \leq j \leq 2^{m-2}$, etc. After m such steps, each utilizing $N/2$ processors, we are left with a block diagonal matrix, and a final parallel step yields the solution $\mathbf{y}_\pi$.

This algorithm is "efficient" but not "optimal" [17] in that if performed serially it has a running time $O(N\log N)$, not the optimal $O(N)$. In fact, the number of processors can be cut down to $O(N/\log N)$, still maintaining an $O(\log N)$ parallel time. This is done using a standard trick to avoid computing unnecessary intermediate quantities. At first, we divide the system into $2^m/m$ groups of m block rows each, $m=\log(N+1)$. Within each group, block elimination is performed serially, using $2^m/m$ processors for the entire system. In the second stage consider only block rows $i=km$, $k=1,...,2^m/m$, and solve for these $\mathbf{y}_i$'s using the odd/even algorithm described above. The last step involves using the kth processor to solve for $\mathbf{y}_i$, $km<i<(k+1)m$, using $\mathbf{y}_{km}$ as a starting value, for $k = 1,...,2^m/m$. Each of these 3 steps can be performed in $O(logN)$ time using $\sim N/\log N$ processors.

It is not difficult to see that the above described parallel and serial algorithms yield precisely the same results. Hence, this parallel algorithm is also stable under very mild conditions.

This algorithm can be readily modified to yield a similar one for BVPs. The obtained algorithm is depicted pictorially in Fig. 2. Each of the steps shown can again be performed in parallel, yielding an $O(logN)$ time complexity if N processors are utilized. (The latter number can again be reduced to $O(N/\log N)$.)

Unfortunately, the BVP algorithm is *not stable* in general! It is easily seen that the prescribed elimination steps produce the single shooting matrix $B_a+B_b\Gamma_N\Gamma_{N-1} \cdots \Gamma_1$ which could have extremely large elements even if the BVP is well-conditioned (i.e. even if κ is of a moderate size, cf. [4, §4.3.4]). (Note that for a stable IVP, in contrast, this matrix is always nicely bounded.) This algorithm, in fact, produces the same output as the compactification

$$\begin{bmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \times & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & \\ & & & & & & & \times & \times \\ \times & & & & & & & & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & & & & & & & \\ + & 0 & \times & & & & & & \\ & & \times & \times & & & & & \\ & & + & 0 & \times & & & & \\ & & & & \times & \times & & & \\ & & & & + & 0 & \times & & \\ & & & & & & \times & \times & \\ & & & & & & + & 0 & \times \\ \times & & & & & & & & \times \end{bmatrix} \rightarrow$$

(a) (b)

$$\rightarrow \begin{bmatrix} \times & \times & & & & & & & \\ \times & & \times & & & & & & \\ + & & 0 & \times & & & & & \\ + & & 0 & & \times & & & & \\ & & & & \times & \times & & & \\ & & & & \times & & \times & & \\ & & & & + & & 0 & \times & \\ & & & & + & & 0 & & \times \\ \times & & & & & & & & \times \end{bmatrix} \rightarrow$$

(c)

$$\rightarrow \begin{bmatrix} \times & \times & & & & & & & \\ \times & & \times & & & & & & \\ \times & & & \times & & & & & \\ \times & & & & \times & & & & \\ + & & & & 0 & \times & & & \\ + & & & & 0 & & \times & & \\ + & & & & 0 & & & \times & \\ + & & & & 0 & & & & \times \\ \times & & & & & & & & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & & & & & & & \\ \times & & \times & & & & & & \\ \times & & & \times & & & & & \\ \times & & & & \times & & & & \\ \times & & & & & \times & & & \\ \times & & & & & & \times & & \\ \times & & & & & & & \times & \\ \times & & & & & & & & \times \\ \times & & & & & & & & 0 \end{bmatrix}$$

(d) (e)

Figure 2 - straightforward odd/even reduction is equivalent to the compactification algorithm. *Here $N=2^m$, $m=3$. Notation is as in Fig. 1.*

algorithm which has long been recognized to be possibly unstable [26], [11]. The algorithm does not achieve an appropriate decoupling of fast increasing and fast decreasing modes when such modes occur in a given problem; indeed, it ignores this question by not consulting the boundary conditions until the last step. For separated boundary conditions the left end boundary conditions "control" the decreasing modes while those at the right interval end "control" the increasing modes.

Note that the compactification algorithm performs Gaussian elimination with some partial pivoting. However, it does not include full across-blocks pivoting, thereby missing a crucial stabilizing effect that the full row pivoting achieves [26].

In the following we will consider the case where the boundary conditions (5) are separated:

$$B_a = \begin{pmatrix} 0 \\ B_0 \end{pmatrix} \quad B_b = \begin{pmatrix} B_1 \\ 0 \end{pmatrix} .$$

It is then natural to move the last rows to be first in $\mathbf{A}$, obtaining the structure

$$\mathbf{A} = \begin{bmatrix} B_0 & & & & & \\ S_1 & R_1 & & & & \\ & S_2 & R_2 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & S_N & R_N \\ & & & & & B_1 \end{bmatrix} \tag{15}$$

Almost all traditional algorithms proposed in the literature are for the case of separated boundary conditions (cf. [4, Ch. 7]). The quest is then to find a *stable*, efficient parallel algorithm for a discrete BVP (12) with (15) holding. The remarks above suggest, however, that this may not be an easy task: If work in parallel is to be performed on $\mathbf{A}$ then how are the boundary conditions supposed to "control" the fast modes?

Indeed, a number of algorithms recently proposed do not appear to be generally stable. A stable algorithm proposed by Lentini [20], where mode decoupling proceeds simultaneously from both interval ends towards the middle, offers only a limited parallelism. Another suggestion made is to proceed with the above described parallel compactification algorithm, monitoring stability by an iterative refinement technique and switching to a stable alternative with time complexity $O(N)$ when an instability is detected. This may prove to be a practical remedy until something better is found. In any case, a theoretical question remains open, namely to find a direct, stable parallel algorithm with complexity $O(logN)$.

We now offer what is the closest that we know of to a positive answer to this question. The idea is simple: Consider the least squares solution of (12), obtaining the normal equations

$$\mathbf{A}^T\mathbf{A}\mathbf{y}_\pi = \mathbf{A}^T\mathbf{g}_\pi. \tag{16}$$

Now if $\mathbf{A}$ has the form (15) then the matrix $\mathbf{B} := \mathbf{A}^T\mathbf{A}$ is block tridiagonal. The formation of the blocks of $\mathbf{B}$ can be done in $O(n^3)$ time using $O(N)$ processors in an obvious way. Also, since $\mathbf{A}$ is nonsingular, $\mathbf{B}$ is (symmetric) positive definite. For such matrices there are stable versions of odd/even reduction and odd/even elimination - see Heller [12], Johnsson [15] and Kapur & Browne [16], which perform in $O(logN)$ time using $O(N)$ processors in parallel. In Fig. 3 we depict an instance of one version of this algorithm.

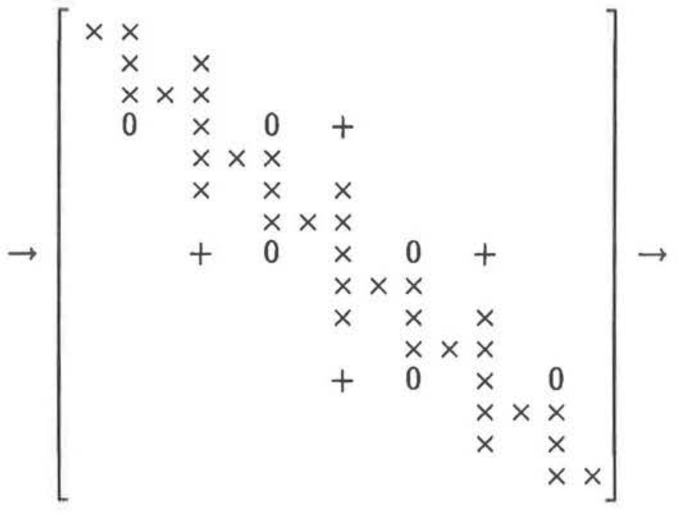
$$\left[\begin{array}{ccccccccccccccc}
\times & \times & & & & & & & & & & & & & \\
\times & \times & \times & & & & & & & & & & & & \\
& \times & \times & \times & & & & & & & & & & & \\
& & \times & \times & \times & & & & & & & & & & \\
& & & \times & \times & \times & & & & & & & & & \\
& & & & \times & \times & \times & & & & & & & & \\
& & & & & \times & \times & \times & & & & & & & \\
& & & & & & \times & \times & \times & & & & & & \\
& & & & & & & \times & \times & \times & & & & & \\
& & & & & & & & \times & \times & \times & & & & \\
& & & & & & & & & \times & \times & \times & & & \\
& & & & & & & & & & \times & \times & \times & & \\
& & & & & & & & & & & \times & \times & \times & \\
& & & & & & & & & & & & \times & \times & \times \\
& & & & & & & & & & & & & \times & \times
\end{array}\right] \rightarrow$$

(a)

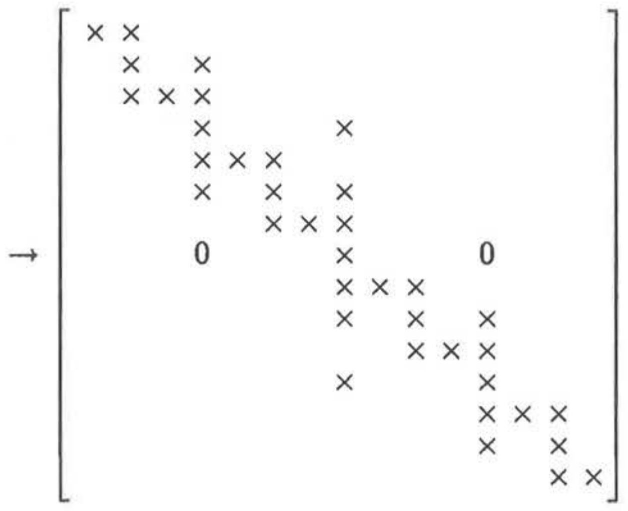
$$\rightarrow \left[\begin{array}{ccccccccccccccc}
\times & \times & & & & & & & & & & & & & \\
0 & \times & 0 & + & & & & & & & & & & & \\
& \times & \times & \times & & & & & & & & & & & \\
& + & 0 & \times & 0 & + & & & & & & & & & \\
& & & \times & \times & \times & & & & & & & & & \\
& & & + & 0 & \times & 0 & + & & & & & & & \\
& & & & & \times & \times & \times & & & & & & & \\
& & & & & + & 0 & \times & 0 & + & & & & & \\
& & & & & & & \times & \times & \times & & & & & \\
& & & & & & & + & 0 & \times & 0 & + & & & \\
& & & & & & & & & \times & \times & \times & & & \\
& & & & & & & & & + & 0 & \times & 0 & + & \\
& & & & & & & & & & & \times & \times & \times & \\
& & & & & & & & & & & + & 0 & \times & 0 \\
& & & & & & & & & & & & & \times & \times
\end{array}\right] \rightarrow$$

(b)

Figure 3 - odd/even reduction for a block tridiagonal system.
Here N=2^m–2, m=4. Notation is as in Fig. 1.



(c)



(d)

Figure 3 (cont.) - odd/even reduction for a block tridiagonal system.

Here $N=2^m-2$, $m=4$. Notation is as in Fig. 1.

In the first step, odd-numbered block rows above and below each even-numbered block row are used to eliminate odd-numbered blocks in it (see Fig. 3(b)). The subsystem formed by every second row block and every second column block is then again block tridiagonal, but of size $\sim N/2$, and a similar step is now applied to it, etc. Each of the m steps (with $N+1=2^m-1$ for convenience) can be performed in parallel. A somewhat unusual backward substitution phase follows (see Fig. 3(d)), starting from the middle block (block row 2^{m-1}) and continuing with block rows 2^{m-2} and $3\cdot 2^{m-2}$ etc., using in total another m parallel steps to recover the solution.

As for the IVP recursion, the number of processors can be cut down to $\sim N/\log N$, still maintaining an $O(logN)$ parallel running time. The way to achieve this is again to divide the block rows into groups of $\sim \log N$ members each. Formation of $\mathbf{B}$ in $O(\log N)$ time is obvious. Elimination within each group is performed serially. Then the set of last block rows of each group is considered as one block tridiagonal system of size $N/\log N$ and the above odd/even reduction algorithm is applied.

The odd/even reduction algorithm is *stable* because it is simply block Gaussian elimination applied to $\mathbf{P}^T\mathbf{A}^T\mathbf{A}\mathbf{P}$ for some permutation matrix $\mathbf{P}$, and $\mathbf{A}^T\mathbf{A}$ is symmetric positive definite [12], [25].

Example 1

The above described algorithms have been implemented on a SUN 3/50 (with the parallelism simulated) using an f77 compiler with double precision (14 hexadecimal digits). The standard multiple shooting method was used, with an accurate IVP integrator (i.e. the fine mesh (11) was much more dense than the shooting mesh (6)).

The example considered has the form (4),(5) with a=0, b=1,

$$A = \begin{pmatrix} -\lambda cos2\omega t & \omega+\lambda sin2\omega t \\ -\omega+\lambda sin2\omega t & \lambda cos2\omega t \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} e^t \\ e^t \end{pmatrix},$$

$$B_0 = \begin{pmatrix} 1 & 0 \end{pmatrix} = B_1.$$

A fundamental solution is given by

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

from which we see that the parameter λ controls the growth and decay of modes, while the parameter ω controls their rotation (cf. [4, Example 4.13]).

In Table 1 we list errors in maximum norm for the first component of $\mathbf{y}$ over the shooting points, comparing full row pivoting (MSH), the direct odd/even reduction algorithm equivalent to the compactification algorithm (MSC) and the least squares odd/even reduction algorithm (OER). Also listed are the number of uniformly spaced shooting points N and the estimated condition number of $\mathbf{A}$ (cond).

From these results it is clear that while the direct odd/even reduction algorithm (MSC) can easily become unstable (this happens when e^{λ} is large), the least squares odd/even reduction algorithm yields satisfactory results as long as $cond(\mathbf{A})^2$ does not become too large. The stability of the algorithm does not appear to depend on ω. For this example, with uniformly spaced shooting points, we can estimate $cond(\mathbf{A}) \sim Ne^{\lambda/N}$. So the anticipated roundoff error magnification for MSH is $\sim Ne^{\lambda/N}$, while that for OER is $\sim N^2 e^{2\lambda/N}$. The numerical results bear this out.

λ	ω	N	cond	MSH	OER	MSC
1	1	7	0.2e+1	0.33e-10	0.33e-10	0.33e-10
1	50	7	0.5e+1	0.31e-7	0.31e-7	0.31e-7
50	1	7	0.6e+3	0.14e-6	0.15e-6	0.65e+3
100	1	7	0.8e+6	0.43e-6	0.63e+2	0.16e+22
100	1	15	0.4e+3	0.27e-7	0.27e-7	0.12e+26
150	1	15	0.1e+5	0.53e-7	0.29e-3	0.24e+45
150	1	31	0.6e+2	0.34e-8	0.34e-8	0.40e+48

Table 1. Comparison of linear solution algorithms using multiple shooting.

$\square$

Other examples have been tried where the shooting points were spread nonuniformly, and where there were more than one fast increasing mode, with qualitatively similar results.

The description of the least squares odd/even algorithm has been for separated BC only. However, Heller [13, p. 39] already notes that a similar principle can be applied for the non-separated BC case, i.e. when $\mathbf{A}$ has the form (13) and correspondingly $\mathbf{B}$ has nonzero blocks in the lower left and upper right corners.

The implementation of odd/even algorithms for block tridiagonal systems on specific parallel architectures has been described by a number of authors. In particular, Kapur and Browne [16] discuss implementation on shared memory machines, proposing use of the odd/even elimination variant (see also [25]). Johnsson [15] discusses in detail implementation issues on distributed memory architectures.

The proposed least squares odd/even reduction algorithm does have two disadvantages. Perhaps the more practical disadvantage is the increased amount of storage and also of sequential operations required. The increase factor is only about 2, though. The other disadvantage is that the condition number involved is that of $\mathbf{A}^2$, not of $\mathbf{A}$. If K or κ are large in (14), there could occur a situation where, as in Table 1, a roundoff error magnification factor of $K\kappa N$ is tolerable, whereas one of $(K\kappa N)^2$ is not. From a theoretical point of view, the desire to find a parallel $O(logN)$ algorithm for solving (12) which is as stable as Gaussian elimination with full row-partial pivoting is not completely satisfied.

4. Solving stiff problems

If the BVP is very stiff, the standard multiple shooting method becomes inefficient. For instance, in Example 1 we must take $N = O(\lambda)$ as $\lambda \to \infty$ in order to keep $cond(\mathbf{A})$ from growing (cf. (14)). Theoretical and practical considerations for the sequential case are given in [3], [2].

In [1], [2] it was shown that symmetric schemes based on collocation at Gaussian points perform very well in many practical situations on a sequential processor. Since these schemes can also be naturally parallelized, they are obvious strong contenders here as well.

Example 2

We have implemented the algorithms of §3 also with the midpoint scheme (i.e. one step of the midpoint scheme is performed between each two points t_i and t_{i+1} of (6)). For the problem considered in Example 1, boundary layers are generally possible when λ is large, and the mesh has to be made fine near the interval ends. Still, the mesh can be chosen coarse (independent of λ) away from the boundaries with N independent of λ (see, e.g., [4, §10.2]). Note that the bound (14) does not hold here for the condition number of $\mathbf{A}$.

Nonetheless, in all our computations, which included some very large values of λ, no difference between the results produced by the MSH and the OER solvers was observed.

$\square$

It is well-known, however, that despite their practical success in many instances, theoretical and practical difficulties do exist when symmetric schemes are applied to very stiff problems (see, e.g. [4, §10.2]). Basically, a symmetric scheme approximates a fast decreasing or increasing mode by a slow decreasing or increasing one, respectively, and while this keeps the norm of the approximation to $Y(t_{i+1};t_i)$ bounded, the discrepancy may cause difficulties in certain cases.

We therefore utilize the *theoretical multiple shooting* framework discussed in [3]: instead of the unstable IVPs solved in §2 to obtain (9) as an approximation of (7) for each i, consider stable local BVPs. On a subinterval $[t_i, t_{i+1}]$, denote by $\Phi_i(t)$ the fundamental solution of (4) and by $\mathbf{v}_i(t)$ the particular solution satisfying

$$\mathbf{B}_i\Phi_i \equiv B_{1i}\Phi_i(t_i^+) + B_{2i}\Phi_i(t_{i+1}^-) = I, \tag{17a}$$

$$\mathbf{B}_i\mathbf{v}_i = 0. \tag{17b}$$

Writing

$$\mathbf{y}(t) = \Phi_i(t)\mathbf{s}_i + \mathbf{v}_i(t), \qquad t_i \le t \le t_{i+1} \tag{18}$$

we get, similarly to (12), (13),

$$\mathbf{A}\mathbf{s}_\pi = \mathbf{g}_\pi \tag{19a}$$

for the vector of unknowns

$$\mathbf{s}_\pi := (\mathbf{s}_1^T, \mathbf{s}_2^T, \ldots, \mathbf{s}_N^T)^T, \tag{19b}$$

the right hand side vector

$$\mathbf{g}_\pi := (\mathbf{g}_1^T, \mathbf{g}_2^T, \ldots, \mathbf{g}_N^T)^T, \tag{19c}$$

$$\mathbf{g}_i = \mathbf{v}_{i+1}(t_{i+1}) - \mathbf{v}_i(t_{i+1}), \quad 1 \le i \le N-1, \qquad \mathbf{g}_N = \beta - B_a\mathbf{v}_1(a) - B_b\mathbf{v}_N(b),$$

and the matrix

$$\mathbf{A} = \begin{bmatrix} \Phi_1(t_2) & -\Phi_2(t_2) & & & & \\ & \Phi_2(t_3) & -\Phi_3(t_3) & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \Phi_{N-1}(t_N) & -\Phi_N(t_N) \\ B_a\Phi_1(a) & & & & & B_b\Phi_N(b) \end{bmatrix}. \tag{19d}$$

The question of how to choose the local BC $\mathbf{B}_i$ so that (19) is stable is answered theoretically in [3]: Assuming that the original BVP (4),(5) is well-conditioned there is a dichotomy.

Writing without loss of generality $\Phi(t) = (\Phi^1(t) \mid \Phi^2(t))$ for the fundamental solution of (4) satisfying

$$B_a\Phi(a) + B_b\Phi(b) = I \tag{20}$$

with Φ^1 the $n-p$ nondecreasing modes and Φ^2 the p nonincreasing modes, choose

$$B_{1i} := \begin{pmatrix} 0 \\ Q_{1i}^T \end{pmatrix} \qquad B_{2i} := \begin{pmatrix} Q_{2i}^T \\ 0 \end{pmatrix} \tag{21}$$

where Q_{1i} and Q_{2i} are matrices with p and $n-p$ orthonormal columns respectively, satisfying

$$Q_{1i}^T\Phi^1(t_i) = 0 \qquad Q_{2i}^T\Phi^2(t_{i+1}) = 0. \tag{22}$$

The idea is then to devise a practical method for a parallel solution of stiff BVPs by devising stable subproblems to be solved in parallel. The solution of the linear system (19) has been discussed in §3. It remains to discuss obtaining (19) for given local BC $\mathbf{B}_i$, and then obtaining a practical approximation of (21),(22).

Suppose first that we are given $\mathbf{B}_i$ for a fixed i. In (4),(17) we have $n+1$ local BVPs, one for each column of $\Phi_i(t)$ and one for $\mathbf{v}_i(t)$. Apply some difference or other stable scheme to approximate these BVPs, using the same mesh, say

$$t_i = x_{M_i} < x_{M_i+1} < \cdots < x_{M_{i+1}} = t_{i+1}.$$

We obtain a linear system of equations with a matrix $\mathbf{A}_i$ of the form (15) and $n+1$ right hand sides. These right hand sides can be constructed and solved for in parallel. Note that in standard multiple shooting the elements of a matrix with a similar zero structure to $\mathbf{A}_i$ are also formed, but the decomposition in that case is simpler, as in §3, because that is a local IVP: $B_{1i}=I$, $B_{2i}=0$. Obviously, some efficiency is lost by solving local BVPs instead of local IVPs. However, much fewer shooting points are needed: For instance, $O(\lambda)$ shooting points are needed with the standard multiple shooting method for the problem of Example 1, while here the number of shooting points is independent of λ.

Next consider obtaining local BC. To actually achieve (21),(22) we need a full decoupling of the ODE, i.e. a solution of the Lyapunov equation, or the kinematic eigenvalues. This is impractical as is, but in order to approximate it let us recall the transformation. Let

$$\mathbf{w}(t) = T^{-1}(t)\mathbf{y}(t) \tag{23a}$$

i.e.

$$\mathbf{w}' = U\mathbf{w} + T^{-1}\mathbf{q} \tag{23b}$$

$$U(t) = T^{-1}AT - T^{-1}T' \tag{23c}$$

with U upper triangular. The diagonal elements of U are kinematic eigenvalues, and we may assume that they are ordered by decreasing real parts. The number $n-p$ of columns in Φ^1 is determined by the number of nonzero rows in B_b in case of separated BC. Otherwise it may be determined as a number such that the real part of the kinematic eigenvalues from $n-p+1$ on is not large positive. For (23) we can choose stable local BC by specifying the last p components of $\mathbf{w}$ at t_i and the first $n-p$ components of $\mathbf{w}$ at t_{i+1} for each segment $[t_i, t_{i+1}]$.

Now, if $T^{-1}T'$ is negligible compared to $T^{-1}AT$, where T is the orthogonal matrix obtained by the QR algorithm for A at each x, then the eigenvalues of A give adequate approximations to the kinematic ones. This occurs if the segmentation (6) is chosen as described in [3, §1] and the other assumptions there hold as well, namely, no rapid solution oscillations occur on "long" segments, and the points t_i are placed outside layers, in smooth solution segments.

Thus, assume that an appropriate segmentation (6) is formed. Applying the QR algorithm for A at each t_i in parallel and calling the resulting orthogonal matrix T_i, we may assume that the eigenvalues are ordered as described above. Then set

$$Q_{1i}^T = \text{last } p \text{ rows of } T_i^{-1}$$

$$Q_{2,i-1}^T = \text{first } n\text{–}p \text{ rows of } T_i^{-1}$$

where p is determined independently of i. Finally, use (21).

Note that if the transformations T_i are saved then the solution $\{\mathbf{y}_i\}$ may be formed in parallel after solving (19) as follows: writing

$$\mathbf{s}_i = \begin{pmatrix} \mathbf{s}_i^1 \\ \mathbf{s}_i^2 \end{pmatrix}$$

we get

$$\mathbf{y}_i = T_i \begin{pmatrix} \mathbf{s}_{i-1}^1 \\ \mathbf{s}_i^2 \end{pmatrix} . \tag{24}$$

A practical investigation of this algorithm will be reported in a future work.

References

[1] U. Ascher, J. Christiansen and R.D. Russell, "Collocation software for boundary value ODEs", Trans. Math. Software 7 (1981), 209-222.

[2] U. Ascher and S. Jacobs, "On collocation implementation for singularly perturbed two-point problems", SIAM J. Scient. Stat. Comput. 10 (1989), 533-549.

[3] U. Ascher and R.M.M. Mattheij, "General framework, stability and error analysis for numerical stiff boundary value problems", Numer. Math. 54 (1988), 355-372.

[4] U. Ascher, R.M.M Mattheij and R.D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations,* Prentice-Hall (1988).

[5] S.D. Conte, "The numerical solution of linear boundary value problems", SIAM Review 8 (1966), 309-321.

[6] L. Dieci, M.R. Osborne and R.D. Russell, "A Riccati transformation method for solving boundary value problems, I: theoretical aspects", SIAM J. Numer. Anal. 25 (1988), 1055-1074.

[7] T.N Gambill and R.D. Skeel, "Logarithmic reduction of the wrapping effect with application to ordinary differential equations", SIAM J. Numer. Anal. 25 (1988), 153-162.

[8] C.W. Gear, "The potential for parallelism in ordinary differential equations", Tech. Rep. 86-1246, Computer Science Dept. University of Illinois, Urbana (1986).

[9] C.W. Gear, "Massive parallelism across the method in ODEs", Tech. Rep. 88-1442, Computer Science Dept. University of Illinois, Urbana (1988).

[10] S. K. Godunov, "Numerical solution of boundary value problems for systems of linear ordinary differential equations", Usp. Mat. Nauk 16 (1961), 171-174

[11] N.A. Haskell, "The dispersion of surface waves in multilayered media", Bull. Seism. Soc. Am. 43 (1953), 17-34.

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

[13] D. Heller, "Direct and iterative methods for block tridiagonal linear systems", PhD Thesis, Dept. Computer Science, CMU 1977.

[14] F. de Hoog and R.M.M. Mattheij, "On dichotomy and well-conditioning in BVP", SIAM J. Numer. Anal. 24 (1987), 89-105.

[15] S.L. Johnsson, "Solving tridiagonal systems on ensemble architectures", SIAM J. Scient. Stat. Comput. 8 (1987), 354-392.

[16] R.N. Kapur, and J.C. Browne, "Techniques for solving block tridiagonal systems on reconfigurable array computers", SIAM J. Scient. Stat. Comput. 5 (1984), 701-719.

[17] R.M. Karp and V. Ramachandran, "A survey of parallel algorithms for shared-memory machines", Tech. Rep. UCB/CSD 88/408, UC Berkeley, 1988.

[18] H.B. Keller, *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, 1968

[19] H.-O. Kreiss, N.K. Nichols, and D.L. Brown, "Numerical methods for stiff two-point boundary value problems", SIAM J. Num. Anal. 23 (1986), 325-368.

[20] M. Lentini, "Parallel solution of special large block tridiagonal systems: TPBVP", Manuscript, 1989.

[21] M. Lentini, "The potential for parallel algorithms in ordinary differential equations: boundary value problems", Tech. Rep. 5/88, Lab. Nacional de Computacao Cientifica - LNCC, Rio de Janeiro, 1988.

[22] M. Lentini, M.R. Osborne and R.D. Russell, "The close relationships between methods for solving two-point boundary value problems", SIAM J. Numer. Anal. 21 (1984)

[23] R.M.M. Mattheij, "Decoupling and Stability of Algorithms for Boundary Value Problems", SIAM Review (1985), 1-44.

[24] J. Nievergelt, "Parallel methods for integrating ordinary differential equations", Comm. ACM 7 (1964), 731-733.

[25] J.M. Ortega and R.G. Voight, "Solution of partial differential equations on vector and parallel computers", SIAM Rev. 27 (1985), 149-240.

[26] M.R. Osborne, "On shooting methods for boundary value problems", J. Math. Anal. Appl. 27 (1969),417-433.

[27] B.A. Schmidt, "An algebraic approximation to the matrix exponential in singularly perturbed boundary value problems", SIAM J. Numer. Anal., to appear.