ELSEVIER

Applied Numerical Mathematics 20 (1996) 39–55

APPLIED NUMERICAL MATHEMATICS

On the performance of parallel waveform relaxations for differential systems

Kevin Burrage [a,*], Carolyn Dyke [a], Bert Pohl [b]

[a] *Centre for Industrial and Applied Mathematics and Parallel Computing, Department of Mathematics, The University of Queensland, Queensland 4072, Australia*

[b] *Seminar für Angewandte Mathematik, ETH Zürich, Ch 8092 Zürich, Switzerland*

Abstract

In this paper, modifications to the distributed memory waveform relaxation package developed by Burrage and Pohl (1994), known as PWVODE, are considered. Various dynamic, adaptive and communication protocols are considered, and a number of numerical comparisons on a 96 node Intel Paragon are presented which show the efficacy of PWVODE.

1. Introduction

Consider solving systems of initial values problems (IVPs) of the form

$$y'(t) = f(t, y(t)), \quad f : [t_0, T] \times \mathbb{R}^m \to \mathbb{R}^m, \quad y(t_0) = y_0. \tag{1}$$

For the simulation of many problems of practical importance, such as very large scale integrated (VLSI) circuits, the dimension m of system (1) may be extremely large. Standard techniques have been shown to be adequate for solving modestly sized systems, but they may be inappropriate for systems consisting of many thousands of equations. As a consequence, substantial effort has been focused towards the development of numerical methods which are capable of solving very large systems of differential equations. With the advent of parallel technology, it is natural to desire to take advantage of the increased computing power available.

In this paper, parallel implementation of waveform relaxation techniques is considered. In particular, a parallel block Jacobi waveform relaxation code will be described, and implementational details as well as numerical results on a 96-processor Intel Paragon will be discussed. General waveform relaxation techniques are introduced in Section 2. Due to the possibility that the general waveform

* Corresponding author. E-mail: kb@maths.uq.oz.au.

SSDI 0168-9274(95)00116-6

relaxation techniques may be slow to converge, methods of increasing the rate of convergence are sought. The multisplitting technique of O'Leary and White [14] is discussed in Section 3. Implementation details of the package developed by Burrage and Pohl [5] are considered in Section 4. Modifications to the package are introduced in the context of a two-dimensional test example in Section 5. Finally, some conclusions are presented.

2. Waveform relaxation methods

The system presented in (1) is generally nonlinear, and extremely stiff. When considering the test problem presented in Section 5, there are often components of the solution which change rapidly, and others which vary only slowly. This forces the use of an implicit integration technique. Thus it is necessary to solve an m-dimensional system of nonlinear equations at each time level. A major disadvantage associated with solving stiff ordinary differential equations (ODEs) with initial values is that there is no natural parallelism across time exhibited. Fortunately, there is parallelism across the system. One of the simplest techniques to implement which exploits parallelism across the system is the Picard method. A sequence of iterative solutions $y^{(0)}(t), y^{(1)}(t), y^{(2)}(t), \ldots$, is generated over the region of interest satisfying the equation

$$y^{(k+1)'}(t) = f(t, y^{(k)}(t)), \qquad y^{(k+1)}(t_0) = y_0. \tag{2}$$

For a system of dimension m, this approach allows the decoupling of the problem into m independent parallel quadrature problems. Unfortunately, unless $||\partial f/\partial y||$ is small the iterations converge very slowly. One way to improve the rate of convergence is to use the concept of *time windows* where the interval of integration is divided into a series of windows. The iterative technique is applied to each window. Another method for improving the rate of convergence is to "split" the right-hand side. This leads to the shifted Picard method (see [16], for example), which takes the form

$$y^{(k+1)'}(t) - M_k y^{(k+1)}(t) = f(t, y^{(k)}(t)) - M_k y^{(k)}(t).$$

The difficulty lies in how to choose the window size and the splitting matrix M_k automatically.

Picard methods form a subset of a more general class of methods known as waveform relaxation techniques. Lelarasmee introduced waveform relaxation techniques in his Ph.D. Thesis which appeared in 1982 [11]. The essential idea is to segment the full system of m equations into smaller subsystems, which can then be solved independently by different processors on a parallel computer. A survey paper in the area of using waveform relaxation methods for the time domain analysis of large scale problems arising from the modelling of integrated circuits is given by White et al. [19]. The discussion in this present paper is due largely to Burrage and Pohl [5].

A simple form of the waveform relaxation technique is merely a natural generalisation of the Picard approach:

$$y^{(k+1)'}(t) = F(t, y^{(k+1)}(t), y^{(k)}(t)), \quad y^{(k+1)}(t_0) = y_0, \tag{3}$$

where the splitting function $F : [t_0, T] \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies

$$F(t, y, y) = f(t, y).$$

The splitting F is chosen in order to attempt to decouple the original system into independent subsystems, which may then be solved separately. For example, denoting the m components of the function f by $f_1, \ldots, f_m$, the scheme in which

$$F_i(t, y^{(k+1)}, y^{(k)}) = f_i(t, y_1^{(k+1)}, y_2^{(k+1)}, \ldots, y_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \quad i = 1, \ldots, m,$$

so that (3) becomes

$$y_i^{(k+1)'} = f_i(t, y_1^{(k+1)}, \ldots, y_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \tag{4}$$

is known as *Gauss–Seidel waveform relaxation,* due to its obvious resemblance to the Gauss–Seidel scheme. The task of solving a differential equation in m variables has been transformed into finding the solution to a sequence of differential equations in a single variable. Obviously this process is typically sequential, whereas the Jacobi waveform relaxation scheme

$$y_i^{(k+1)'} = f_i(t, y_1^{(k)}, y_2^{(k)}, \ldots, y_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \tag{5}$$

is inherently parallel. Using a more complicated form for (3), a continuous-time waveform relaxation method is given by

$$\begin{aligned} &z^{(k+1)'}(t) = G(t, z^{(k+1)}(t), y^{(k+1)}(t), y^{(k)}(t)), \\ &z^{(k+1)}(t_0) = y_0, \quad y^{(0)}(t_0) = y_0, \\ &y^{(k+1)}(t) = g(t, z^{(k+1)}(t), y^{(k)}(t)), \end{aligned} \tag{6}$$

where $G : [t_0, T] \times \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ and $g : [t_0, T] \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ satisfy

$$G(t, y, y, y) = f(t, y), \quad g(t, y, y) = y.$$

Then the waveform relaxation SOR method is

$$\begin{aligned} &G_i(t, z^{(k+1)}, y^{(k+1)}, y^{(k)}) = f_i(t, y_1^{(k+1)}, \ldots, y_{i-1}^{(k+1)}, z_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \\ &g_i(t, z^{(k+1)}, y^{(k)}) = \omega z_i^{(k+1)} + (1 - \omega) y_i^{(k)}, \quad i = 1, \ldots, m, \end{aligned} \tag{7}$$

where $\omega \in (0, 2)$ is a relaxation parameter.

Theoretical analyses of waveform relaxations methods have been presented by Miekkala and Nevanlinna [12], Nevanlinna [13], and Vandewalle [17], as well as others. Only a brief guide to the considerations for convergence are presented here.

The convergence of the functions $y^{(k)}(t)$ is considered where the formulation for the solution is (3). Let $\|\cdot\|$ be any fixed norm on $\mathbb{R}^m$. Then given appropriate Lipschitz conditions with Lipschitz constant L on F, assuming that all necessary derivatives exist, and denoting the maximum norm of a function $w : [\alpha, \beta] \to \mathbb{R}^m$ by

$$\|w\|_\infty = \max_{t \in [\alpha, \beta]} \|w(t)\|,$$

it can be shown that the iterates $y^{(k)}$ converge uniformly to the solution y of (1) on $[\alpha, \beta] = [t_0, T]$ such that

$$\|y^{(k)} - y\|_\infty \leqslant \frac{L^k (T - t_0)^k}{k!} \|y^{(0)} - y\|_\infty. \tag{8}$$

Thus, if the window of integration is long, or L is large, convergence is very slow.

When considering VLSI problems, for example, it is natural for an extremely large system of ODEs to result. In addition, as has been stated earlier, the system is nonlinear and very stiff. The general class of waveform relaxation methods discussed so far involve decomposing the full system into a number of smaller subsystems, which can be solved independently by different machines in a network, or different processors on a parallel machine. This type of approach means that subsystems involving rapidly changing components may be integrated with smaller stepsizes than those containing slowly changing components. A major obstacle which has been identified (Carlin and Vachoux [6]) is that the convergence of the iteration is slow when there is strong coupling amongst the subsystems.

3. Linear problems and multisplitting waveform relaxation

Consider the linear problem

$$y'(t) = Qy(t) + g(t), \quad y(0) = y_0. \tag{9}$$

The waveform relaxation algorithm for problems of this type is based on a splitting of the matrix Q into $Q = N - M$. Then

$$y'(t) + My(t) = Ny(t) + g(t), \quad y(0) = y_0,$$

and this is written iteratively as

$$y^{(k+1)'}(t) + My^{(k+1)}(t) = Ny^{(k)}(t) + g(t), \quad y^{(k+1)}(0) = y_0.$$

The approach taken here is a generalisation of the work by O'Leary and White [14] for algebraic systems of equations. The method which they presented involved splitting the matrix Q a number of times into $Q = N_l - M_l$ for $l = 1, 2, \ldots, S$. Jeltsch and Pohl [10] adapted that approach to systems of ODEs and allowed overlapping of components also.

Definition 3.1. A *multisplitting* of Q is a collection of matrices M_l, N_l, and $E_l \in \mathbb{R}^{m\times m}$, $l = 1, 2, \ldots, S$, such that

$$Q = N_l - M_l,$$

and each E_l is a nonnegative diagonal matrix satisfying the consistency condition

$$\sum_{l=1}^{S} E_l = I_m.$$

Using the multisplitting idea, (9) may be rewritten as

$$y_l'(t) + M_l y(t) = N_l y(t) + g(t), \quad y(0) = y_0, \quad l = 1, \ldots, S. \tag{10}$$

Again this may be solved in an iterative manner:

$$y_l^{(k+1)'}(t) + M_l y_l^{(k+1)}(t) = N_l y_l^{(k)}(t) + g(t), \quad y_l^{(k+1)}(0) = y_0, \tag{11}$$

for any $l \in \{1, \ldots, S\}$, in which case

$$y^{(k+1)}(t) = \sum_{l=1}^{S} E_l y_l^{(k+1)}.$$

The convergence of (11) can be analysed in much the same way as that which leads to the convergence results given in (8). Let

$$k_l(t) = \exp(-tM_l)N_l, \quad l = 1, \ldots, S, \tag{12}$$

and assume a finite window. Then, with

$$||u||_\infty = \max_{t \in [0,T]} ||u(t)||,$$

and

$$||k_l||_\infty \leqslant C_l, \quad l = 1, \ldots, S, \tag{13}$$

it is possible to write (11) in fixed point form

$$y^{(k+1)}(t) = Ky^{(k)}(t) + \phi(t),$$

where

$$||K^k||_\infty \leqslant \frac{(CT)^k}{k!}, \tag{14}$$

and $C = \max\{C_l\}$. Thus (11) is a contraction mapping on the interval $[0, 1/C]$, and will converge on any finite window. Unfortunately, for stiff problems, C will be large and (14) implies convergence on small windows, although this bound may be unduly pessimistic. Recently, Burrage et al. [4,3], have considered new waveform techniques which can be accelerated by the process of preconditioning on the left or right. This process is completely analogous to those techniques used in the static case for linear systems of equations of the form $Qy = b$.

4. Adaptive waveform algorithms

In the previous section, some convergence properties for waveform relaxation techniques were given. In order to obtain some understanding of how waveform convergence behaves as a function of the window size for nonlinear problems, a block step Jacobi waveform relation method is applied to a two-dimensional reaction–diffusion equation based on the interaction between two chemical species. The problem considered is large and structured, which allows the code to exploit the parallelism associated with the system. It is assumed that the problem has been decomposed into a number of subproblems by some form of waveform relaxation. After the problem has been decomposed, it is necessary to solve systems of ODEs within each subproblem. Thus efficient techniques for integration are required. Rather than formulating new integration codes, the code developed by Burrage and Pohl [5], which will henceforth be known as PWVODE, exploits existing sequential packages that are known to be efficient and robust. Two such packages are VODE and VODEPK [1], which are available from NETLIB.

Because an existing sequential package can be used in SPMD (single program multiple data) mode, with each processor running the same form of VODE or VODEPK, most of the parallel implementation of the waveform algorithm involves communication issues. Here standard message-passing environments such as $p4$ and *PVM* can be used. An important advantage in using either of these environments is that they can be used to combine a network of workstations as one single computational resource. This makes code generation and debugging a much more efficient task, and indeed the waveform code presented in this paper was developed and tested in a distributed environment of SPARC2 workstations at the Department of Mathematics in the University of Queensland using the software environment $p4$. However, in order to gain some uniformity and portability across architectures, the code is currently being rewritten using the MPI interface.

VODE has a user interface which is almost identical to LSODE [9] but improves on the efficiency of problems which require frequent and large changes in stepsize. In the solution of the nonlinear systems for the update point, VODE offers a choice of either functional iteration or modified Newton iteration in which the Jacobian is either user supplied or computed internally. VODE also caters for full or banded problems. Since most of the computational work in any differential equation solver involves the solution of the nonlinear equations defining the update, VODE uses a number of techniques to reduce this work. These include reusing the LU factors of the amplification matrix until convergence properties deteriorate, as well as accelerating the convergence of the iterations within the Newton step by relaxing the amplification matrix based on estimates of the extreme eigenvalues of the problem. At the end of each step within VODE, the local error is estimated and a stepsize change is considered for the current step or subsequent steps depending on the magnitude of the error.

The code uses direct linear algebra techniques for solving the linear systems associated with implicit methods. Recent advances in iterative techniques for linear systems based on preconditioned Krylov GMRES methods [15] have meant that these approaches can now be incorporated into ODE solvers and this has been done within VODEPK. VODEPK has a very similar structure to VODE except that a preconditioned GMRES technique based on a scaled preconditioned incomplete version of GMRES (SPIGMR) is used for the linear solver.

In the tests presented in this paper, VODE is used as the core solver in PWVODE.

4.1. PWVODE code

The original version of PWVODE involves a large number of control features aimed at improving the rate of convergence of the technique. The waveform algorithm itself is based on a block Jacobi multisplitting technique. The VODE package is used to solve the linearised system of ODEs. Since, in general, little is known about the problem to be solved, Burrage and Pohl [5] split it more or less evenly among the available processors. This achieves a rudimentary load balancing. If S is not a factor of m, that is, $m = \alpha S + \beta$ with $\beta \neq 0$, then one element is added to each of the subsets $S_{S-\beta+1}, \ldots, S_S$.

In the case of overlap, this strategy has to be modified slightly, depending on whether there is a one sided or two sided overlap. Illustrated in Fig. 1 is the formation of the subsystems when $m = 18$, $S = 5$, and the overlap varies from an overlap of zero to an overlap of two. It is seen that, in this case, the two sided overlap gives a slight ill balancing, but the load balancing strategy is easily modified here.

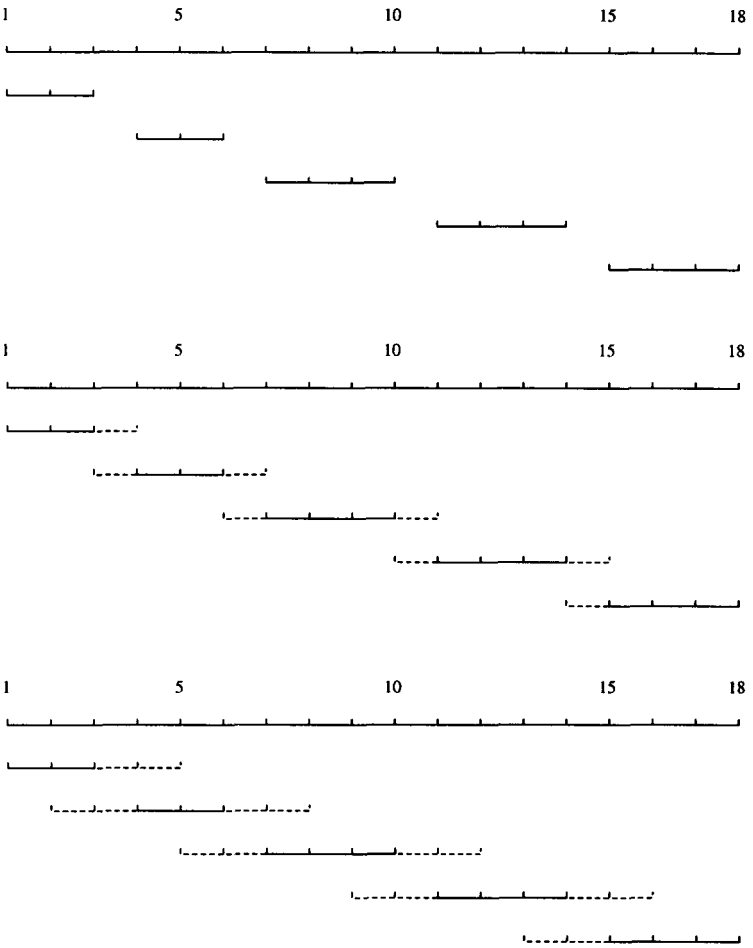

Fig. 1. Overlaps 0, 1, and 2 for $m = 18$, $S = 5$.

Within the confines of PWVODE, Burrage and Pohl [5] only allow the step setting for the weighting matrices E_l, which means that overlapped components are computed within a block but they are not given any weighting when formulating the solution after each iteration. The initial choice for the problems considered in Section 5 involves the step setting of the matrix. In this case, the weighting that is attributed to the overlapping is always the weighting that would be given in the nonoverlapping case. Thus, when $m = 18$, $S = 5$, and the overlap is 2, only components 1, 2, and 3 are given a full weighting in the first subsystem. The amount and type of overlap may be chosen by the user. This implementation is a static one, although more recently Burrage [2] has considered with Pohl a dynamic implementation, in which adaptive load balancing is performed by continuously varying the overlap (see Section 5.5).

Initially, an attempt is made to solve the problem given in (15) below on the interval of integration $[t_0, T]$ using one time window. The default settings for the problem considered in Section 5 are

$t_0 = 0$, and $T = 6$. If, within the call to VODE, the number of steps needed to go to a certain distance exceeds some default, the window is rescaled automatically. There is also provision for more than one time window to be chosen. Burrage [2] has chosen differing numbers of time windows in order to study the effect on execution time. Within PWVODE, there is a default setting which gives the number of output points in a given window. This is set to 40. An option in VODE is used to calculate an approximation to the solution at exactly these points. The waveform for that sweep is then calculated by a piecewise linear approximation.

Care must be taken in choosing the convergence criteria for the problem. In fact there are two criteria: the waveform tolerance, and the tolerance that VODE uses. The waveform tolerance (ϵ_{wr}) is used to determine whether successive waveforms are sufficiently close. The maximum absolute difference between successive waveforms over all components and all time levels is computed. The iterates are deemed to have converged once this difference is less than ϵ_{wr}. The VODE tolerance ϵ_{vo} controls the local error and the choice of stepsize. It is chosen as a mixture of absolute and relative errors.

Included in PWVODE is a number of choices relating to the problem to be solved, the starting function to use, the type of splitting to implement, how often to store the solution, as well as various communication protocols which exploit the structure of the problem.

5. A case study

In order to test the performance of the waveform code, only large structured problems have been considered. This will allow the code to exploit the parallelism associated with the problem. To evaluate the behaviour of a number of proposed acceleration techniques, the test problem considered is a reaction–diffusion equation known as the diffusion Brusselator equation, defined on the unit square. It is described in [8] and has the form

$$\begin{aligned}
\frac{\partial u}{\partial t} &= B + u^2 v - (A+1)u + \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \\
\frac{\partial v}{\partial t} &= Au - u^2 v + \alpha\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right).
\end{aligned} \tag{15}$$

The choice of initial values in this problem has a marked effect on the rate of convergence of the iterates. Thus two sets of initial conditions were considered. The "easy" initial conditions are

$$\begin{aligned}
&u(x,y,0) = 2 + \mu_1 y, \quad v(x,y,0) = 1 + \mu_2 x, \\
&A = 3.4, \quad B = 1, \quad \alpha = 0.002, \quad \mu_1 = 0.25, \quad \mu_2 = 0.8,
\end{aligned} \tag{16}$$

and the "hard" initial conditions are

$$\begin{aligned}
&u(x,y,0) = 0.5 + \mu_1 y, \quad v(x,y,0) = 1 + \mu_2 x, \\
&A = 3.4, \quad B = 1, \quad \alpha = 0.002, \quad \mu_1 = 1, \quad \mu_2 = 5.
\end{aligned} \tag{17}$$

In each case, Neumann boundary conditions are imposed. Here u and v represent the concentrations of each of the products of the reaction. The concentrations of the input reagents are denoted by A

and B, and are assumed to be constant. The constant $\alpha = d/L^2$, where L is a reactor length and d is a diffusion coefficient.

The method of lines is used to convert the problem into a system of ordinary differential equations. This means that the second order spatial derivatives are replaced by centralised finite differences on a uniform array of $N \times N$ points. If the subintervals in each of the directions have length $1/(N+1)$ then (15) is replaced by a system of coupled nonlinear equations of order $m = 2N^2$. These equations may be shown to be of the form

$$\begin{aligned} u'_{ij} &= B + u_{ij}^2 v_{ij} - (A+1)u_{ij} + \alpha(N+1)^2(u_{i-1,j} + u_{i+1,j} - 4u_{ij} + u_{i,j-1} + u_{i,j+1}), \\ v'_{ij} &= Au_{ij} - u_{ij}^2 v_{ij} + \alpha(N+1)^2(v_{i-1,j} + v_{i+1,j} - 4v_{ij} + v_{i,j-1} + v_{i,j+1}). \end{aligned} \tag{18}$$

5.1. Ordering

Burrage and Pohl [5] observed that there are two natural ways of ordering the components of the system (18). The first of these is

$$u_{11}, \ldots, u_{1N}, u_{21}, \ldots, u_{2N}, \ldots, u_{NN}, v_{11}, \ldots, v_{1N}, v_{21}, \ldots, v_{2N}, \ldots, v_{NN}, \tag{19}$$

and the second

$$u_{11}, v_{11}, u_{12}, v_{12}, \ldots, u_{NN}, v_{NN}. \tag{20}$$

If the first ordering is used, the Jacobian of the problem has the structure

$$\begin{bmatrix} T_1 & D_1 \\ D_2 & T_2 \end{bmatrix},$$

where each of the matrices is of dimension N^2, and D_1 and D_2 are diagonal. For the second ordering, the Jacobian is banded with a half bandwidth of $2N$. Ignoring the effect of the ordering on the convergence rate of the waveforms, and considering only the storage requirements of each ordering, it is apparent that the second choice is far superior. The bandwidth is a factor of $N/2$ smaller, which yields a substantial saving in storage, and also the linear algebra requirement. In the case of a parallel implementation however, the situation is more complicated.

Using the ordering in (19), if the number of subsystems S satisfies

$$\frac{2N^2}{3} \geqslant S \geqslant 2N,$$

then a block Jacobi approach leads to the solution of a tridiagonal problem on each processor. If $S < 2N$, then portions of the upper diagonal and lower diagonal blocks of T_1 and T_2 must be included into each subsystem. On the other hand, for the ordering in (20), if S satisfies

$$\frac{2N^2}{5} \geqslant S \geqslant N,$$

then a block Jacobi approach leads to the solution of a banded problem with bandwidth 5 on each system. It is obvious that, in a parallel environment, the first ordering appears more appropriate if $S \geqslant 2N$, while the second ordering is preferable if $S < 2N$.

Table 1
Overlap and communication times

overlap	0	1	2	3	4	5
local	211.8	207.7	207.7	208.7	211.7	260.7
master–slave	470.1	463.9	458.7	462.9	462.7	508.7

5.2. Communication

This analysis, however, is still incomplete because it does not cover overlapping or communication issues. In the case of a dense problem, a waveform algorithm requires communication between all subsystems to update the waveform at the end of each iterate and to compute the input for the next step. Rather than perform multibroadcasts (with possible risk of deadlock) a master–slave model can be used. In this model, the master collects and sends at the end of each waveform iteration all the necessary information in order to proceed with the next iteration. This involves each node sending the computed waveform for its subsystem as well as error diagnostics to the host at the end of each iteration. This option is used as the default option if the problem is dense or if the user does not wish to exploit any structure within the problem. In this model, it does not matter whether there is overlap or no overlap. Nor does it particularly matter what subset of processors are allocated to a task or what the connection topology is.

If the first ordering is used, it is not possible to write the problem in a block-tridiagonal form to allow for local communication in an efficient manner, whereas, in the case of the second ordering, this is possible and hence local communication can be exploited. This means that when the waveform is to be communicated to the relevant processors for the next sweep only the processors to the left and right of a processor need communicate their results to that processor. Burrage [2] solves (18) on 64 processors of an Intel Paragon using both master–slave and local communication, for various overlaps. The problem dimension is $m = 5000$, and the interval of integration is $t = [0, 6]$. The second ordering of components is chosen. The executions times (in seconds) are given in Table 1.

These results show that a master–slave implementation is at least twice as slow as a local communication implementation which exploits the tridiagonal structure of the problem. Although it is claimed that long messages can be sent by wormhole routing efficiently in a global manner, local communication is clearly important if locality of data can be guaranteed.

Despite the previous comments on ordering, it has been found (see [2]) that the second ordering is much more efficient, in that the number of iterations needed to achieve satisfactory convergence is considerably less. The reason for this is that there is a natural coupling between the u_{ij} and v_{ij} which the second ordering exploits, while the first ordering breaks this coupling by putting corresponding elements u_{ij} and v_{ij} in separate blocks far removed from one another. However, the partitioning of the system for the second ordering must always be such that there is no splitting between a u_{ij} and a v_{ij} component. Thus, in this paper, numerical results are based on the second ordering of the components.

The results of the first tests performed may be found in [5], but they will be described briefly below in order to introduce the subsequent modifications. Initially, a solution to the problem was sought using the ordering given in (19). In [5], the effect differing overlaps have on the efficiency of the code, as well as the significance of using $p4$ and the Intel message-passing routines, have been explored. Essentially they observed some slight improvement when using Intel message-passing

constructs, and that it can be important to overlap components. Obviously, the difficulty is knowing *a priori* how to select an overlap. If the overlap is too large, the size of each subsystem increases, and the associated linear algebra and memory costs may override any advantage gained due to the decrease in number of iterations required. However, in Section 5.6, an adaptive technique for dynamic load balancing by controlling the overlap, which was developed by Burrage [2] in conjunction with Pohl, is discussed briefly.

5.3. Adapting the VODE tolerances

The default settings for PWVODE implement VODE with the tolerance set at 10^{-8}. Furthermore, the initial guess is merely a matrix of constants. Therefore it is highly unlikely that the first few iterates will be particularly accurate, and thus it seems that demanding the VODE tolerance to be 10^{-8} is a little unrealistic. A feature of PWVODE is the ability to change the VODE tolerances so that initially they are relaxed and then progressively tightened. The user can thus specify a sequence of VODE tolerances, and a corresponding sequence of iteration numbers which describes for how many iterations a given VODE tolerance is imposed. Such a modification generally results in a decrease in the total execution time by at least a factor of 2.

5.4. Adapting the window size

As has been observed, a feature of the waveform approach is that as t increases towards T, the waveforms are slower to converge, hence the need for smaller time windows. If n windows are specified by the user, then each window is of length $(T - t_0)/n$. Although this strategy does enable some control over the length of each window, the control does not take into account the development of the waveforms. If it is expected that convergence deteriorates towards the end of the window of length $T - t_0$, then it may be ideal to use a nonuniform division of the windows (see [7]). A major problem with trying to use a nonuniform approach is that, *a priori*, it will not be known a priori the best place in which to segment the window. Thus the second modification considered involves endeavouring to automate the choice of the number of time windows and their length. Initially the integration over a window would be interrupted only if the integrator (VODE) stalled at some time level. In order to satisfy convergence requirements, the difference between successive approximations is calculated and, once this difference is less than ϵ_{wr}, the waveforms are deemed to have converged. A calculation similar to this may be exploited when trying to decide on the rescaling of a window.

If $y^{(i)}(c,t)$ represents an approximation to the solution component c of y at time t during iteration i, and assuming that an approximation to the solution has already been calculated for iterations $i-2$ and $i-1$, then the behaviour of the ratio

$$\frac{||y^{(i)}(c,t) - y^{(i-1)}(c,t)||_2}{||y^{(i-1)}(c,t) - y^{(i-2)}(c,t)||_2} \tag{21}$$

may be monitored. If over k successive time levels, $t, t+1, \ldots, t+k-1$, this ratio is greater than some convergence tolerance (tol) specified, in advance, by the user, then the window is rescaled from $[t_0, T]$ to $[t_0, t]$. If there are p time levels in the original window, and the problem occurs at the kth time level, the new window created from $[t_0, t]$ will contain k time levels, leaving the second window from $[t, T]$ with $p - k$ time levels.

The window is rescaled by interrupting the integration in other subsystems. As soon as a troublesome time level t is encountered, integration on the processor involved ceases. The point t is sent to the master. Then the master sends emergency interrupt messages to all other subsystems. Once a subsystem has received the point t, there are two options available to it. Firstly, if the time level t has not already been attained, then the integration proceeds until t is reached, although a limit is placed on the number of steps that will be taken in attempting to reach a specified time level. Secondly, if t has already been reached, then integration stops immediately. Meanwhile, the master is calculating the new number of time levels required in each window, and rescaling the window to the new endpoint. Integration then commences on each subsystem for the new window $[t_0, t]$, using the last iterate as the starting solution. In some applications, it is possible that this resizing process will be necessary more than once. If this is the case, suppose that the window has already been rescaled to the interval $[t_0, t_1]$ and that further rescaling to $[t_0, t_2]$ is necessary, where $t_2 < t_1$. Then, on reaching t_2, the new window is set to $[t_2, t_1]$. This represents a global approach to windowing, in which the largest possible window size is chosen initially, and is only reduced if appropriate.

As in [5], the modifications were tested on a cluster of SPARC2 workstations before being ported to the Intel 96 node Paragon in Zürich.

5.5. Results

The results in this section were obtained on the 96 node Intel Paragon located at ETH Zürich. Each node consists of two, 50 MHz, i860XP processors, with 32 Mbytes of memory and a 16 Kbyte cache. Only one processor per node is accessible for computation, and this processor has a peak rating of 75 Mflops.

The aims were to study the efficacy of adapting the VODE tolerances versus controlling the error by adaptively varying the window. Each particular problem was solved using the easy and hard initial conditions. The introduction of overlap was also considered. Using the results of Burrage [2], the ordering in (20) was chosen. Burrage found that this choice of ordering provides greater flexibility in the choice of problems to be solved, and is also, generally, more efficient. The experiments were designed so that various problems were solved on 32, 64 and all 96 processors. The number of equations assigned to each processor was increased in multiples of 2, from 16 to 64. Thus, in some cases, the same problem is solved on differing numbers of processors. The efficiency of the modifications is considered in terms of the total execution time of the problem. An internal timing function is introduced to make this possible. In Tables 5 and 6, execution times are in seconds. The column labeled "VODE" represents the execution time using the one window on $[0, 6]$, and the VODE tolerances fixed at 10^{-8}. "Adapt" describes the execution time once the VODE tolerances are tightened during iteration, and "Error" the execution time once the error is controlled.

To see the behaviour of PWVODE, VODE was run on a single processor on the problem sizes given in Table 3.

It can be seen from Tables 2 and 3 that, when imposing the easy initial conditions, as the problem size increases, it becomes more desirable to consider a parallel implementation.

When considering the parallel experiments undertaken, it becomes obvious that adapting the VODE tolerances is an extremely effective way of decreasing the execution time. For the easy initial conditions, it is also apparent that monitoring the ratio in (21) yields comparable results. The added attraction associated with monitoring the error is that no knowledge of the total number of iterations

Table 2
Timings (seconds) with easy initial conditions, and no overlap

Number of processors	Number of equations per processor								
	16			32			64		
	VODE	Adapt	Error	VODE	Adapt	Error	VODE	Adapt	Error
32	21	12	14	66	37	35	492	280	228
64	47	26	24	155	94	77	447	271	210
96	83	49	39	233	152	149	740	344	330

Table 3
Timings (seconds) for sequential VODE

	512	1058	2048	4050
Easy IC	11	34	108	345
Hard IC	39	121	367	1115

Table 4
Timings (seconds) with hard initial conditions, and no overlap

Number of processors	Number of equations per processor								
	16			32			64		
	VODE	Adapt	Error	VODE	Adapt	Error	VODE	Adapt	Error
32	56	26	38	155	73	95	1160	573	443
64	83	44	70	221	109	147	627	307	259
96	103	57	87	297	135	173	813	410	439

expected is required a priori. The problems are repeated using the hard initial conditions.

On comparing Tables 3 and 4, the advantages of a parallel approach become clear. For moderately sized problems of dimension 1024 or less, there is only a modest speed-up of PWVODE over VODE. As the dimension increases to 4096 and beyond, speed-ups increase to a factor of 5. It would be expected to see greater increases in speed-up as the dimension increases further. However, since, on the system that was available, it was not possible to use VODE on one processor for problems of dimension more than 6000 or so because of memory limitations, further speed-up results are not available.

For the hard initial conditions, when 16 equations are assigned to each processor, there is obviously an increase in execution time if error control is chosen. However, upon increasing the number of equations per processor, controlling the error becomes more efficient than the standard implementation, and eventually competes well with the adaptation approach.

The experiments are now repeated using a fixed overlap in each direction. After some preliminary tests, it is observed that, for the easy initial conditions, an overlap of 2 is desirable, while an overlap of 3 is preferable for the hard initial conditions.

From Tables 5 and 6, it may be seen that the conclusions made previously are applicable once the overlap is introduced. Also apparent is that, with one exception, the introduction of overlap has generated a further decrease in execution times. The only exception is where 16 processors are utilised, each with 64 equations to solve. In this case, there is an increase in execution time noted for both the easy and hard initial conditions.

Considering the results presented, the following conclusions can be made:

Table 5
Timings (seconds) with easy initial conditions, and overlap of 2

Number of processors	Number of equation per processor								
	16			32			64		
	VODE	Adapt	Error	VODE	Adapt	Error	VODE	Adapt	Error
32	21	10	14	63	31	34	556	286	250
64	44	23	26	132	78	80	305	179	190
96	70	32	44	204	104	122	470	244	285

Table 6
Timings (seconds) with hard initial conditions, and overlap of 3

Number of processors	Number of equations per processor								
	16			32			64		
	VODE	Adapt	Error	VODE	Adapt	Error	VODE	Adapt	Error
32	40	19	46	128	67	116	1235	606	658
64	64	33	81	173	85	145	498	245	265
96	76	40	98	241	124	203	779	383	446

- when applying either set of initial conditions, in general, either adapting the VODE tolerances, or adapting the windows is effective;
- when an overlap in each direction is introduced, there is generally a decrease in overall execution time, and there may also be a slight decrease in the number of iterations required;
- as the dimension of the problem is increased towards 5000, PWVODE can give speed-ups of about a factor of 5 over standard sequential VODE. This performance will further improve as the dimension is increased.

5.6. *Further work*

In spite of the apparent uniformity of the subsystems in (18), it has been observed that, through a study of the space–time diagram created by ParaGraph running on the Paragon on a run of PWVODE, there are still considerable load balancing difficulties (see Fig. 2). Thus a dynamic load balancing approach has been implemented by exploiting the use of overlap.

ParaGraph is an extremely useful software tool for analysing the parallel performance of a code. The results given in Fig. 2 show a concurrency profile for a 16-processor implementation of the code using, respectively, static load balancing or dynamic load balancing by controlling the overlap.

For the static load balancing, it appears that the full 16 processors are busy for approximately only 30% of the execution time and that, for example, for 20% of the execution time, only seven processors are fully busy. This suggests a poor load balancing. In the case of dynamic load balancing, however, the profile indicates that all 16 processors are busy for 60% of the time and there is an even distribution of busy time in general. This shows the success of dynamic load balancing by controlling the overlap. These results have been confirmed by analysing a space–time diagram created by ParaGraph over the entire execution of the program. Here dynamic load balancing was enabled for the first two iteration sweeps and then turned off. For the first two iterations, the space–time diagram shows that the processors are much busier than in later iterations. The static load balancing

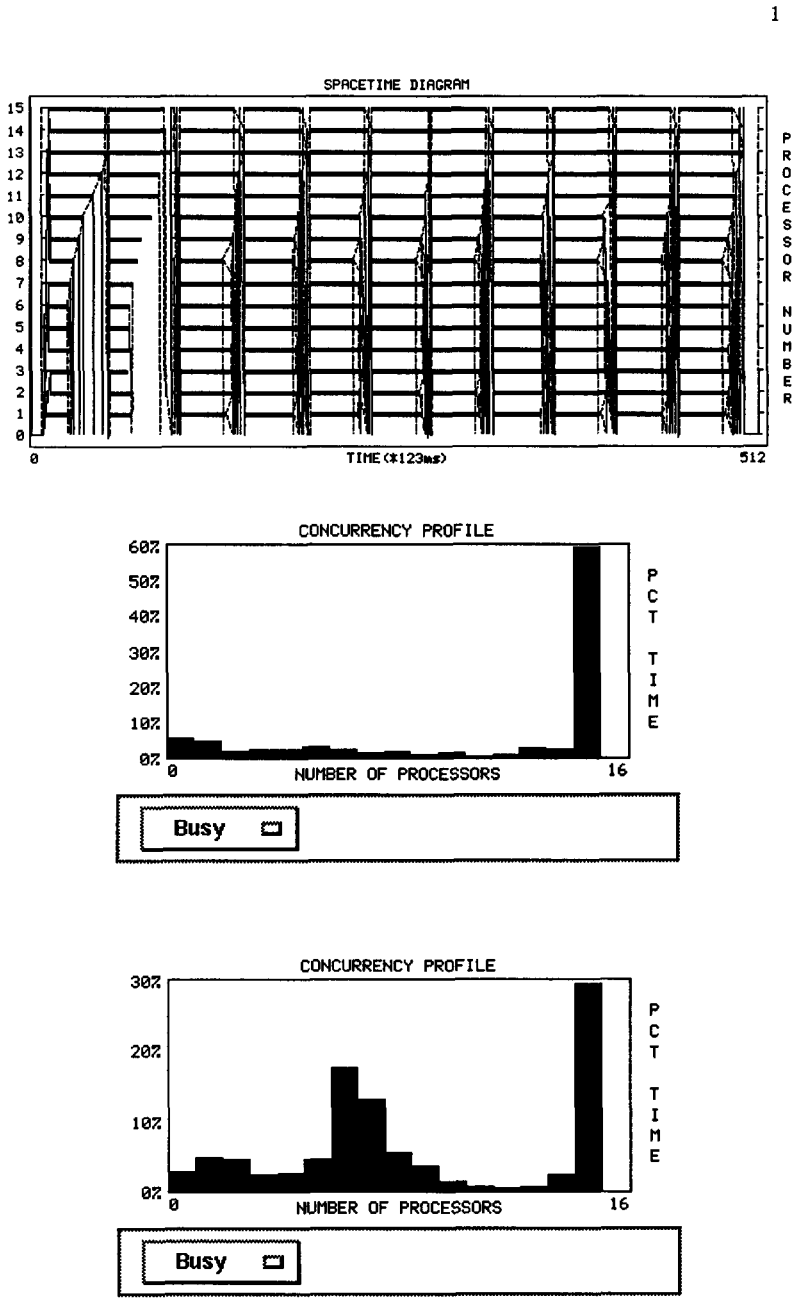

Fig. 2. Space–time diagram, concurrency profiles (dynamic and static)

assumes that each subsystem needs more or less the same amount of processor time. Thus, given p processors, $S = p - 1$ subsystems, a problem of dimension m and an overlap of θ, the dimensions of the S subsystems, for $m = qS + r$, are given by:

$$d(\theta) = \begin{cases} q + \theta & \text{for } l = 1, \\ q + \gamma * \theta & \text{for } l = 2, \ldots, S - r, \\ q + \gamma * \theta + 1 & \text{for } l = S - r + 1, \ldots, S - 1, \\ q + \theta + 1 & \text{for } l = S. \end{cases} \tag{22}$$

If overlap in both directions is used, $\gamma = 2$, while in the case of one sided overlap, $\gamma = 1$.

Dynamic load balancing works by assuming that initially the work load is distributed as above, with a small initial overlap, θ_0, of say two or three. After a few waveform sweeps, on a particular window, each processor, i, computes the average time, T_i, taken for one waveform sweep for its own subsystem. A native global routine is then used to compute the maximum ($T_{\max}$), minimum ($T_{\min}$) and average (T_{av}) of these times over all the processors. These numbers now reside on all the processors. Each processor now computes a load balancing factor

$$B_i = C \frac{T_{\text{av}} - T_i}{T_{\max} - T_{\min}}, \tag{23}$$

where C is a safety factor chosen to be 0.5 which mitigates against changing the dimensions of the subsystems by too much. This factor is now used to increase or decrease the dimension of the ith subsystem (see [2] for more details). Periodically these quantities are updated after a suitable number of sweeps.

6. Conclusions

The tests performed indicate that waveform relaxation can be effective for solving large systems of equations in a parallel environment. Various implementation strategies have been investigated to improve the behaviour of the waveform relaxation code presented here, including adapting the tolerances and adapting the window size. It appears that adapting the VODE tolerances is particularly effective, regardless of the initial conditions imposed. Another effective possibility is the adaptive error control facility.

Monitoring the error has the potential to be effective, and its attraction lies in the fact that no knowledge of the total number of iterations is required beforehand, and the rescaling process is totally automatic. As communication speeds continue to improve in parallel machines, it is expected that adapting the window will become more successful, as the extra communication overheads needed with this strategy will become less intrusive. It is also apparent that some form of dynamic load balancing through the use of variable overlap is desirable, since with a large number of processors it is undesirable to have them idle for large periods of time. In addition, the use of variable overlap in this way allows the code to cope better with slow convergence effects due to the strong coupling of components as well as any effects due to the nature of the initial conditions. Thus a combination of the three modifications presented here may prove to be the most effective, and will be investigated in further work.

Acknowledgment

The authors wish to thank the staff at ETH in Zürich for allowing access to the 96 node Intel Paragon.

References

[1] P.N. Brown, G.D. Byrne and A.C. Hindmarsh, VODE: a variable-coefficient ODE solver, *SIAM J. Sci. Statist. Comput.* 20 (1989) 1038–1051.

[2] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations* (Oxford University Press, Oxford, 1995).

[3] K. Burrage, Z. Jackiewicz, S.P. Norsett and R.A. Renaut, Preconditioning waveform relaxation iterations for differential systems, *BIT* (to appear).

[4] K. Burrage, Z. Jackiewicz and R.A. Renaut, Waveform relaxation techniques for pseudospectral methods, *BIT* (to appear).

[5] K. Burrage and B. Pohl, Implementing an ODE code on distributed memory computers, *Comput. Math. Appl.* 28 (1994) 235–252.

[6] C.H. Carlin and C. Vachoux, On partitioning for waveform relaxation time-domain analysis of VLSI circuits, in: Proceedings International Conference on Circuits and Systems, Montreal, Que. (1984).

[7] C.T. Dyke, The solution of differential equations in a parallel environment, Ph.D. Thesis, Mathematics Department, University of Queensland, Queensland (1995).

[8] E. Hairer, S.P. Norsett and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems* (Springer, New York, 1987).

[9] A. Hindmarsh, LSODE and LSODI, two new initial value ordinary differential equation solvers, *ACM SIGNUM News Lett.* 15 (1980) 10–11.

[10] R. Jeltsch and B. Pohl, Waveform relaxation with overlapping systems, Res. Rept. 91-02, Seminar für Angewandte Mathematik, ETH Zürich, Zürich (1991).

[11] E. Lelarasmee, The waveform relaxation method for the time domain analysis of large scale nonlinear systems, Ph.D. Thesis, University of California, Berkeley, CA (1982).

[12] U. Miekkala and O. Nevanlinna, Convergence of dynamic iterations for initial value problems, *SIAM J. Sci. Statist. Comput.* 8 (1987) 459–482.

[13] O. Nevanlinna, Remarks on Picard–Lindelöf iteration, *Numer. Math.* 57 (1989) 147–156.

[14] D.P. O'Leary and R.E. White, Multi-splittings of matrices and parallel solution of linear systems, *SIAM J. Algebraic Discrete Methods* 6 (1985) 630–640.

[15] Y. Saad and M. Schultz, GMRES: A generalised minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.

[16] R.D. Skeel, Waveform iterations and the Shifted Picard Splitting, *SIAM J. Sci. Statist. Comput.* 10 (1989) 756–776.

[17] S. Vandewalle, The parallel solution of parabolic partial differential equations by multigrid waveform relaxation methods, Department of Computer Science, Katholieke Universiteit Leuven, Leuven (1992).

[18] S. Vandewalle and R. Piessens, Numerical experiments with nonlinear multigrid waveform relaxation on a parallel processor, *Appl. Numer. Math.* 8 (1991) 149–161.

[19] R.E. White, A. Sangiovanni-Vincentelli, F. Odeh and A. Ruehli, Waveform relaxation: theory and practice, *Trans. Soc. Comput. Simulation* 2 (1987) 95–133.