Parallel methods for systems of ordinary differential equations

Kevin Burrage
Centre for Industrial and Applied Mathematics
and Parallel Computing
Department of Mathematics
The University of Queensland
Australia 4072
email: kb@maths.uq.oz.au

February 22, 1995

1 Introduction

Recently, considerable attention has been paid to development of efficient parallel algorithms for the numerical solution of ordinary differential equations (ODEs) of initial value type of the form;

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

To give an idea of the magnitude of some of the problems that have to be dealt with we mention the modeling of long-range transport of air pollutants in the atmosphere [14]. A relatively simple model generates a system of 267,264 ODEs which has to be solved over a long time scale in order to study seasonal variations in the pollutants. Clearly such problems cannot be solved in reasonable time without exploiting some concurrency.

In attempting to solve (1) three different types of parallelism can be identified:

(i) parallelism across the method,

(ii) parallelism across the system (space),

(iii) parallelism across time.

It is highly likely that efficient parallel algorithms may well take elements from all three of these categories, so that such algorithms will lie in a three dimensional space as indicated in Figure 1.

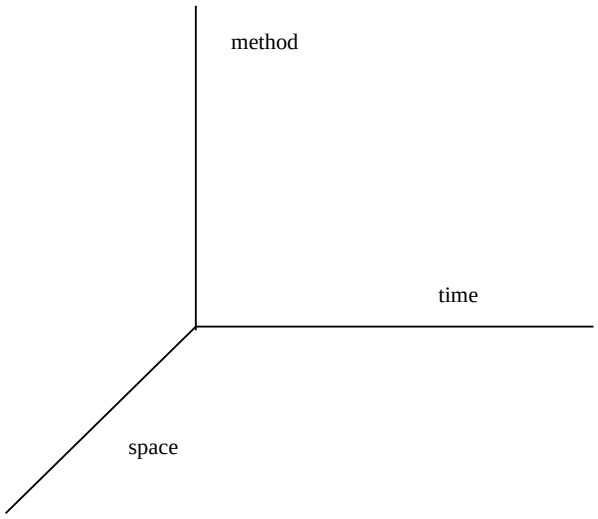

Figure 1: The parallelism space

In this article, two different algorithms, will be introduced and numerical results given to indicate the efficiency of these approaches. One algorithm is based on parallelism across the method and is suitable for implementation on a SIMD architecture (*i.e.*, the MasPar). The other algorithm is based on parallelism across the system and is suitable for implementation on a MIMD architecture (*i.e.*, the iPSC860).

2 Parallelism across the method

One way of exploiting parallelism across the method is to perform several function evaluations concurrently on different processors. This is possible with multistage methods such as Runge-Kutta. In general, there is little advantage in the direct approach although extrapolation techniques, with the work evenly balanced across the processors, are well-suited for parallel implementation when the problem size is large or function evaluations are costly [5]. However, indirect methods such as prediction-correction techniques can prove efficient in a parallel setting.

2.1 Prediction-correction

A popular technique for exploiting parallelism across the method [11] is based on the concept of a block method in which a block of values is predicted concurrently by some explicit method from a previous set of computed values, which

are then corrected a number of times by an implicit method using a fixed-point approach.

To illustrate this approach consider the block method in which a set of k values are updated concurrently at equidistant points $t_{n+1}, \ldots, t_{n+k}$ by using an explicit Euler predictor and then corrected twice by a trapezoidal corrector. This method can be computed in three steps

$$\begin{aligned} y_{n+j}^{(0)} &= y_n + jh\, f(t_n, y_n), \quad j = 1, \ldots, k \\ y_{n+j}^{(1)} &= y_n + \tfrac{h}{2} f(t_n, y_n) + \tfrac{h}{2} f(t_{n+j}, y_{n+j}^{(0)}), \quad j = 1, \ldots, k, \\ y_{n+j}^{(2)} &= y_n + \tfrac{h}{2} f(t_n, y_n) + \tfrac{h}{2} f(t_{n+j}, y_{n+j}^{(1)}), \quad j = 1, \ldots, k. \end{aligned} \tag{2}$$

Although this method is a very simple one it is illustrative of a much more general technique in which a block of k values, with components $y_1^{(n)}, \ldots, y_k^{(n)}$, are computed concurrently from step to step based on a Hermite predictor

$$Y^{(0)} = A_0 \otimes Y_n + hL_0 \otimes F(Y_n) \tag{3}$$

and an implicit corrector (with $Z_n = A_1 \otimes Y_n + hL_1 \otimes F(Y_n)$)

$$Y^{(j)} = Z_n + hL_2 \otimes F(Y^{(j-1)}), \quad j = 1, \ldots, r, \tag{4}$$

where $F(Y_n)$ denotes the vector with components $f(y_1^{(n)}), \ldots, f(y_k^{(n)})$.

However, in general, such methods can suffer from poor stability and/or large error coefficients unless a large number of corrections are performed [3]. As a simple rule, each time a correction of the form (4) is performed, the order of the method increases by one until the order of the corrector is reached [2]. Further corrections do not increase the order of the method but do smooth out successively higher and higher truncation coefficients in the local error expansion. However, for large block sizes the extrapolatory error in the predictor can be very large and it can take many corrections before acceptable accuracy is guaranteed. Nevertheless, the efficiency of this approach can be dramatically improved by the use of splitting techniques (which can be interpreted as a preconditioning) applied directly to the underlying corrector in (4).

This approach gives rise to a general iteration scheme of the form

$$M_{k,n} Y_{n+1}^{(k+1)} = (M_{k,n} - I) Y_{n+1}^{(k)} + Z_n + hL_2 \otimes F(Y_{n+1}^{(k)}), \quad k = 0, 1, \ldots \tag{5}$$

In the case that

$$M_{k,n} = I, \quad \forall k, n \tag{6}$$

(5) gives the standard prediction-correction approach which is just fixed-point iteration; while if

$$M_{k,n} = I - hL_2 \otimes J_n, \quad \forall k, \tag{7}$$

where J_n is the Jacobian of the problem evaluated at some point y_n, say, then (5) represents a modified Newton approach.

The $M_{k,n}$ can be chosen intermediate to the choices in (6) and (7) in an attempt to obtain both good convergence properties and cheap implementation in a parallel environment.

Defining

$$e_{n+1}^{(k+1)} = Y_{n+1}^{(k+1)} - Y_{n+1} \tag{8}$$

then a linearization of the problem gives

$$e_{n+1}^{(k+1)} = R_{k,n} e_{n+1}^{(k)}, \quad R_{k,n} = I - M_{k,n}^{-1}(I - hL_2 \otimes J_n). \tag{9}$$

Another way of viewing this is to apply the underlying corrector in (4) to the linear problem

$$y'(t) = J(t)y \tag{10}$$

which gives

$$PY^{(k)} = Z_n, \quad k = 1, \ldots, r, \quad P = I - hL_2 \otimes J_n. \tag{11}$$

Thus the choice of $M_{k,n}$ in (6) and (7) represents a preconditioning of the matrix P which will enable an acceleration of (5). If the eigenvalue structure of the underlying problem (1) is known (and this is often the case, for example, for problems arising from parabolic partial differential equations by the method of lines) then polynomial preconditioning is a well-known procedure for accelerating the convergence. For example,

$$\begin{array}{l} \text{TYPE I}: \quad M_k^{-1} = \alpha_k I \Rightarrow R_{k,n} = I - \alpha_k P \\ \text{TYPE II}: \quad M_k^{-1} = \alpha_k I - \beta_k P \Rightarrow R_{k,n} = I - \alpha_k P + \beta_k P^2. \end{array} \tag{12}$$

Suppose now that the eigenvalues of J_n are real and lie in the interval $[-q, 0]$, $q > 0$, the rate of convergence over p iterations can be maximized by minimizing $\rho\left((\prod_{k=1}^{p} R_{p-k,n})^{\frac{1}{p}}\right)$ where $\rho(H)$ denotes the spectral radius of H.

Some analysis using Chebyshev polynomials [4] gives that the spectral radii of the amplification matrices, as functions of $z = hq$, are minimized with

$$\begin{array}{l} \alpha = \frac{2}{1+v}, \quad \rho(R) = 1 - \alpha \\ \alpha = \frac{8(1+v)}{1+6v+v^2}, \quad \beta = \frac{\alpha}{1+v}, \quad \rho(R) = 1 - \alpha + \beta \\ \alpha_k = \frac{4}{4v + s_k^2(1-v)^2}, \quad \beta_k = \frac{\alpha_k}{1+v}, \quad s_k^2 = \sin^2\left(\frac{(2k-1)\pi}{4p}\right), \quad k = 1, \ldots, p. \end{array} \tag{13}$$

Here $v = det(I + zL_2)$, which in the case of the trapezoidal rule is $1 + z/2$.

The advantage of this approach is that the implementation properties are similar to explicit methods but the stability properties are similar to A-stable

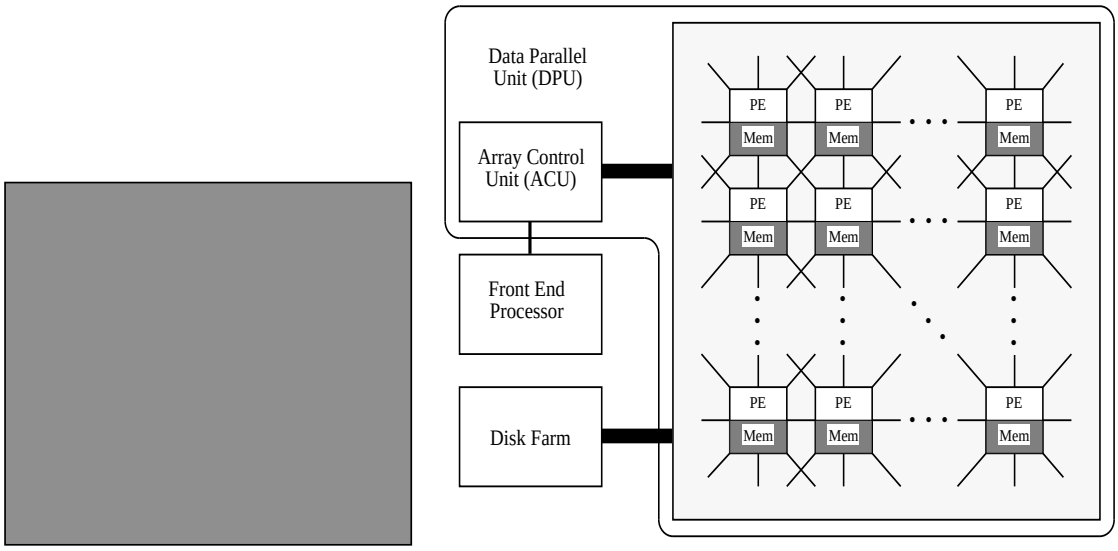

Figure 2: The computational space and network topology

implicit methods. The computational savings arise from the fact that the corrections can be calculated from simple matrix-vector operations. No solutions of linear systems are required which can be difficult to program efficiently in a parallel environment. This approach is particularly appropriate for so called Single Instruction Multiple Data (SIMD) computers such as the MasPar and has been programmed on a 4K processor MasPar MP1 at the University of Queensland.

2.2 The MasPar

An array computer is a large collection of processing elements (PEs) arranged in a mesh topology or some close derivative. Typically the PEs operate in a synchronized fashion with all the PEs performing the same instruction in lock-step but on different data.

Since many problems in modeling arising in fluid mechanics, stress analysis and spatial modeling can easily be approximated by a spatial discretization mesh, there is a natural processor topology which allows automatic parallelization by the use of Fortran 90 constructs, in particular BLAS routines (see below).

This MP1 consists of a front-end Unix workstation which performs the serial part of the computation and a back-end Data Parallel Unit (DPU). The DPU

consists of an Array Control Unit (ACU) and an array of processing elements (PEs). The ACU handles all the scalar variables program code. The processing element consists of a 4-bit processor, 64 Kbytes of memory, and generates a sustainable performance around 50 Kflops per PE.

Communication is through the XNET, which is a lock-step inter-processor communication protocol through the eight nearest neighbors. Alternatively, there is a global router which allows for arbitrary processor to processor communication via a three stage switch router.

The preconditioning approach described in section 2.1 has been coded in MPFortran, which is based on the new Fortran 90 standard. Since the only operations that are required here are BLAS-type operations, full advantage is taken of the parallelization by the Fortran compiler which automatically partitions arrays and vectors on the DPU. Comments and numerical results are given in section 4.

3 Parallelism across the system

Perhaps the simplest way to exploit parallelism across the system is through the concept of Picard iteration and, more generally, iteration in the function space. Picard's method for obtaining global approximations to the solution of (1) is based on solving a sequence of functional iterations of the form

$$y^{(k+1)'}(t) = f(t, y^{(k)}(t)), \quad y^{(k+1)}(t_0) = y^{(k)}(t_0). \tag{14}$$

In this case at each iteration level the problem can be split into m independent quadrature problems and this approach appears to be an appropriate one for obtaining massive parallelism. Unfortunately, the convergence of the iterates $\{y^{(k)}(t)\}$ to $y(t)$ is very slow. For example, for the standard linear test problem

$$y' = \lambda y, \quad t \in [0, T], \quad \lambda < 0 \tag{15}$$

it can be shown that the iterates $y^{(k)}(t)$ satisfy the following global error bound

$$\left| y(t) - y^{(k)}(t) \right| \leq \frac{(\mid \lambda \mid t)^{k+1}}{(k+1)!} \quad t \in [0, T] \tag{16}$$

so that there is no convergence until $k \geq \mid \lambda \mid T$. Thus one way to improve the convergence is by the technique of windowing in which the region of integration is split up into a series of windows and the iterative process then takes place on each window.

The first extensive study of more general functional iteration schemes occurred in the Electrical Engineering group at Berkeley University in the early 1980's [9] [13] and was given the name waveform relaxation. This technique allows standard iteration schemes for solving linear systems to be applied directly to the differential system to create a sequence of differential systems which

converge to the solution of (1), each of which can be solved by some discrete method. Thus Jacobi and Gauss-Seidel waveforms are described, respectively, as

$$y_i^{(k+1)'}(t) = f_i(y_1^{(k)}, \ldots, y_{i-1}^{(k)}, y_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \quad i = 1, \ldots, m \qquad (17)$$

and

$$y_i^{(k+1)'}(t) = f_i(y_1^{(k+1)}, \ldots, y_i^{(k+1)}, y_{i+1}^{(k)}, \ldots, y_m^{(k)}), \quad i = 1, \ldots, m \qquad (18)$$

It is easy to prove analogous results to (16) in the nonlinear case but again this implies that convergence can be very slow. Nevertheless, for certain classes of problems, such as those studied by the Berkeley group based on integrated circuit design, waveform relaxation techniques can work very well indeed. This is because the physicality of the model suggests how the components can be grouped together in tightly coupled subsystems, whose coupling occurs only over very short time intervals.

But in general, there are difficulties in knowing how both to group the components together and reorder the equations which are crucial to the efficiency of waveform relaxation [7]. It was noted that there can be a slow convergence of the iterations in the case of strong coupling between subsystems.

Recently, multigrid acceleration techniques have been applied directly to linear problems arising from the solution of linear parabolic differential equations by the method of lines. It has been shown [10] that these multigrid techniques can dramatically accelerate the convergence behavior of such iteration schemes. This work has been extended in [12] to nonlinear problems.

3.1 Distributed computing

The waveform approach is a suitable one for implementation in a distributed environment since it allows a decoupling of the original problem into subproblems which can then be solved more or less independently of one another on different processors (this of course depends on the nature of the coupling of the components in the original problem). This approach allows the programmer to take existing sequential codes which are known to be efficient and robust in a sequential environment and to apply them to the set of subproblems. One such code is VODE [1], which is based on Adams and BDF methods and is suitable for both stiff and nonstiff problems.

As a consequence of this, the programmer now only has to focus on the communication protocols and these can be programmed in some generic message-passing environment such as PVM (**P**arallel **V**irtual **M**achine) [8] or P4 (**p**ortable **p**rograms for **p**arallel **p**rocessors) [6]. These software environments are suited

for Fortran77 or C programs that consist of subtasks that offer a large granularity of parallelism and are based on the message passing model, allowing message transmission, barrier synchronization and broadcast.

An essential difference between PVM and P4 is that PVM uses a *pvmd* daemon to control the status of the processes, whereas P4 does not, so that P4 communication on distributed memory machines is done by message passing.

A code has been implemented which uses VODE as the basic integrator and P4 as the message-passing "glue." Numerical results are presented in the next section. The advantage of P4 is the code can be debugged and tested on a network of workstations and then ported to an iPSC860 with no additional changes.

4 Numerical results

In order to demonstrate some of the previous material two test problems are chosen which come from two-dimensional partial differential equations. The first problem is the linear diffusion equation defined on the unit square

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{19}$$

with Dirichlet boundary conditions given by

$$u(t, x, 0) = u(t, x, 1) = u(t, 0, y) = u(t, 1, y) = 1,$$

which can be converted into a system of ordinary differential equations by the method of lines. If the second order spatial derivatives are replaced by central finite differences on a uniform grid with the grid discretization parameter given by $h = \frac{1}{N+1}$ then this leads to a linear system of differential equations of size N^2 of the form

$$u' = (N+1)^2 Q u, \quad u(0) = 1. \tag{20}$$

Here Q is a block tridiagonal matrix of the form (I_N, T, I_N) where I_N is the identity matrix of order N and T is the tridiagonal matrix $(1, -4, 1)$ with -4 on the diagonal entries and 1 on the upper and lower subdiagonal entries. Here q for this problem can be $8(N+1)^2$.

The second problem is a reaction–diffusion equation known as the diffusion Brusselator equation [12] and takes the form

$$\left.\begin{array}{rcl} \frac{\partial u}{\partial t} & = & B + u^2 v - (A+1)u + \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \\ \frac{\partial v}{\partial t} & = & Au - u^2 v + \alpha \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \end{array}\right\} \tag{21}$$

with initial conditions

$$u(0,x,y) = 2 + 0.25y, \quad v(0,x,y) = 1 + 0.8x \quad A = 3.4, \quad B = 1, \quad \alpha = 0.002$$

and Neumann boundary conditions

$$\frac{\partial u}{\partial n} = 0, \quad \frac{\partial v}{\partial n} = 0.$$

Here u and v denote chemical concentrations of reaction products, A and B are constant concentrations of input reagents and α a constant based on a diffusion coefficient and a reactor length.

Again central differencing leads to a system of coupled nonlinear equation of order $2(N+2)^2$ (with $\hat{\alpha} = \alpha(N+1)^2$) of the form

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

4.1 SIMD implementation

The linear problem and the one-dimensional form of the Brusselator have been solved on the 4K MasPar sited at the University of Queensland using a fixed step-size scheme based on the trapezoidal corrector. Rather than present a number of tables recording the computational results, these will be summarized in the following remarks:

1. In the case of the linear problem of dimension N^2 some care must be taken in choosing N. For example, if $N = 65$ and the number of available processors is $64 \times 64 = 4096$ then the compuational time will be approximately twice as long as in the case $N = 64$. This is because the MasPar automatically layers the computational grid into memory so that the $N = 65$ case requires two layers. This is done automatically and does not require programmer intervention. On the other hand the time for solving an N^2 dimensional problem (where $N \leq 64$) should be approximately the same, but can depend on the machine load.

2. Although the implementation described in section 2 requires only BLAS operations of the form Qv, where Q is as in (20) and v is a vector defined on all the elements of the computational grid, it is important to structure the problem so this is done efficiently. This is achieved by representing v as an $N \times N$ matrix and forming Qv as a sequence of `EOSHIFT`s:

```
EOSHIFT(v,SHIFT=-1,BOUNDARY=f1,DIM=1) +
EOSHIFT(v,SHIFT=-1,BOUNDARY=f2,DIM=2) - 4.0*v +
```

```
EOSHIFT(v,SHIFT=+1,BOUNDARY=f3,DIM=2) +
EOSHIFT(v,SHIFT=+1,BOUNDARY=f4,DIM=1)
```

Here `SHIFT` represents a shift up or down the computational grid, `DIM` represents column or row shifts and $f1$, $f2$, $f3$, $f4$ the boundary conditions.

3. For the one-dimensional Brusselator, there are two coupled vectors each of dimension N and these are automatically layered as two row vectors onto the MasPar topology. For a Type II implementation, a Jacobian matrix has to be evaluated at each time step. Since the Jacobian matrix has a simple block tridiagonal structure with the Identity matrix as the off-diagonal blocks, the forming of the vector product of the Jacobian times each of the two vectors representing the components of the problem is easily done as a sequence of two `EOSHIFT`s columnwise for each vector.

4. (20) has also been solved by a block method of size 2 based on two-stage Radau corrector of order 3. In this case two approximations (one a third of the way along the integration step and one at the end of the integration step) are computed per processor. The computational time is, as expected, approximately twice that for the trapezoidal corrector.

4.2 MIMD implementation

The two-dimensional from of the Brusselator has been solved on a distributed cluster of Sparc 2 workstations and on a 32 processor iPSC860 at Jülich in Germany using Jacobi waveform in conjunction with the ODE package VODE. An automatic time-windowing is used based on adaptive monitoring of the convergence of the iterations. The message-passing has been programmed in P4 since this has meant that the program could be debugged and tested on a workstation cluster and then migrated to the iPSC860 without any change. Comparisons are made between VODE running sequentially on one processor and waveform running on all 32 processors. Two plots are given based on the measurement of the speed-up versus the dimension size $2N^2$ when there is no overlapping of the components and when the optimal overlap is chosen (in terms of the best speed-up for a particular dimension size). A speed-up of nearly 7 on a 32 node machine for problem of size 800 is certainly satisfactory.

5 Conclusions

The results obtained for the MasPar suggest that large moderately-sized stiff problems can be solved without recourse to the solution of large systems of linear equations (possibly at each time step) and that these techniques are ideally suited to massively parallel machines.

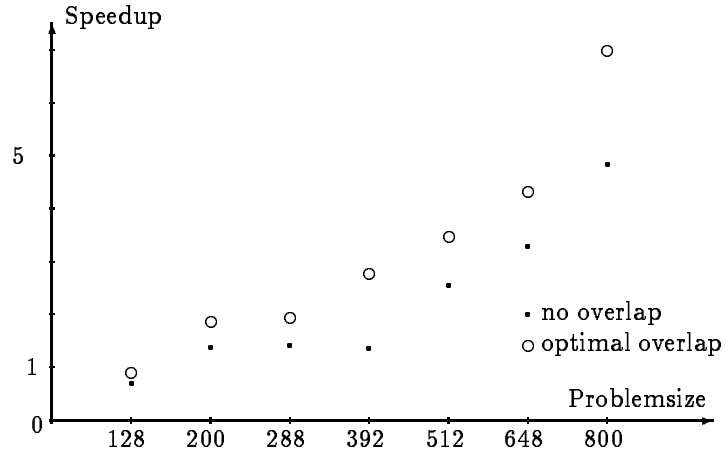

Figure 3: [0,6]; 80 timepoints; 32 processors;

In a MIMD environment the approach emphasized here suggests that, where possible, parallel algorithmic development should make use of existing sequential packages which have been fine-tuned over a number of years and which have proven to be robust and efficient. Not only does this provide some robustness to the parallel algorithms but means that the programmer only has to focus on the inter-processor communications. If this is done using packages such as PVM or P4 then this provides significant portability. It should be note that the implementation described here can be improved if a multigrid waveform is used.

Lack of space has prevented a more detailed discussion of the wide variety of parallel algorithms for solving ordinary differential equations. However, it is clearly true that not all these algorithms will fare equally well on a wide range of architectures. It seems likely that in the immediate future there will have to be a greater variety, than in the sequential case, of codes which are both problem-dependent and architecture-dependent. However, with an apparent vendor trend towards massively parallel MIMD machines it may well be that this situation is temporary and that there will be a uniformity and portability of codes across a large set of parallel machines not just in the area of differential equations but in all areas of scientific computation.

References

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

[2] K. Burrage, *The error behavior of a general class of predictor–corrector methods*, App. Num. Math., 8(1991), pp. 201–216.

[3] K. BURRAGE, *Parallel methods for initial value problems*, App. Num. Math., 11(1991), pp. 5–25.

[4] K. BURRAGE, *Parallel and sequential methods for differential equations*, monograph to be published, Oxford University Press, 1994.

[5] K. BURRAGE AND S. PLOWMAN, *The numerical solution of ODEIVPs in a transputer environment*, In Applications of Transputers2, eds. D.J. Pritchard, C.J. Scott, IOS Press, pp. 495–505., 1990.

[6] R. BUTLER, AND E. LUSK, *User's Guide to the p4 Programming System*, Argonne Nat. Lab. Rep. ANL-92/17.

[7] C.H. CARLIN AND C. VACHOUX, *On partitioning for waveform relaxation time-domain analysis of VLSI circuits*, in Proc. of Int. Conf. on Circ. and Syst., Montreal, 1984.

[8] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, V. SUNDERAN, *PVM 3.0 User's Guide and Reference Manual*, Report ORNL/TM–12187, Mathematical Sciences Section, Oak Ridge National Laboratory.

[9] E. LELARASMEE, *The waveform relaxation method for the time domain analysis of large scale nonlinear dynamical systems*, PhD thesis, University of California, Berkeley, CA, 1982.

[10] C. LUBICH AND A. OSTERMANN, *Multi-grid dynamic iteration for parabolic equations*, BIT, 27(1987), pp. 216–234.

[11] W.L. MIRANKER AND W. LINIGER, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comp., 21(1967), pp. 303–320.

[12] S. VANDEWALLE AND R. PIESSENS, *Numerical experiments with nonlinear multigrid waveform relaxation on a parallel processor*, App. Num. Math., 8(1991), pp. 149–161.

[13] J. WHITE, A. SANGIOVANNI-VINCENTELLI, F. ODEH AND A. RUEHLI, *Waveform relaxation: Theory and practice*, Trans. of Soc. for Computer Simulation, 2(1985), pp. 95–133.

[14] Z. ZLATEV, *Treatment of some mathematical models describing long-range transport of air pollutants on vector processors*, Parallel Computing, 6(1988), pp. 87–98.

Acknowledgements: My thanks to Pamela Burrage and Dr. Bert Pohl for their assistance in programming on the MasPar and the iPSC860, respectively.