A parallel matrix-free implementation of a Runge-Kutta code

Kevin Burrage, Craig Eldershaw and Roger Sidje
Department of Mathematics,
The University of Queensland,
Queensland 4072, Australia.
kb@maths.uq.edu.au, ce@maths.uq.edu.au, rbs@maths.uq.edu.au

August 9, 1999

Abstract

It is known that matrix-free numerical implementations for solving stiff ordinary differential equations (ODEs) can be considerably more effective than implementations which rely on direct linear algebra techniques to solve the implicit equations governing the stage values. In this paper it will be shown how fully implicit, high order Runge-Kutta methods can be efficiently implemented in a matrix-free, parallel environment. The advantage of this is that no new parallel algorithms need be developed and that existing sequential methods that are adpated using these techniques need have no special structure (such as singly implicitness). This is demonstrated by the conversion of an existing Radau IIA method (RADAU5) to a matrix-free implementation using a dynamically pre-conditioned GMRES algorithm to solve the appropriate linear systems. Numerical results are presented for an implementation on a shared memory SGI Power Challenge and show the efficacy of this approach.

1 Introduction

Recently a great deal of work has been done in developing novel ODE methods for exploiting parallelism. In this paper it is shown how an existing sequential method can easily be adapted to give effective parallelism using matrix-free techniques.

Thus consider the IVP, written in non-autonomous form,

$$y'(x) = f(x, y(x)), \quad f : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m, \quad y(x_0) = y_0. \tag{1}$$

Parallel techniques for solving differential systems such as (1) can arise at many levels. At the first level there is the parallelization of extant sequential code through the use of parallelizing compilers and the restructuring of DO loops, while at the second level, parallelization of serial algorithms can take place. In addition, if the system is stiff and large, then enhanced performance can also be gained through the use of parallel linear algebra.

A more novel approach requires the modification and redesign of sequential algorithms in terms of the target parallel architecture. At a very basic level this can involve exploiting concurrent function evaluations within a step or methods that compute blocks of values simultaneously. Such approaches are called by Gear (1986) **parallelism across the method**. A very important coarse-grained technique is via the decomposition of a problem into subproblems which can then be solved in parallel, with the processors communicating as appropriate. Multirate and decomposition techniques such as waveform relaxation (White et al. (1985)) are examples of **parallelism across the system.** A third approach called **parallelism across the steps** includes techniques, such as the parallel solution of recurrences, which solve simultaneously over a large number of steps. Efficient parallel algorithms may well take elements from all three of these categories.

In order to get effective performance from a parallel implementation, a number of conditions relating to the nature of the problem should hold (Burrage (1995)). In particular, the problem should possess at least one of the following attributes:

- expensive function evaluations; long integration interval; repeated integration; large dimension.

Of all these factors, perhaps the last is the most significant. There is no point in expecting reasonable parallel performance if the dimension of the problem is only one or two orders of magnitude, since it is highly likely that for such problems communication time will dominate computation time.

In this paper only Runge-Kutta methods will be considered, as they offer the opportunity to exploit parallelism across the method. An s-stage Runge-Kutta method applied to (1) takes the form

$$\begin{aligned} Y_i &= y_n + h\sum_{j=1}^{s} a_{ij} f(x_n + hc_j, Y_j), \quad i = 1, \ldots, s \\ y_{n+1} &= y_n + h\sum_{j=1}^{s} b_j f(x_n + hc_j, Y_j), \end{aligned} \tag{2}$$

where $Y_1, \ldots, Y_s$ represent the intermediate approximations to the solution at $x_n + c_1 h, \ldots, x_n + c_s h$.

Such a method can be represented by the so-called **Butcher tableau**

$$\begin{array}{c|c} c & A \\ \hline & b^{\mathsf{T}} \end{array}, \tag{3}$$

where

$$c = (c_1, \ldots, c_s)^{\mathsf{T}}, \quad b^{\mathsf{T}} = (b_1, \ldots, b_s), \quad c = Ae, \quad e = (1, \ldots, 1)^{\mathsf{T}}.$$

The maximum order of (2) is $2s$ and these high order methods are known to be $A-$stable with a maximal stage order of s. Another important class of methods are those satisfying the condition

$$e_s^{\mathsf{T}} A = b^{\mathsf{T}} \iff b_j = a_{sj}, \quad j = 1, \ldots, s, \quad e_s^{\mathsf{T}} = (0, \ldots, 0, 1) \tag{4}$$

and are called **stiffly accurate**. These methods can have maximal order $2s - 1$ and include the RadauIIA methods. The RadauIIA method with $s = 3$ is given by

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

Radau IIA, s=3

2 Implementation

Consider the autonomous version of (1) and let

$$Y = (Y_1^{\mathsf{T}}, \ldots, Y_s^{\mathsf{T}})^{\mathsf{T}}, \quad F(Y) = (f(Y_1)^{\mathsf{T}}, \ldots, f(Y_s)^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{R}^{sm};$$

then any Runge-Kutta method can be written in tensor notation as

$$\begin{aligned} Y &= e \otimes y_n + h(A \otimes I_m) F(Y) \\ y_{n+1} &= y_n + h(b^{\mathsf{T}} \otimes I_m) F(Y). \end{aligned} \tag{5}$$

This system of sm nonlinear equations can be solved by using the Newton iteration scheme.

Thus, if at the end of one iteration Y_i is replaced by $Y_i + \delta_i$, then

$$(I_s \otimes I_m - h\tilde{J})\delta = \Psi, \tag{6}$$

where $\tilde{J}$ is a block matrix of order sm whose (i, j) element is $a_{ij} f'(Y_j)$, $i, j = 1, \ldots, s$ and

$$\begin{aligned} \delta &= (\delta_1^{\mathsf{T}}, \ldots, \delta_s^{\mathsf{T}})^{\mathsf{T}}, \quad \Psi = (\Psi_1^{\mathsf{T}}, \ldots, \Psi_s^{\mathsf{T}})^{\mathsf{T}} \\ \Psi_i &= -Y_i + y_n + h\sum_{j=1}^{s} a_{ij} f(Y_j), \quad i = 1, \ldots, s. \end{aligned}$$

This can be simplified if the $f'(Y_j)$ are approximated by evaluating the Jacobian at a single point (y_n say). Thus by letting $J = f'(y_n)$, then (5) can be written as

$$(I_s \otimes I_m - hA \otimes J)\delta = \Psi. \tag{7}$$

If (7) is solved by an LU factorization followed by forward and back substitutions, then the number of floating point operations to perform one iteration of the Newton process is, respectively,

$$O(cm^3/3), \quad O(dm^2), \quad \text{where} \quad c = s^3 \quad \text{and} \quad d = s^2.$$

This is approximately s^3 times the work for the equivalent implementation of a Backward Differentiation Formulae (BDF) method where $c = d = 1$. Fortunately, structures can be imposed on the Runge-Kutta matrix which reduce the computational cost of certain classes of implicit Runge-Kutta methods to approximately that of a linear multistep method. The nature of these structures depends on whether a sequential or parallel implementation is being considered. In the sequential case there are two natural structures which lead to methods known as singly implicit and diagonally implicit methods.

Butcher (1976) proposed an ingenious technique for improving the computational efficiency of implicit Runge-Kutta methods by transforming A to its Jordan canonical form. In the case that A has only real eigenvalues, and is similar to a diagonal matrix, then $c = d = s$. On the other hand if A has a one-point spectrum (so-called SIRKs (Burrage (1978)), so that A is similar to the matrix with λ on the diagonal and $-\lambda$ on the subdiagonal, then $c = 1, \quad d = s$.

This approach compares favourably with the computational cost for BDF methods. However, at each iteration of the transformed Newton procedure, Butcher's technique requires the Y and $F(Y)$ to be transformed and untransformed by the similarity transformation matrix. If l iterations are performed per step, the total cost for a singly implicit method and BDF method over one integration step is approximately

$$C_S = m^3/3 + lsm^2 + 2ls^2m, \quad C_B = m^3/3 + lm^2.$$

Thus only if m is large is C_S comparable with C_B.

Butcher's transformation technique applies to any implicit method, but if the order of a Runge-Kutta method is greater than $s+1$, then some of the eigenvalues are necessarily complex. This, for example, is the case of the three-stage, stiffly accurate Radau IIA method of order 5 (which has been implemented (RADAU5) by Hairer and Wanner (1991)) in which Λ takes the form

$$\begin{pmatrix} \gamma & & 0 \\ 0 & \alpha & -\beta \\ 0 & \beta & \alpha \end{pmatrix}.$$

In this case, two linear systems of dimension m and $2m$ must be solved. Alternatively, the real $2m \times 2m$ linear system could be solved as an m-dimensional complex system. If s and m are both small, then the high-order methods (such as the Radau IIA methods) are competitive. However, as the dimension of the problem becomes large it can be seen that Radau IIA methods become approximately $2s-2$ or $2s-1$ times as expensive as singly implicit methods (although their order is approximately double).

3 Matrix-free methods and parallelism

One of the disadvantages of all the above implementations is that they require the use of the Jacobian matrix in solving for the intermediate stages and that it is explicity assumed that direct linear algebra techniques (such as LU-factorisation) will be used for solving the ensuing linear system of equations. In the case of very large problems, storing the Jacobian explicitly can provide very real problems either in terms of memory limitations or in extracting the data from potentially slow secondary memory. Furthermore, if the Jacobian is not available in its analytic form, then although it can be calculated by a series of finite differences of function evaluations, this leads to $O(m^2)$ function evaluations just to calculate a single Jacobian. Even if the matrix is sparse there can be problems associated with fill-in if direct linear algebra methods are used. For these reasons some researchers have considered so-called matrix free implementations, in which the Jacobian matrix is never explicitly computed or stored and exploit recent advances in iterative techniques for linear systems based on preconditioned Krylov GMRES methods (Saad and Schultz (1986)) - see VODPK (Brown and Hindmarsh (1989)) and DASPK (Brown et al. (1993)). VODPK has a very similar structure to VODE (Brown et al. (1989)) except that a preconditioned GMRES technique based on a scaled preconditioned incomplete version of GMRES (SPIGMR) is used for the linear solver, while DASPK is based on DASSL, a code for solving differential-algebraic equations. VODPK allows for both left and right preconditioned matrices P_1 and P_2 so that the iterative method is applied to $P_1^{-1}QP_2^{-1}$, where P_1P_2 is an approximation to Q. In the stiff mode, the user must supply two routines: JAC, which evaluates and processes any part of the Jacobian involving P_1 and P_2, and PSOL, which solves the linear systems involving P_1 or P_2 as the coefficient matrix. In the case of DASPK only a left preconditioner is used.

Brown and Hindmarsh (1989) note that these preconditioners have to be chosen very carefully, but if they are chosen well then VODPK can be very effective. One of the advantages of iterative solvers is that the factored preconditioners can often be re-used over many more integration steps than is the case with the iteration matrix

in a direct solve. Another advantage (not exploited in VODE but which is used in VODEPK and DASPK) is that the Jacobian matrix need never be computed explicitly, since only Mv need ever be computed. Here v is some vector in the GMRES implementation, while M represents the matrix obtained from the application of Newton's method to the implicit stage approximations - see below. This matrix-vector quantity can be computed by a differencing of the form

$$(f(x, y + \sigma v) - f(x, y))/\sigma,$$

where σ is a vector with small components tending towards 0.

Maier et al. (1994) have used a preconditioned GMRES iteration for the associated linear systems within DASSL when solving large-scale Differential-algebraic equations (DAEs) on a 512-processor CM5. They showed that the quality of the preconditioner can significantly influence the stepsize, and that no one preconditioner performed uniformly well on each of the three test problems.

The implicit equations to be solved for the stage components is of the form $Mv = (I - \gamma A \otimes J)\sigma = b$. Here J is assumed to represent some approximation to the Jacobian at a single point (y_n say). By using an iterative linear algebra method which only ever uses J when pre-multiplying a vector (ie. the matrix only ever appears in the form $J\delta$) then a simple vector difference approximation can be made. Suitable first-order and second-order approximations are given by

$$J(x)\delta = \lim_{\sigma \to 0} \left(\frac{f(x + \sigma\delta) - f(x)}{\sigma} \right) = \lim_{\sigma \to 0} \left(\frac{f(x + \sigma\delta) - f(x - \sigma\delta)}{2\sigma} \right).$$

This differencing scheme can be used to generate an expression for Mv. Given that $A = [a_{ij}]$ is of size $(s \times s)$, J of $(m \times m)$, I of $(ms \times ms)$ and that $v = (\delta_1^T, \ldots, \delta_s^T)^T$ it can then be deduced that

$$Mv = (I - \gamma A \otimes J)v = \begin{pmatrix} \delta_1 - \gamma \sum_{j=1}^{s} a_{1j} J\delta_j \\ \vdots \\ \delta_s - \gamma \sum_{j=1}^{s} a_{sj} J\delta_j \end{pmatrix},$$

where all the $J\delta_j$ terms can be approximated using one of the above differencing schemes.

Thus if any Runge-Kutta method is implemented as a matrix-free algorithm in the manner described above then an obvious improvement can be obtained by parallelism, since the s terms $J\delta_1, \ldots, J\delta_s$ are completely independent, and can be calculated independently, and can also be summed independently.

By calculating each of these terms on s processors, the time required for this part of the program can be reduced by up to a factor of about s. We have therefore modified the RADAU5 code to create RADpk, a matrix-free version. The modifications were along the same lines as those made transforming VODE to VODpk. One additional benefit of the switch to a matrix-free method is that the transformation of A to TAT^{-1} is no longer necessary and avoids a costly overhead.

In addition, some new iterative techniques developed in Burrage et al. (1996) and Erhel et al. (1996) are used in RADpk. These allow the adaptive construction of a preconditioner for GMRES which resolves the challenging problem of constructing suitable preconditioners for efficient matrix-free implementations. The idea is to estimate the invariant subspace corresponding to the smallest eigenvalues and after each restart new eigenvectors are estimated so as to increase the invariant subspace. The preconditioner built is almost equal to the matrix on the approximated invariant subspace and is taken as the identity on the orthogonal subspace. Numerical results presented in Erhel et al. (1996) show that for many systems this technique can converge much faster than the standard restarted version and almost as fast as the full scheme.

4 Numerical results

Two test problems are used in this paper. The first is the reaction-diffusion Brusselator problem (solved using the method of lines) given by

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

with initial conditions

$$u(0, x, y) = 2 + \mu_1 xy, \quad v(0, x, y) = 1 + \mu_2 x, \quad t \in [0, 1],$$
$$A = 3.4, \quad B = 1, \quad \alpha = 0.002, \quad \mu_1 = 0.25, \quad \mu_2 = 0.8$$

and Neumann boundary conditions $\frac{\partial u}{\partial n} = 0, \quad \frac{\partial v}{\partial n} = 0$.

When discretised by a $N \times N$ array of points, it yields the following $2N^2$ equations:

$$u'_{i,j} = 1 + u^2_{i,j} v_{i,j} - 4.4 u_{i,j} + \alpha(N+1)^2 (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j})$$

$$v'_{i,j} = 3.4_{i,j} - u^2_{i,j} v_{i,j} + \alpha(N+1)^2 (v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{i,j})$$

With boundary conditions

$$u_{0,j} = u_{2,j}, \quad u_{N+1,j} = u_{N-1,j}, \quad u_{i,0} = u_{i,2}, \quad u_{i,N+1} = u_{i,N-1}$$

$$v_{0,j} = v_{2,j}, \quad v_{N+1,j} = v_{N-1,j}, \quad v_{i,0} = v_{i,2}, \quad v_{i,N+1} = v_{i,N-1}.$$

The $2N^2$ variables could be ordered in any manner, the order chosen was such that the u and v alternated in the form $(u_{11}, v_{11}, u_{12} \ldots, u_{NN}, v_{NN})^T$. This gives a smaller bandwidth for the Jacobian (important for direct factorization methods) than other orderings.

Four programs were used to solve this problem: VODE, VODpk, RADAU5 and RADpk and both the absolute and relative errors were set to 10^{-5}. VODE and RADAU5 were each run twice using internal differencing: once configured to use a full Jacobian and once to use a banded Jacobian (with half-bandwidth of $2N$).

$M = 2N^2$	RADAU5	VODE	RADAU5b	VODEb	RADpk	VODpk
200	1.87	0.59	0.385	0.126	0.48	0.028
450	16.0	4.04	2.14	0.478	1.13	0.079
800	70.9	18.1	6.01	1.47	2.21	0.159
1250	241	59.2	14.5	2.88	3.75	0.282
1800	581	143	31.0	6.17	8.57	0.403
2450	-	322	66.6	11.4	8.61	0.639
3200	-	-	95.4	20.2	16.8	0.828
5000	-	-	245	48.0	31.3	1.48

Table 1: timings in seconds for the Brusselator

Note that the RADAU5 and VODE trials were not performed for the largest values of N due to the inordinate length of time required, and that the suffix b denotes banded versions. It is clear from this table how the two matrix-free methods clearly outperform the two direct Jacobian methods. For the larger values of M, this difference is two to three orders of magnitude. This is in addition to only allocating $O(M)$ bytes of memory as opposed to the $O(M^2)$ required for the full-matrix versions.

The RADpk code was also parallelised in accordance with the description in section 3 on the Brusselator problem and was run with several different values for N with 1, 2 and 3 processors on a shared memory SGI Power Challenge sited at the University of Queensland. The times are given in Table 2 table below.

$M = 2N^2$	1 processor	2 processors	3 processors
5000	9.28	8.47	7.78
7200	13.6	12.3	11.2
9800	83.0	64.8	48.8
12800	121	118	113
16200	298	292	283

Table 2: parallel performance on the Brusselator

Here the speed-up is not dramatic because the function evaluations are not expensive and the problem is sparse so communication effects outweigh computation effects. In order to see what happens when this is not the case we constructed a dense expensive problem given by

$$y' = Qy + g, \quad y(0) = e, \quad t = [0, t_M],$$

where Q is a random symmetric negative definite matrix and g is the forcing term such that $g_i = \exp(\sum_{j=1}^{i} -y_j^2)$. Because the problem seems to be unstable for moderate size values of t_M, $t_M = 0.01$ and 0.001 for M equal 1000 and 2000, respectively.

The times and speed-ups obtained are presented in Tables 3 and 4. Along with the overall time, the bracketed figure gives the time spent in the code-section corresponding to the evaluation of the s-stage components of the method (Ftime). Table 4 shows that with 3 processors the speed-up in the Ftime component of approximately

M	1 processor	2 processors	3 processors
1000	34.29(28.92)	26.88(21.24)	17.31(11.61)
2000	133.95(121.12)	105.87(91.95)	62.48(49.25)

Table 3: timings for the dense problem

2.5 gives an efficiency of over 80%. Note that we could have used more than three processors by partitioning the function evaluations accordingly, and a slightly better performance than that recorded might also be expected if the accumulating sums of the $J\delta_j$ were done in parallel, but this was not done here.

M	2 processors	3 processors
1000	1.27(1.36)	1.98 (2.49)
2000	1.26(1.31)	2.14 (2.46)

Table 4: total speed-up and Fspeed-up (brackets)

5 Conclusions

This paper has highlighted the problems with conventional ODE solvers which use full Jacobian evaluations in their Newton step and shown how these algorithms could be modified to be in a matrix free form. The benefits of such a modification are quite marked both in terms of speed, and in terms of memory consumption. Furthermore, it has been shown how some very natural parallelism exists within any implicit Runge-Kutta method which has been converted in this manner. However the speed-ups gained from such parallelism are only significant if the function evaluations themselves are relatively expensive. All of these benefits have been demonstrated by the successful modification of the RADAU5 program written by E. Hairer and G. Wanner into a matrix-free, parallel version, here called RADpk.

References

Brown, P.N. and Hindmarsh, A.C. (1989). Reduced storage matrix methods in stiff ODE systems. *J. Appl. Math. and Comp.*, **31**, 40–91.

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

Brown, P.N., Hindmarsh, A.C. and Petzold, L.R. (1993). *Using Krylov methods in the solution of large-scale differential-algebraic systems.* Rep. TR 93–37, Dept. Comp. Sci., University of Minnesota, U.S.A.

Burrage, K. (1978). A special family of Runge-Kutta methods for solving stiff differential equations. *BIT*, **18**, 22-41.

Burrage, K. (1995). Parallel and sequential methods for ordinary differential equations. Oxford University Press.

Burrage, K., Erhel, J. and Pohl, B. (1996). *A deflation technique for linear systems of equations.* To appear in SISC.

Butcher, J.C. (1976). On the implementation of implicit Runge-Kutta methods. *BIT*, **6**, 237–240.

Erhel, J., Burrage, K. and Pohl, B. (1996). Restarted GMRES preconditioned by deflation. *J. of Comp. and App. Maths.*, **69**, 303–318.

Gear, C.W. (1986). *The potential of parallelism in ordinary differential equations.* Tech. Rep. UIUC–DCS–R–86– 1246, Comp. Sci. Dept., Univ. of Illinois at Urbana-Champaign, U.S.A.

Hairer, E. and Wanner, G. (1991). *Solving ordinary differential equations II, stiff and differential-algebraic problems.* Springer-Verlag, Berlin.

Maier, R.S., Petzold, L.R. and Rath, W. (1994). *Parallel solution of large-scale differential-algebraic systems.* TR 94–10, Dept. Comp. Sci., Univ. of Minnesota, U.S.A.

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

White, J., Sangiovanni-Vincentelli, A., Odeh, F. and Ruehli, A. (1985). Waveform relaxation: theory and practice. *Trans. of Soc. for Computer Simulation*, **2**, 95–133.