Asian Journal of Information Technology 6 (11): 1110-1116, 2007
ISSN: 1682-3915
© Medwell Journals, 2007

VSMRK: A Parallel Implementation and the Performance of Variable Stepsize Multistep Runge-Kutta Methods for Stiff ODEs

[1]H. Suhartanto and [2]K. Burrage
[1]Faculty of Computer Science, University of Indonesia, Depok 16424, Indonesia
[2]Department of Mathematics, The University of Queensland, Brisbane 4072, Australia

Abstract: Many natural phenomena and applications in industry can be modeled as systems of Stiff Initial Value Problems (IVPs) of Ordinary Differential Equations (ODEs). Usually the problems to be solved are huge in dimension, hence require huge computing resources and time. This study describes the parallel implementation of Variable stepsize Multistep Runge-Kutta (MRK) method of Radau Type for solving stiff IVPs of ODEs and its performance on SGI Origin 2000. The numerical results show the superiority of the code compared to various standard code on dense and large sparse problems.

Key words: Multistep runge-kutta, stiff ordinary differential equations, parallel implementation

INTRODUCTION

Several numerical experiments using Multistep Runge-Kutta (MRK) methods have been performed to solve stiff Initial Value Problems (IVPs) for systems of Ordinary Differential Equations (ODEs)

$$y'(x) = f(y(x)), \quad f : R^m \circledR R^m, y(x_0) = y_0 \qquad (1)$$

for example (Burrage and Suhartanto 1997; Schneider, 1993). This equation represents some physical phenomena such as evaluating ozone depletion, designing aircraft, researching the human anatomy, real-time forecasting based on sophisticated climate models, simulating reactions, superconductivity and molecules, simulating semiconductors materials, simulating the behavior of nuclear power plants and weapons in operating conditions exceeding the safety limit of practical experiments and study of turbulence and seismic data.

However, these methods used constant stepsize MRK formula. In this study, the underlying method is the r-step, s-stage Variable stepsize Multistep Runge-Kutta (VMRK) method of Radau type described in Burrage and Suhartanto (1997), Burrage and Suhartanto (2000) and Suhartanto (1998). In this study, the methods used will be briefly reviewed. This includes the underlying methods, iteration techniques and the parallel technique. These will be followed by section detailing the numerical experiments and the discussion of the results.

MATERIALS AND METHODS

The core methods: The underlying method is the r-step, s-stage Multistep Runge-Kutta (MRK) method of Radau type characterized by

$$\begin{aligned} Y_n &= (A \otimes I_m)y^n + h_n(B \otimes I_m)F(Y_n), \\ y_{n+1} &= (e_s^T \otimes I_m)Y_n \end{aligned} \qquad (2)$$

Here, $Y_n \in R^{sm}$ represents s stages vectors $(Y^T_{n,1}\ldots,Y^T_{n,s})^T$ and are approximations to the solutions at the off step points $x_n + c_i h_n$, $i = 1,\ldots,s$ and $F(Y_n) \in R^{sm}$ represents *s*-derivatives of stage vectors $(f(Y_{n,1})^T,\ldots, f(Y_{n,1})^T)^T$, $y^{(n)} \in R^{rm}$ represents r previous solutions $(y^T_n,\ldots, y^T_{n+1-r})$ and I_m denotes the identity matrix of order m, h_n is the integration step, A is an s-by-r matrix and B is an s-by-s matrix and $\otimes$ is Kronecker product. All parameters of the variable stepsize methods, (c, A, B) depend on the ratio of the stepsize given by

$$\rho_{n,r-i} = \frac{h_{n-i}}{h_n}, i = 1,\ldots, r-1,$$

Where, $h_i = x_{i+1} - x_i$, $i = n - r + 1,\ldots,n$.
By defining

$$c^j = [c_1^j,\ldots,c_s^j], \quad q^0 = e, \quad q^p = [q_1^p,\ldots,q_r^p]^T \quad (p \geq 1),$$

Where:

$$e = [1,\dots,1]^T \in R^r, \quad q_1 = 0, \quad q_j = \frac{-\sum_{i=1}^{j-1} h_{n-i}}{h_n}, \quad j = 2,\dots,r.$$

and considering also the assumptions

$$\begin{aligned} B(w): &\quad p\gamma^T c^{p-1} + \alpha^T q^p = 1 \quad p = 1,\dots,w \\ C(\eta): &\quad pBc^{p-1} + Aq^p = c^p \quad p = 0,\dots,\eta \end{aligned}$$

We proved in Burrage and Suhartanto (2000) and Suhartanto (1998) that The maximum attainable order of a stiffly accurate VMRK method is 2s+r-2. Methods with these order exist satisfying C(s + r - 1) and with $c_1,\dots, c_{s-1}$ real, distinct and lying in the interval $[q_r, 1]$.

The iteration technique: To solve Y_n in (2), we proposed an iteration technique in Burrage and Suhartanto (1997) and Suhartanto (1998) that is by iterating Y_n for example L times

$$\begin{aligned} Y_n^{(j)} &= (A \otimes I_m)y^{(n)} + h_n([B - W] \otimes I_m)F(Y_n^{(j-1)}), \\ &+ \quad h_n(W \otimes I_m)F(Y_n^{(j)}), \ j = 1,\dots,L \qquad (3) \\ y_{n+1} &= (e_s^T \otimes I_m)Y_n^{(L)} \end{aligned}$$

Where $Y^{(0)}_n$ denotes an initial approximation or predictor to the vector Y_n and W is the splitting matrix.

Several types of predictors for $Y^{(0)}_n$ are considered, these are P_0 which is a trivial predictor where we define $Y^{(0)}_n = Y_{n-1}$, P_1, which is an extrapolation predictor using r previous solutions, P_2 which is an extrapolation predictor using r previous solutions and s previous stages vectors, P_3 which is an extrapolation predictor using s previous stages vectors of the last step and P_4 which is an extrapolation predictor using s previous stages vectors of the last step and y_{n-1}. We also proved in Suhartanto (1998) that given a predictor $Y^{(0)}_n$ of order p, it attains order p+l after l iterations.

The parallelism: In Suhartanto (1998), it was also noted that the matrix W in (3) should be in a form such that the iteration could be done in parallel for each $Y^{(j)}_{n,1}$, i = 1,...,s. The obvious choice of this matrix is W = D, where D is some diagonal matrix, hence the s components of each vector $Y^{(j)}_n$ can be computed in parallel provided that there are s processors available. In the case of the fixed-step-size methods, W is easily defined as it is fixed, available and used on every step. However, this is not the case for a VMRK method, as the computation of W requires a certain condition to be satisfied which is difficult to implement. For example, consider choosing W = D, then it is required that the spectral radius of the matrix $(I - D^{-1} B)$ vanish in order to have good convergence for stiff problems. However, in general these schemes are very delicate and very difficult to adapt to our methods as it requires symbolic computation of D. However, for methods with small number of stages, say three, we will construct the splitting matrices W. In this implementation we define them to be the triangular matrices obtained from the application of Crout methods on B. This requires the use of Butcher similarity transformation matrices but these can easily be determined.

Now since the computation of each $Y^{(j)}_{n,i}$, I = 1,..., s is independent because of the choice of the spliting matrix W then the process of evaluating the derivative, factorization of the iteration matrices and solving the linear systems of each of the stages are done concurrently.

RESULTS

The computing environment: Some tests were done on a SGI Power Challenge and a SGI Origin2000 system comprising 64 R10000 MHz CPUs each with 4 MB cache, a shared-memory multiprocessor machine, i.e., it consists of a collection of homogeneous processors (equivalent capabilities) which can execute distinct instruction streams in parallel. The machine, sited at the University of Queensland, runs IRIX release 6.2 IP21 and the code is written in Fortran 90 compiled with MIPSpro Fortran 90.

Problems tested: The first two test problems BRUS-0 and BRUS-1 are based on the reaction-diffusion Brusselator problems

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

BRUS-0 uses intial conditions

$$u(0,x,y) = 22y(1-y)^{1.5}, \ v(0,x,y) = 27x(1-x)^{1.5}, \\ t \in [0,1.5] A = 3.4, \ B = 1, \alpha = 1/m^2$$

Whereas BRUS-1 uses intial conditions

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

Both BRUS-0 and BRUS-1 uses Neuman boundary conditions.

Table 1: Timing and accuracy obtained using predictors P0, P1, P2, P3 and P4 (on the SGI Origin, 2000)

		N = 10	N = 15	N = 20	N = 26	N = 30	N = 36
		BRUS-0					
B-ord							
	4	2.21/4.3	5.79/6.9	1.411/6.6	28.46/6.6	51.79/6.5	80.12/5.8
	3	1.97/6.4	5.44/6.4	12.30/4.6	24.77/6.0	43.80/6.3	75.37/5.8
	2	2.53/7.1	7.00/7.5	15.95/7.0	32.92/7.1	56.08/6.5	91.34/6.5
	1	2.21/7.0	6.20/11.0	14.54/11.0	28.12/11.0	50.51/11	80.46/×
A-ord							
	4	6.76/1.7	92.97/2.0	131.7/1.7	422.54/2.4	+ / +	+ / +
	3	6.14/2.1	30.89/1.3	123.4/2.6	378.09/ 2.0	+ / +	+ / +
	2	7.36/1.8	39.85/1.6	154.2/1.9	488.78/2.8	+ / +	+ / +
	1	6.37/1.9	95.11/1.6	198.0/1.3	417.95/1.9	+ / +	+ / +
	0	6.86/2.9	35.43/11.0	143.5/11.0	425.41/11.0	+ / +	+ / +
		BRUS-1					
B-ord							
	4	1.04/7.4	1.82/4.7	4.33/7.4	8.61/4.2	14.46/7.1	26.6/7.5
	3	0.60/6.4	1.45/3.7	3.60/6.4	8.26/7.1	13.61/7.2	22.37/3.8
	2	0.86/7.2	2.22/7.1	4.41/7.2	9.30/7.2	15.98/9.9	28.67/7.3
	1	0.66/6.3	1.66/6.5	3.69/6.8	9.10/7.6	15.9/7.8	25.01/7.8
	0	0.75/7.5	1.76/11.0	3.98/11.02	8.98/11.0	14.11/11.0	22.77/11.0
A-ord							
	4	2.52/7.4	14.52/4.7	43.28/7.4	127.29/4.2	347.52/7.1	+ / +
	3	2.14/6.4	9.90/3.7	38.57/6.4	117.15/7.1	330.55/7.2	+ / +
	2	2.24/7.2	12.64/7.1	44.46/7.2	132.11/7.2	346.06/9.9	+ / +
	1	1.92/6.3	10.29/6.5	37.19/6.8	133.81/7.6	341.24/7.8	+ / +
	0	3.03/7.5	11.45/11.0	42.21/11.0	132.76/11.0	336.15/11.0	+ / +

Table 2: Timing and accuracy at N = 30 and various tolerances (on the SGI Origin, 2000) with A/B ordering, m = 2N^2

TP	-log(tol) = 6	7	8	9	10
BRUS-0					
4	38.84/4.4	42.07/5.5	52.55/6.3	57.25/6.9	82.54/7.9
3	38.86/4.5	37.39/5.3	45.45/6.6	56.66/6.9	73.33/4.7
2	37.45/4.7	46.10/6.5	54.58/6.5	70.56/7.7	105.07/8.4
1	39.74/5.0	42.23/5.5	50.29/6.3	64.88/7.2	80.89/8.4
0	31.65/4.7	35.95/44	49.81/6.7	70.90/4.8	96.91/88
BRUS-1					
4	12.49/5.3	12.59/4.1	15.11/7.0	15.96/7.5	20.54/8.3
3	12.22/3.8	12.39/6.0	14.86/7.0	17.31/77	18.60/85
2	13.01/5.8	15.47/6.9	16.00/3.9	19.31/8.2	23.16/88
1	13.02/5.3	13.67/6.5	15.36/7.8	17.31/8.1	22.69/89
0	16.66/5.4	14.14/6.4	15.12/7.4	17.72/8.4	20.04/91

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

When the Brusselator problems discretized by N × N arrays points. It yields the following m = 2N^2 equations.

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

It is obvious that there at least two natural ways of ordering the u and v components of the system raised from the problem (Burrage, 1995). These orderings, respectively (A) and (B) are

$$u_{11},\ldots,u_{1N},u_{2N},\ldots,u_{NN},v_{11},\ldots,v_{2N},\ldots,v_{NN}$$
$$\text{and } u_{11},v_{11},u_{12},v_{12},\ldots,u_{NN},v_{NN.}$$

For the ordering (A), the Jacobian of the problem has the structure

$$\begin{pmatrix} T_1 & D_1 \\ D_2 & T_2 \end{pmatrix}$$

Here each T$_i$ is a block-tridiagonal matrix consisting of N blocks of blocksize N. The diagonal blocks of the T$_i$ are tridiagonal matrices, while the off-diagonal blocks are diagonal. D$_1$ and D$_2$ are also diagonal matrices. It was not mentioned in Burrage (1995) that the boundary conditions applied will effect the structure of the Jacobian. For instance, if the following boundary conditions

$$\begin{aligned} &u_{o,j} = u_{2,j}, \quad u_{N+1,j} = u_{N-1,j}, \quad u_{i,0} = u_{i,2}, \quad u_{i,N+1} = u_{i,N-1} \\ &v_{o,j} = v_{2,j}, \quad v_{N+1,j} = v_{N-1,j}, \quad v_{i,0} = v_{i,2}, \quad v_{i,N+1} = v_{i,N-1} \end{aligned} \quad (4)$$

is used then the ordering (B) leads to a block-tridiagonal Jacobian, consisting of N blocks of dimension 2N. The diagonal blocks are themselves pentadiagonal matrices and the off-diagonal blocks are diagonal matrices. However, when the following boundary conditions (applied in the Brusselator problem with 2-D diffusion (Hairer, 1997).

$$u_{0,j} = u_{N-1,j},\quad u_{N+1,j} = u_{1,j},\quad u_{i,0} = u_{i,N-1},\quad u_{i,N+1} = u_{i,1}$$
$$v_{0,j} = v_{N-1,j},\quad v_{N+1,j} = v_{1,j},\quad v_{i,0} = v_{i,N-1},\quad v_{i,N+1} = v_{i,1} \qquad (5)$$

are used then the ordering (B) leads to a block-tridiagonal Jacobian but with the half-bandwith $2(N^2\text{-}N)$ and the ordering (A) gives the same structure of the Jacobian. For this reason, we only run the experiments with boundary condition (4).

The next problem tested is the Dense problem defined in Burrage (1995), Burrage *et al.* (1997) and Suhartanto (1998) as the following;

$$y'_i = (QY)_i + e^{-\sum_{j=1}^{i} y_j^2},\ i = 1,\ldots,$$
$$m, Y \in R^m, Y(0) = e,\ x \in [0,1]$$

Where $Q = Q^T_1 DQ_1$, Q_1 has orthonormal columns of a random dense matrix M and

$$D(i,i) = \begin{cases} 100 & \text{if } (\mathrm{mod}(i,10)) = 0 \\ \mathrm{mod}(i,10) & \text{otherwise} \end{cases},\ i = 1,\ldots,m$$

DISCUSSION

To see the performance of the code with several types of predictors,we solve the BRUS-0 problem and BRUS-1 problem with two orderings at tolerance 10^{-8}. Numerical Jacobian approximation is used in all tests on Brusselator problems. Method VS33 with three processors is used. In order to see the accuracy obtained, the estimated `exact' solution is computed by the R14 method with four processors at tolerance 10^{-11}. The results are recorded in Table 1, where the number 0,1,2,3,4 in the first column represents the predictors used, P_0, P_1,..., P_4; A-ord and B-ord labels is A-ordering and B-ordering, respectively and the x/y table entries indicate time spent in x seconds and the y digits of accuracy.

It is obvious that more time is required to solve the problem with A-ordering and in some tests it produces less accurate solutions. One can also recognize that the predictor P3 outperforms the other predictors. Unexpected results were shown by predictor P0 whose performances are better than those of P1, P2 and P4. This is due the fact that P1, P2 and P4 involve previous step points which are relatively too far from the next stage points. Since we are using one inner iteration, more inner iterations might give similar behavior for the predictor.

We also solved the problem with B-ordering on several tolerances and the results are recorded in Table 2. We can clearly see that predictor P3 is superior to the others. Due to excessive function evaluations, some tests were not conducted. These are shown by the star (*) symbol entries in the tables.

In order to see the speed-up and efficiency of the code, we performed some tests on both the dense problem and Brusselator problems. The results, recorded in histogram charts, are displayed in Fig. 1 and 2. Here p_0, p_1, p_2, p_3, p_4 denotes the type of predictors used, the two-digits number 13 and 14 denote IRK methods of Radau type, vs_{xy} are the two-digits numbers representing variable stepsize MRK methods, the accuracy digits D is the minimum digits accuracy achieved at the end point of integration.

Each figure consists of two sub-figures (A) and (B), each of which consists of three charts and each charts represents speed-up, efficiency and digits accuracy, respectively. Figure (1A) shows the results on the dense problem with m (problem size) equal to 500. while Fig. (1B) with m equal-750. It is obvious that the speed-up achieved by the 2 processor method ranges from 1.6-2.4 of the three processor methods, while IRK Radau three (R13) and four (R14) stage methods achieved speed-ups 2.3 and 3.2, respectively. The efficiency obtained by the methods ranges from 79-85%, whilst IRK Radau method achieved 60-80% efficiency. The accuracy obtained by the methods of both MRK and IRK type ranges from 4.7-8.9 digits, however each of predictors performed differently.

Methods with two-processor (two stages) achieved a speed-up of about 1.5, efficiency ranges from 75-81%, digits accuracy 5-6.8 digits and a poor accuracy was provided by method V42 with predictor P3. Methods with three processor implementation (three stages) obtained a speed-up about 2.2, efficiency ranges from 70-75%, digits accuracy ranges from 6.1-8.2. The IRK Radau achieved a speed-up of about 2.5 and 3.5 for three and four stage, respectively and efficiency from 76 to 82 and 65-75%, respectively and accuracy various from 5.5- 8.8 digits accuracy.

Figure (2A) and (2B) describe the results on BRUS-0 problem with N = 20 and N = 30, respectively (recall the dimension is $2N^2$). It is obvious that all three-stage methods obtain a speed-up of 1.7-2.1, efficiency between 60-70% and D about 6 decimal digits. The two-stage method attains a speed-up of about 1.5, efficiency between 60-70% and accuracy between 4-7 decimal digits and the IRK-Radau method of four-stages attains a speed-

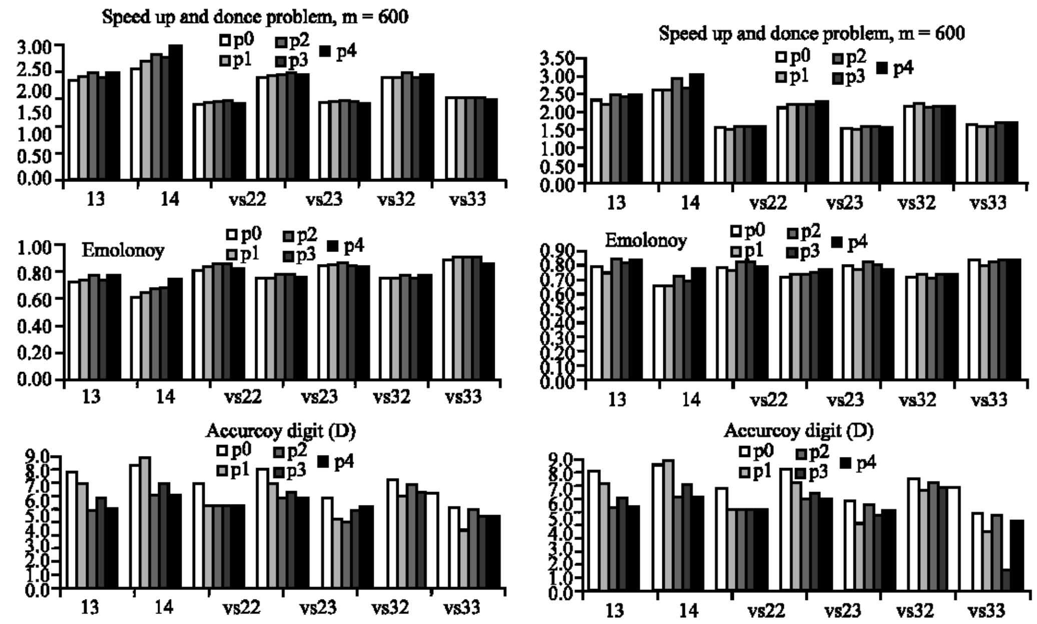

Fig. 1: The results on the dense problems (on the SGI power challenge)

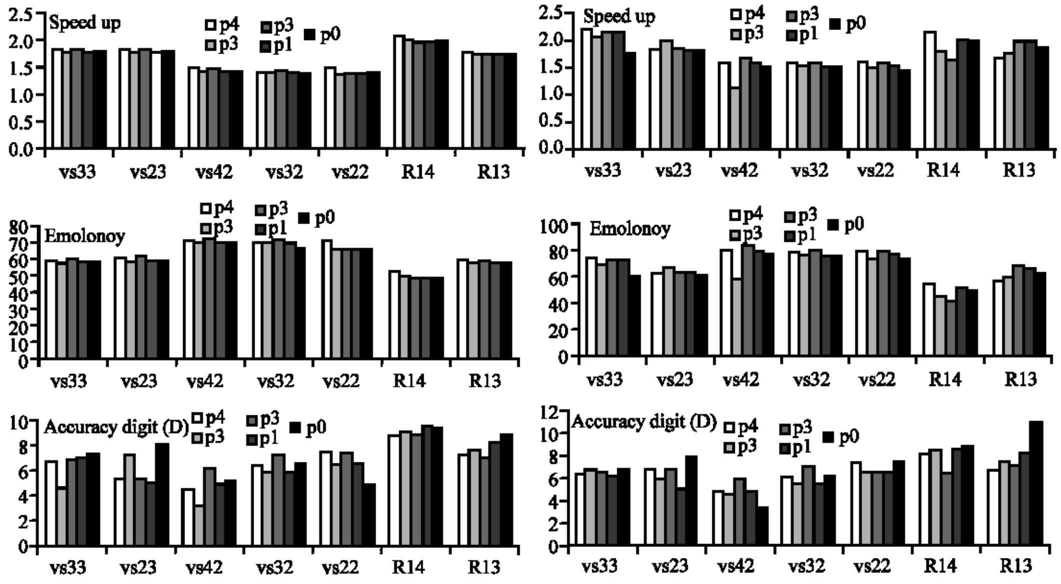

Fig. 2: The results on BRUS-0 problem, N = 20 and N = 30 (on the SGI power challenge)

up between 1.6-2, efficiency about 50% and accuracy between 6-9 decimal digits. We see that, in comparison with the previous Figure 1, there is a decrease in the speed-up obtained on the BRUS-0 problem. This is one of the effects of using numerical differencing to approximate the Jacobian. When function evaluations are costly, it will contribute significant overheads to the whole process. This addresses the need for possible parallelism in the Jacobian approximation, which will be investigated further in later research.

To compare the performance of the code to those of VODE and the parallel iterated IRK-Radau type, we tested the code on both Brusselator BRUS-0 and BRUS-1 with various sizes of the systems. B-ordering is used and the computation is done at tolerance 10^{-8}. Since a numerical Jacobian is imposed in our codes, we set a flag MF = 25 prior to calling VODE. This flag forces VODE to generate numerical banded Jacobian approximations. The timing required by sequential VODE, parallel VS33 method with three processors and IRK-Radau R14 with four processors

Table 3: Timing required by VS33, R14 and Vode on Brusselator BRUS-0 and BRUS-1 (on the SGI ORIGIN, 2000)

	NP	N = 10	N = 15	N = 20	N = 25	N = 30	N = 35	N = 40
BRUS-0								
VS33	3	2.05	5.44	12.41	24.81	44.68	70.84	109.85
VODE	4	2.06	5.97	14.15	25.83	47.98	74.27	114.13
	1	1.27	4.25	10.47	21.81	42.24	74.09	120.51
BRUS-1								
VS33	3	14.07	22.93	38.49	61.14	110.01	182.93	292.76
R14	4	13.17	22.34	35.68	61.52	108.21	176.14	300.28
VODE	1	13.37	24.53	40.40	92.13	199.39	300.56	496.04

Table 4: Staristics of VS33 and VODE at BRUS21, N = 10 and N = 50

N	MF	NSTEP	NREJ	FEVALS	JEVALS	LU
10	VS33	15	0	119	7	15
	VODE	93	2	196	2	15
50	VS33	16	0	127	9	16
	VODE	94	2	517	2	15

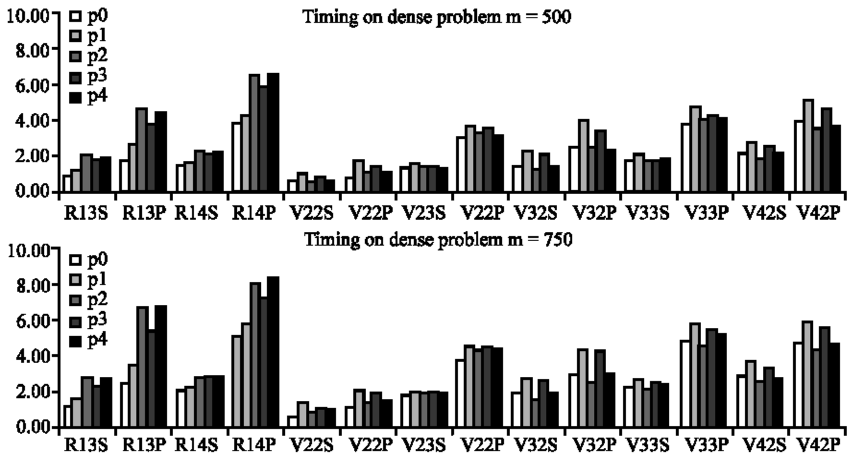

Fig. 3: The timing of Vode relative to the timing of VSMRK on the SGI power challenge

are given in Table 3. Here NP denotes the number of processors used. It is obvious that as the size increases the VS33 method is becoming superior to R14 and VODE.

Note that the results in Table 3 are influenced by strategies in terms of computing the Jacobian matrices and factorizing the Newton iteration matrix. In the code, the old Jacobian will be reused when the error (differences) of the stages vectors is less then 0.001 and the old LU factors will be used when reuse of the Jacobian occurs and the ratio of stepsize changes lies between 1.0 and 1.2. VODE, however, computes the new Jacobian at 50 steps after the last evaluation, or when the Newton iteration failed with outdated Jacobian and the relative changes of the new and old coefficient ($rc = h/l_1$) of the method less than 0.2, or Newton iteration failed with current Jacobian or with a singular matrix.

In addition, the factorization is done at every 20 steps after the last factorization or when rc is greater than 0.3. As we will see, if the cost of the function evaluations is expensive, even though codes have similar number of Jacobian evaluations and factorizations, the timing required might be different. For this we display the statistics required to solve the BRUS-1 with N = 10 and N = 50 with the method VS33 and VODE in Table 4 with the same tolerances.

Note that at N = 10, the method VS33 has more Jacobian evaluations than VODE and an equal number of factorizations to those of VODE. As the problem becomes larger then the function evaluations are more costly and this contributes to the time spent. At N = 50, due to the large number of function evaluations required by VODE, VODE requires more time than VS33. This phenomena is more obvious when the dense problem is solved and these results are given in the Fig. 3.

Figure 3 shows the timing comparison of our methods to VODE. It plots the relative values of the time spent by VODE to the time spent by our methods. Values greater than one indicate that our methods are more efficient than VODE. Here, R1×S and R1×P indicate methods of IRK Radau type, which run with one and x processors, respectively, VxyS and VxyP indicate VMRK methods which run with one and y processors, respectively. It is obvious that on the dense problem all parallel execution and some sequential process of the code are more efficient than VODE.

CONCLUSION

Parallel codes for solving stiff IVPs for ODEs have been developed. The code provides options to the user to choose the desired methods either-VMRK or IRK Radau type. Tests on sparse and dense problems have been conducted. The code shows significants efficiencies, these are shown by both VMRK and IRK methods of Radau type. Consistent digits accuracy were obtained by the method V42 and V33. In term of time consumed by the methods, they show superiority on dense problems over standard VODE code, even in some sequential processes. For sparse Brusselator-like problems, they started to outperform VODE and IRK-Radau at very large problem sizes.

Our investigation on the time spent on all the tested problems shows that the LU factorization of the linear system matrices dominates the whole process, consuming about 50% of the time, while the backward substitutions required to solve the system consume about 30% of the time and the function evaluations only spent less than 10% of time. These indicate that the global parallelism on function evaluations introduced in Suhartanto and Burrage (2003) will not significantly speed-up the process unless the same technique be applied to the LU factorization and backward substitutions. The latter however is not possible to use due to the fact that the current parallelism feature of the Fortran 90 compiler does not support this. However, recoding the parallel numerical algebra routines as for the function evaluations might solve the problem.

In addition to the above mentioned global parallelism technique, some possible future development of the codes include techniques to minimize the use of memory consumption and the time required in solving the linear algebra parts.

REFERENCES

Burrage, K., 1995. Parallel and Sequential Methods for Ordinary Differential Equations. Oxford University Press, New York.

Burrage, K., C.C. Eldershaw and R.B. Sidje, 1997. A Parallel matrix free implementation of a Runge Kutta code. In Numerical Mathematics, IMACS Series in Computational and Applied Math., II: 45-50.

Burrage, K. and H. Suhartanto, 1997. Parallel iterated method based on Multistep Runge-Kutta of Radau type for Non Stiff Problems. Adv. Comput. Math., 7: 59-77.

Burrage, K. and H. Suhartanto, 2000. Parallel iterated method based on Variable stepsize Multistep Runge-Kutta. Adv. Comput. Math., 13: 257-270.

Hairer, E., 1997. Testset in Geneva, http://www.unige.ch/math/folks/hairer/testset/.

Schneider, S., 1993. Numerical experiments with a Multistep Runge Kutta method. BIT 33: 332-350.

Suhartanto, H., 1998. Parallel Iterated Techniques based on Multistep Runge-Kutta Methods of Radau Type, PhD. Thesis, Department of Mathematics, The Univesity of Queensland, St. Lucia, QLD, Australia,

Suhartanto, H. and Burrage, 2003. K. NSMRK: A Parallel Implementation of Multistep Runge Kutta methods for non Stiff ODEs. Jurnal Ilmu Komputer dan Teknologi Informasi., 3: 20-30.