INFORMATICA, 1994, Vol. 5, No. 3–4, 373–384

A PARALLEL ALGORITHM FOR STIFF ORDINARY DIFFERENTIAL EQUATIONS

Dana PETCU

Department of Computer Science, Faculty of Mathematics
University of Timişoara
1900 Timişoara, B-dul Vasile Pârvan 4, Romania

Abstract. The problem associated with the stiff ordinary differential equation (ODE) systems in parallel processing is that the calculus can not be started simultaneously on many processors with an explicit formula. The proposed algorithm is constructed for a special classes of stiff ODE, those of the form $y'(t)=A(t)y(t)+g(t)$. It has a high efficiency in the implementation on a distributed memory multiprocessor when the ODEs function has many components. The approximation error is equal to that produced by the analogous sequential algorithm.

Key words: parallel numerical algorithm, stiff ordinary differential equations, implementation efficiency, distributed memory multiprocessor.

1. Introduction. The subject of this paper is the parallel computing of the solution of stiff ordinary differential equations systems. The main problem associated with a stiff system is that of the stability. Explicit numerical methods are not suitable for the integration. More details about the stiff system can be found in Hairer and Wanner (1991).

Parallelism in solving ODE can be expressed via three distinct *avenues* (Iserles and Norsett, 1990):

(1) coding a specific method so that it can be performed simultaneously on several processors,
(2) splitting variables in a multivariable ODE system between processors,
(3) exploiting parallelism in performing the request computer algebra, solving linear and nonlinear algebraic systems of equations.

The proposed algorithm is a hibrid method. It can be applied to linear stiff system with variable coefficients, divides the responsibilities of the processors on the interval of integration (second avenue), produces the same error like the sequential algorithm and makes use of the linearity of the system (third avenue). The efficiency depends on the complexity of the system function. The degree of parallelism depends on the dimension of the system.

The numerical results show that a high efficiency can be obtained for a reasonable system dimension.

2. Previous work. The parallel methods from the first avenue preserve the stability and accuracy of the basic sequential algorithm. Such methods are the predictor–corrector schemes which can be easily implemented in a parallel mode. Many examples are presented by Miranker and Liniger (1967). The application of those to stiff system is not successfully because almost all schemes are equivalent with some explicit formulas. The stability characteristics can be improved. In this direction we can mention the paper of Worland (1993), Ghoshal and *et al.* (1989). These improvements make possible to handle slightly stiff equations.

The block methods are easily adapted to a parallel mode with no degradation in the accuracy of solution. The performance of Hutchinson and Kalaf parallel implementation (1991) is dependent on the number of scheme nodes. Chu and Hamilton's (1978) algorithm has a numerical efficiency dependent on the dimension of the solving system and on the complexity of the system function.

Iserles and Norsett (1990) investigate the degree of the parallelism of the Runge–Kutta methods. The parallelism depends on exploitation of the sparsity structure of the Runge–Kutta matrix. Karakashian and Rust's algorithm (1989) is designed to solve a linear system of ODE by a Runge–Kutta process. The efficiency of this algorithm depends on the system dimension. For the proposed test system with variable dimension, the parallel mode becomes competitive only for a number of components of hundreds order. The parallel Runge–Kutta method proposed by Evans and Sanugi (1989) makes also use of the special form of the method matrix.

Galligani and Ruggiero (1989) propose a parallel method which is suitable to be applied to a large set of linear ODE systems. The method has good stability properties which makes it applyable to the stiff systems.

One parallel algorithm which subscribe also to the first avenue is proposed by Petcu (1994).

The second way request a reconstruction of the class of numerical methods for ODE. The basic idea is that of Nivergelt (1964). The integration interval is divided into equal subintervals and each processor is responsible for integrate the solution on only one subinterval. To make efficient the algorithm, the processors must be started at appropriate time and must work simultaneously. For this purpose it is used an starting value, an approximation of the solution at the beginning of the processor subinterval. In the special case of a stiff system, the starting value can not be obtained be an explicit method, only when the subintervals number and the step size are very smalls. In the case of using an implicit starting formula, we reach the class of block methods. An other problem associated with the stiff case is that the approximative solution produced by the parallel algorithm can be unstable, also when the basic method for integration on each subinterval has good stability properties. The error of the approximative parallel solution is different from that of sequential solution.

Knirsch (1992) studies a scheme for solving the stages algebraic equations of a Runge–Kutta method (the third way). The reason for unsuccessfully using of this scheme in the stiff case is the same like for Nivelgelt's method.

3. The problem to be solved. We consider a stiff system of the form

$$y'(t) = A(t)y(t) + g(t), \quad t \in [0, T], \tag{1}$$

with the initial condition $y(0) = y^0$. Note with D the dimension of this system. For the numerical integration, the integration interval is divided in N equals subintervals of h length.

The class of such stiff systems is not empty. For example, Iserles

(1981) proposed the stiff system

$$\begin{cases} y_1'(t) = -\left(80 + \frac{1}{3(1+t)}\right) y_1(t) - \left(40 - \frac{2}{5(1+t)}\right) y_2(t), \\ y_2'(t) = -\left(40 - \frac{2}{5(1+t)}\right) y_1(t) - \left(20 + \frac{4}{5(1+t)}\right) y_2(t), \end{cases}$$

$$y(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad T = 100. \tag{2}$$

4. The basic method. Applying the implicit Euler rule

$$y_{n+1} = y_n + h f_{n+1}, \quad n = 0, \ldots, N_1, \tag{3}$$

we get

$$[I - hA(t_{n+1})]\, y_{n+1} = y_n + hg(t_{n+1}).$$

The step n of the sequential algorithm consists on the following stages:

(1) Evaluate $A(t_{n+1})$ and $g(t_{n+1})$.

(2) Solve the linear system. If we use a Gauss like procedure, the principal steps in solving a linear system $Qx = b$ are:

(a) Transform Q to an upper triangular form.

(b) Transform b in the same manner.

(c) Solve the upper triangular linear system in x.

5. Splitting the computational effort. We dispose of a distributed memory multiprocessor with P processors connected in a circular network. We choose two integer values K and r so that

$$KPr = N = T/h, \tag{4}$$

and make the distribution of the points of the integration interval like in Fig. 1. The idea is to compute at each k stage, (1) and (2a) in parallel, and (2b) and (2c) in a serial mode.

6. Outline the algorithm. There are K stages. In stage k, where $k = 0, \ldots, K-1$, the processor p $(p = 0, \ldots, P-1)$ must execute the following:

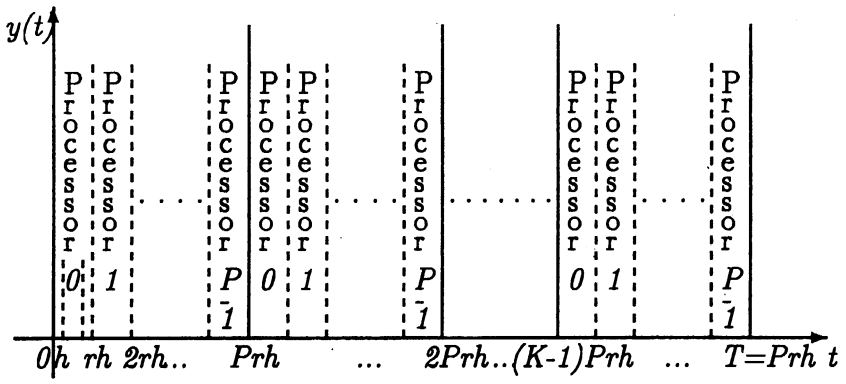

Fig. 1. Responsibility of the processors.

(1) for $j = 1, \ldots, r$:
  (a) evaluate $A(t_{krPh+prh+jh})$ and $g(t_{krPh+prh+jh})$;
  (b) find the matrix C_j so that $C_j[I - hA(t_{krPh+prh+jh})]$ to have a triangular upper form;

(2) if $p \neq 0$ or $p = 0$ and $k \neq 0$, receive from the processor $(p-1) \bmod P$ the vector y_{krP+pr}, else the entry vector is y_0;

(3) for $j = 1, \ldots, r$:
  (a) transform the entry vector by applying C_j;
  (b) find the solution of the transformed system $y_{krP+pr+j}$;

(4) if $p \neq P-1$ or $p = P-1$ and $k \neq K-1$, send $y_{krP+(p+1)r}$ to processor $p+1 \bmod P$ and set $k \rightarrow k+1$, else stop algorithm.

7. Study of the efficiency. Note T_p the execution time of an implementation of the algorithm using p processors, T_1 the execution time of the same implementation of the algorithm using only one processor and T_0 the execution time of the best implementation of the basic sequential algorithm.

The efficiency of the parallel algorithm may be computed in many ways. The numerical efficiency of the parallel algorithm , E_{num}, is computed by the following formula

$$E_{num} = \frac{T_0}{T_1}. \tag{5}$$

The efficiency of the parallel implementation of the algorithm, E_p^{par}, is given by the next relationship

$$E_p^{par} = \frac{T_1}{pT_p}. \tag{6}$$

The efficiency of the parallel algorithm with p processors, E_p, is given by the formula

$$E_p = \frac{T_0}{pT_p} = E_{num} E_p^{par}. \tag{7}$$

The ideal values of these efficiency measurements are 1. The practical values are lowest: for E_{num}, because almost all sequential algorithms are very difficult to divide in a number of equal units (in the sense of the same computing effort), for E_p^{par}, because the communication time between the processors is greater that the computing time necessary for an arithmetic operation (the ratio is between 500 and 1000, depending on the network), and, for E_p, because $E_{num} \leqslant 1$ and $E_p^{par} \leqslant 1$.

Let F be the medium time necessary to evaluate $A(\cdot)$ and $g(\cdot)$, U, the medium time necessary to determine one transforming matrix C, V, the medium time necessary to transform a D-dimensional vector according the matrix C and to find the solution of the transformed upper triangular system, S, the medium time necessary to send a D-dimensional vector to a processor directly linked with the current processor, R, the medium time necessary to receive a D-dimensional vector from a processor directly linked with the current processor, and G, the medium time necessary to solve a linear system of dimension D with the standard Gauss procedure.

The first stage of the algorithm look like in Fig. 2. With a continue line we have note the time when a processor is busy.

The condition that the 0-processor does not wait until the $P-1$ processor finishes the first stage is

$$\begin{aligned} &r(U+F)+rV+S+r(U+F) \\ &\quad \geqslant r(U+F)+(P-1)(rV+S)+(P-2)R+R+rV+S, \end{aligned}$$

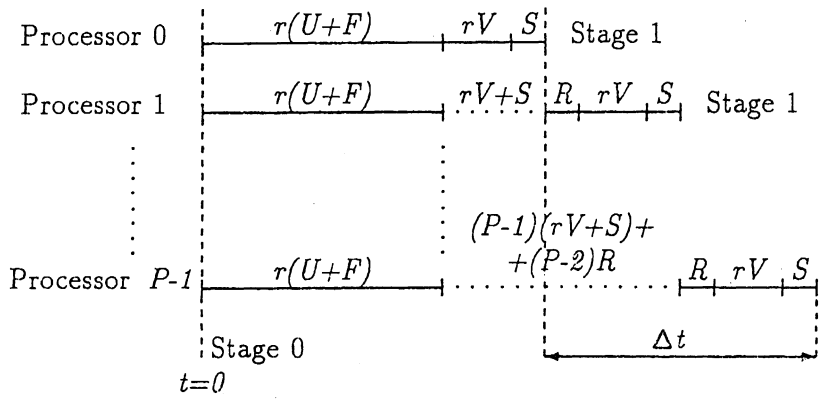

Fig. 2. The first stage, in time.

that means

$$P-1 \leqslant \frac{U+F}{V+(S+R)/r}, \tag{8}$$

or

$$r \geqslant \frac{(S+R)(P-1)}{U+F-(P-1)V}, \quad U+F \geqslant (P-1)V. \tag{9}$$

If these inequalities are satisfied, then there are not waiting times in the stage $k > 0$ (see Fig. 3).

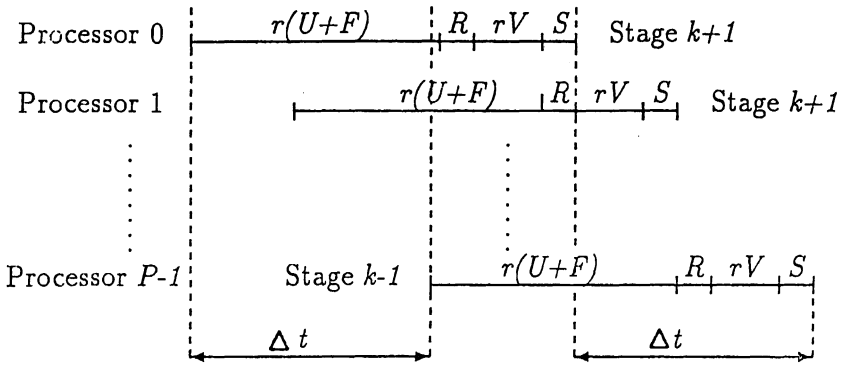

Fig. 3. The stage $k > 0$, in time.

Theoretically, we have the following equations:

$$\begin{aligned} T_p =& K\big[r(U+F)+rV+S+R\big] \\ &+(P-1)(rV+S)+(P-2)S, \\ T_1 =& PK\big[r(U+F)+rV\big] = PKr(U+F+V), \end{aligned}$$

and then

$$E_p^{par} = \frac{T_1}{PT_p} = \frac{1}{1+\frac{K(R+S)+(P-1)(rV+S)+(P-2)S}{Kr(U+F+V)}}. \tag{10}$$

Note that

$$T_1 = T_0 + N(U+V-G).$$

REMARKS:

(i) $S \approx R$ and does not depend on D;

(ii) U, F and V depend superlinear on the system dimension. In addition, F depends on the system function complexity.

(iii) The stages number K has influence on the efficiency. For system with a small D, K must be small, because $U+F \approx S$.

(iv) The approximation error produced by the parallel algorithm is equal to the one produced by the sequential algorithm.

(v) To maximize E_p^{par}, for a given system (for which we know the values S, T, U, F) and a given number of integration steps, N, we search (P,K,r) for which

$$\min_{(K,P,r)\in A} \frac{K(S+R)+(P-1)(rV+S)+(P-2)R}{Kr(U+F+V)}, \tag{11}$$

where

$$\begin{aligned} A =& \big\{(K,P,r) \in N^3 \big| r\big[U+F-(P-1)V\big] \\ &\geqslant (S+R)(P-1),\ PKr = N\big\}. \end{aligned}$$

(vi) For simple system function, $F \ll U$ and $F \ll V$, and $U/V \leqslant (D+1)/4$. The condition that the 0-processor has not dead times implies that

$$p \leqslant \frac{D+3}{2}.$$

8. Numerical results. We have tested the proposed algorithm on the system (2) and on the following system

$$y_i'(t) = -i^5 y_i(t), \quad t \in [0,1], \quad i = 1, \ldots, D. \tag{12}$$

This system generalized in D dimension the stiff system $A4$ given by Enright and *et al.* (1975), for which $D = 10$. The test results are presented in Tables 1–2. We have use a T-800 mutiprocessor under PARIX.

Analyzing Table 1 we conclude that the efficiency of the parallel implementation of the algorithm increases with the dimension D and descreases with the processors number P. For $P = 2, E_P^{par}$ is very close to the ideal value.

From Table 2 we deduce that all the measurements of the efficiency increase with the number N of the points where the solution of the ODE system is numerical evaluated.

The complexity of the system function has a significant influence on the numerical efficiency (compare the values $P = 2$ from both tables). For the test systems, the value of E_{num} is close to 0.8.

The numerical efficiency is the principal value which has a great influence on the efficiency of the parallel algorithm. Hence, E_P depends on D, P and on the complexity of the system function.

9. Conclusions. The proposed parallel algorithm is designed to solve linear stiff systems of ODE with variable coefficients and a perturbation depending on time. The approximation error is the same like in the sequential algorithm. The efficiency of the parallel algorithm increases with the dimension and the complexity of the system function and decreases with the number of used processors.

Table 1. The efficiency in the integration of system (12) with $N = 1000$

	E_p^{par}			E_p			E_{num}
$D\backslash P$	2	4	8	2	4	8	
2	0.962	0.500	0.260	0.683	0.355	0.185	0.710
3	0.973	0.610	0.303	0.700	0.440	0.218	0.720
4	0.978	0.667	0.331	0.703	0.476	0.236	0.714
5	0.983	0.712	0.355	0.700	0.508	0.253	0.713
8	0.987	0.838	0.419	0.709	0.602	0.301	0.719
10	0.989	0.908	0.456	0.716	0.657	0.330	0.724
15	0.992	0.919	0.549	0.730	0.677	0.405	0.737
20	0.993	0.926	0.634	0.742	0.691	0.473	0.747

Table 2. The efficiency in the integration of the Iserles's system (2) with dimension $D = 2$ and $P = 2$ processors

$\lg(N)$	E_2^{par}	E_2	E_{num}
1	0.644	0.453	0.704
2	0.700	0.550	0.779
3	0.890	0.703	0.789
4	0.962	0.760	0.790

Acknowledgement. The author is grateful to the peers of the TEMPUS program for providing the opportunity to work on this paper at the University of Heidelberg.

REFERENCES

Chu, M.T., and H.Hamilton (1978). Parallel solution of ODE's by multi-block

methods. *SIAM J. Sci. Statistic Comput.*, **27**, 413–420.

Enright, W.H., T.E.Hull and B.Lindberg (1975). Comparing numerical methods for ordinary differential equations. *BIT*, **15**, 1–48.

Evans, D.J., and B.B.Sanugi (1989). A parallel Runge–Kutta integration method. *Parallel Computing*, **11**, 245–251.

Hutchinson, D., and B.M.S.Khalaf (1991). Parallel algorithms for solving initial value problems: front broadening and embedded parallelism. *Parallel Computing*, **17**, 957–968.

Galligani, I., and V.Ruggiero (1989). Solving large systems of linear ordinary differential equations on a vector computer. *Parallel Computing*, **9**, 359–365.

Ghoshal, S.K., M.Gupta and V.Rajarman (1989). A parallel multistep predictor-corector algorithm for solving ordinary differential equations. *Journal of Parallel and Distributed Computing*, **6**, 636–648.

Hairer, E. and G.Wanner (1991). *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.* Springer–Verlag, Heidelberg. 540pp.

Iserles, A. (1981). Quadrature methods for stiff ordinary differential systems. *Mathematics of Computing*, **36**, 171–182.

Iserles, A., and S.P.Norsett (1990). On the theory of parallel Runge–Kutta methods. *IMA Journal of Numerical Analysis*, **10**, 463–488.

Karakashian, O.A., and W.Rust (1989). On the parallel implementation of implicit Runge–Kutta methods. *SIAM Review*, **31**, 1023–1028.

Knirsch, R. (1992). A parallel implicit Runge–Kutta method. In G.Bader, R.Rannacher, G.Wittum (Eds.), *Parallel Solution Methods for Differential Equations*, Preprint Nr. 698, Universität Heidelberg, Stochastiche Mathematische Modelle.

Miranker, W.L., and W.Liniger (1967). Parallel methods for the numerical integration of ordinary differential equations. *Mathematics of Computation*, **21**, 303–320.

Nivergelt, J. (1964). Parallel methods for integrating ordinary differential equations. *Communications of the ACM*, **7**(12), 731–733.

Petcu, D. (1994). *Numerical Methods for Solving the Stiff Differential Systems.* Ph.D. Thesis, University of Timişoara (in Romanian).

Worland, P.B. (1993). Parallel methods for ODEs with improved absolute stability boundaries. *Journal of Parallel and Distributed Computing*, **18**, 25–32.

Received September 1994

D. Petcu born in 1966, received the M.S. degree in informatics from the Mathematics Faculty of the University of Timişoara and the Ph.D. degree in numerical analysis in 1994. At present she is lecturer in the above mentionated institute. Her research interests include the numerical analysis, especially the ordinary differential equation, as well the parallel computing.