Numerical Solution of ODEs with Distributed Maple

Dana Petcu

Western University of Timişoara, Computer Science Department
B-dul V.Pârvan 4, RO-1900 Timişoara, Romania,
tel./fax.:++40-56-194002, e-mail: petcu@info.uvt.ro,
home page on web: http://www.info.uvt.ro/~petcu

Abstract. We describe a `Maple` package named `D-NODE` (Distributed Numerical solver for `ODE`s), implementing a number of difference methods for initial value problems. The distribution of the computational effort follows the idea of parallelism across method. We have benchmark the package in a cluster environment. `Distributed Maple` ensures the inter-processor communications. Numerical experiments show that parallel implicit Runge-Kutta methods can attain speed-ups close to the ideal values when the initial value problem is stiff and has between ten and hundred equations. The stage equations of the implicit methods are solved on different processors using `Maple`'s facilities.

Keywords: parallel numerical methods for ordinary differential equations, distributed computer algebra systems, performance analysis.

1 Introduction

We will concerned with the numerical solution of systems of initial value ordinary differential equations (IVPs for ODEs) of the form

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0 \tag{1}$$

where $y_0 \in R^m$ and $f : R \times R^m \to R^m$.

In the numerical solution of ordinary differential equations by implicit time-stepping methods a system of linear or nonlinear equations has to be solved each step. The costs of the linear algebra associated with the implementation of the implicit equation solver generally dominate the overall cost of the computation.

The numerical integration of large IVPs is also time consuming. Such large (and stiff) problems often arise in the modeling of mechanical and electrical engineering systems or in the solution of semi-discretization of convection-diffusion problems [7] associated to time-dependent parabolic PDEs of the form

$$\frac{\partial u}{\partial t} + \alpha(x,u)\frac{\partial^2 u}{\partial x^2} + \beta(x,u)\frac{\partial u}{\partial x} + \gamma(x,u)u = g(t,x). \tag{2}$$

The stiffness of these problems requires that the numerical methods to be used should be unconditionally stable, and therefore implicit. The methods are computationally demanding and require today's fastest high performance computers for practical implementations. However, access to a fast high-speed computer is not sufficient. One must also ensure that the great potential power of the computer is correctly exploited. If the access to a fast high-speed computer is not possible, a secondary option is the use of a local network of workstations.

The aim of this paper is to investigate in what extent parallel implicit Runge-Kutta methods can be used to solve stiff initial value problems of ten to hundred equations using `Distributed Maple`. Stage systems to be solved are distributed among the processors of a cluster system. Tables and figures illustrate the performance of the implemented methods.

The paper is organized as follows. Section 2 motivates the present work. Section 3 describes the objectives of a `Maple` package named `D-NODE` (Distributed Numerical solver for `ODE`s), implementing a number of difference methods designed in the idea of parallelism across method. In Section 4 we report the numerical results obtained by four known parallel implicit Runge-Kutta methods. The test problems are a linear ODE system, a nonlinear ODE system arising from the semi-discretized nonlinear convection-diffusion problem of the form (2), and a nonlinear ODE system arising from celestial mechanics. We have benchmark the package in a cluster environment. The cluster comprises 4 dual-processor Silicon Graphics Octanes (2 R10000 at 250 MHz each) linked by three 10 Mbit Ethernet subnetworks connected to a router. Similar IVPs have been studied in [7] for the case of a shared-memory parallel computer.

2 IVP solving strategies

2.1 Implicit solvers

Among the numerical techniques commonly considered for the efficient solution of stiff initial value ordinary differential equations are the implicit Runge-Kutta schemes (IRK). The use of an s-stage implicit method for ODEs requires the solution of nonlinear systems of algebraic equations of dimension sm, where m is the size of the differential problem. Usually, the solution of this system represents the most time-consuming section in the implementation of such method.

The single application of many implicit schemes for IVPs of the form (1) results in the solution of a system of algebraic equations of the form

$$Y - h(C \otimes I_m)F - G(Y_0) = 0 \tag{3}$$

where h is the step-size, C is a full $s \times s$ matrix, I_m is the identity matrix of size m, G is a known function and $Y = (y_1, \ldots, y_s)^T,\ F = (f_1, \ldots, f_s)^T,\ f_i = f(t_i, y_i)$ are the vectors with the unknown approximations to the solution of the grid-points $t_1, \ldots, t_s$. It is common practice to use fixed-point iterations or, in the stiff case, some modified Newton iterations. The convergence rate of such methods depends on the method step-size.

2.2 Parallel solvers

A general way of devising parallel ODE solvers is that of considering methods whose work per step can be split over a certain number of processors. The so-called solvers with parallelism across the method are then obtained. Such methods are essentially Runge-Kutta schemes. For the parallelization of implicit Runge-Kutta methods a parallel computation of stages is obvious: then the system (3) can be split into a number $k \leq s$ independent subsystems.

From the computational point of view, the diagonally implicit RK methods (DIRK methods) are the most attractive methods since they have suitable stability properties and the implementation can be carried out with a lower computational cost than fully IRK methods.

The so-called PDIRK methods are parallel diagonally iterated RK methods. The computational cost involved in their implementation is similar to DIRK methods. PDIRK methods are able to produce accurate results at a relatively high price. As a result, these methods are not the most suitable for solving semi-discretized PDEs in which it is necessary to generate relatively low-accuracy results at low price [2]. The construction of a some PDIRK using `Maple` is presented in [4].

2.3 Distributed Maple

Computer algebra systems (CAS) can be used with success in prototyping sequential algorithms for symbolic or numeric solution of mathematical problems, in particular for ODEs. `Maple`, widely used environment for scientific computing, is such a CAS. Constructing prototypes for parallel algorithms in `Maple` for numeric solution of ODEs is a challenging problem.

`Distributed Maple` [13] is a portable system for writing parallel programs, in a computer algebra system, which allows to create concurrent tasks and have them executed by `Maple` kernels running on different machines of a network. The system can be used in any network environment where `Maple` and `Java` are available. It provides message passing facilities via a global heap. The user interacts with the system via the text oriented `Maple` front-end. It also provides facilities for the online visualization of load distribution and for post-execution analysis of the dynamic behavior of a session.

2.4 Network of workstations in IVP solving

We know that solving systems of algebraic or differential equations of order several hundreds can be an unsolvable problem for an actual CAS. Systems of order several tens equations can be solved with `Maple` but the long running-time may be a great problem for the user. A correct use of extensions like `Distributed Maple` can improve the solution computation time. The computer facilities required by such an extension are reasonable for any user since it not supposes access to super-computers.

We have recently present [12] some test results concerning the distributed computation of ODE solutions of very large IVPs in distributed environments.

2.5 Numeric versus symbolic computations

Combining numeric and symbolic computations can be the best way to solve a nonlinear IVP. A CAS offers the opportunity to combine them. Exploiting this opportunity we propose in the present paper a modification of the standard implementation of a parallel RK method in a message-passing computing environment.

In general, the system (3) is solved numerically using repeated evaluation of the function F at different values (in the case of a stiff system, they are also required some repeated Jacobian matrix evaluations). In a message-passing computing environment these values must be communicated between different processors participating to a time-step integration of the IVP. Sending to a working processor the algebraic expressions of the part of F for which it is responsible can be a better solution since we eliminate a significant quantity of function values to be communicated between the supervisor-processor which stores F-function and the worker-processors. The interpretation of an algebraic expression (eventual also a Jacobian matrix evaluation) requires at a worker processor side at least a small specific expression interpreter. If the worker process is based on a `Maple` kernel, that is not a problem.

Note also that the implicit equation solver can substantially affect the global error of the numerical solution of an IVP. Take for example the fixed-point iterations which usually do not converge in the stiff IVP case to the exact solution of the system (3). Using a fixed-point iterations and ignoring this remark and also the use error control strategies, we can obtain a numerical solution far from the real solution.

Note that in the practical implementation of implicit time-stepping methods the hardest parts are the implicit equation solver implementation and the error control mechanism combined with variable step-size strategies. Using numerical facilities of CAS systems to do the first job it can simplify the programmer work.

We propose the use of implicit equation solver of `Maple` for the solution of system (3). In the case of parallel IRK, independent stage-subsystems in `Maple` algebraic form are to be send to some worker-processors in order to solve them.

3 D-NODE objectives

We shortly present the project of a `Maple` package, `D-NODE` (Distributed `Numeri`cal solver for `ODE`s), an update to the `ODEtools` `Maple` package. It implements a number of difference methods designed in the idea of parallelism across method [16]. The package is a part of a bigger project of an expert system for numerical solution of ODEs [11] and it is expected to be finalized at the end of this year.

The facilities of `ODEtools` from `Maple`, and the similar tools from other CAS are far to cover all the user needs (for example, the stiff IVP solving case). Recent reports demonstrate the effort to improve these tools. For example, the paper [14] describes mathematical and software developments for a suite of programs for solving ODEs in `Matlab`.

D-NODE package has similar facilities with EpODE (ExPert system for ODEs), recently presented in [9] and [11] and available at http://www.info.uvt.ro/~petcu/epode: a large collection of parallel methods working in a distributed computing environment, automatic detection of method properties including method classification, order, error constant and stability, degree of parallelism, method-interpreter for describing new methods, automatic detection of problem properties (like stiffness), step-size selection mechanism according the method and problem properties, numerical solution computation on a distributed network of workstations (in EpODE based on PVM – Parallel Virtual Machine [8]).

The current collection of parallel methods includes block methods, predictor-corrector methods based on linear multistep methods, block explicit RK methods, diagonally implicit RK methods, singly and multi-implicit RK methods, parallel diagonally iterated RK methods.

4 Numerical experiments

This section is devoted to the interpretation of the test results in the integration of large non-linear ODE systems and to the comparisons with the test results of other similar tools (one of them being EpODE, part of the same project [10])

We consider four methods representative for their class of parallel IRK which were included in D-NODE. We have benchmark the corresponding package functions in a cluster environment. The cluster comprises 4 dual-processor Silicon Graphics Octanes (2 R10000 at 250 MHz each) linked by three 10 Mbit Ethernet subnetworks connected to a router.

Table 1 presents the selected formulae. Formula (5) is a diagonally implicit Runge-Kutta method. As a parallel diagonally iterated Runge-Kutta method we take the formula (6). Formula (7) is a block-diagonally implicit Runge-Kutta method. As a parallel singly diagonally iterated Runge-Kutta method we take the formula (8) (the method coefficients are truncated).

Applying the direct-graph method proposed in [5] we can guess the degree of parallelism and the number of parallel stages of each method without any theoretical background. EpODE automatically detects those method properties. Figure 1 generated by EpODE presents the proposed distributions of the computations on processes and parallel stages for the above mentioned methods. Like EpODE, D-NODE will include a function to detect such process-stages distributions.

In order to show the performance of the methods on semi-discrete PDEs we include in our tests the linear IVP obtained from the following PDE [14]:

$$\frac{\partial u(x,t)}{\partial t} = e^t \frac{\partial^2 u(x,t)}{\partial x^2} \tag{4}$$

with initial and boundary conditions $u(x,0) = \sin(x),\ u(t,0) = u(t,\pi) = 0$.

As the second test problem we take the nonlinear IVP obtained by the semi-discretization of the following nonlinear convection-diffusion problem [1] (particular case for the general problem (2)):

Table 1. Selected methods: a DIRK method, a PDIRK method, a BDIRK method and a PSDIRK method

Method iterative equations and properties

4-stage, 2-processor, 4th-order, A-stable DIRK [6]

$$\begin{aligned} y_{n+1} &= y_n + \frac{h_n}{16}[(9+3\sqrt{57})(k_1+k_2)-(1+s\sqrt{57})(k_3+k_4)] \\ k_1 &= f(t+h/3, y_n+h_nk_1/3) \\ k_2 &= f(t+2h_n/3, y_n+h_n(k_1+k_2)/3) \\ k_3 &= f(t+h_n(21+\sqrt{57})/48, y_n+h_n(21+\sqrt{57})k_3/48)) \\ k_4 &= f(t+h_n(27-\sqrt{57})/48, y_n+h_n(2(3-\sqrt{57})k_3+(21+\sqrt{57})k_4)/48) \end{aligned} \tag{5}$$

6-stage, 2-processor, 3th-order, A-stable PDIRK based on Radau IIA corrector [15]

$$\begin{aligned} y_{n+1} &= y_n + \frac{h_n}{4}(3k_5+k_6) \\ k_1 &= f(t+h_n/3, y_n+h_n((\sqrt{6}-2)f(y_n)+(4-\sqrt{6})k_1)/6) \\ k_2 &= f(t+h_n, y_n+h_n((6-\sqrt{6})f(y_n)+(4+\sqrt{6})k_2)/10) \\ k_3 &= f(t+h_n/3, y_n+h_n((2\sqrt{6}-3)k_1-k_2+2(4-\sqrt{6})k_3)/12) \\ k_4 &= f(t+h_n, y_n+h_n(15k_1-(3+2\sqrt{6})k_2+2(4+\sqrt{6})k_4)/20) \\ k_5 &= f(t+h_n/3, y_n+h_n((2\sqrt{6}-3)k_3-k_4+2(4-\sqrt{6})k_5)/12) \\ k_6 &= f(t+h_n, y_n+h_n(15k_3-(3+2\sqrt{6})k_4+2(4+\sqrt{6})k_6/20)) \end{aligned} \tag{6}$$

4-stage, 2-processor, 4th-order, L-stable Hammer-Hollinworth BDIRK [5]

$$\begin{aligned} y_{n+1} &= y_n + \frac{h_n}{2}(3k_1+3k_2-2k_3-2k_4) \\ k_1 &= f(t+(3-\sqrt{3})h_n/6, y_n+h_n(5k_1+(1-2\sqrt{3})k_2)/12) \\ k_2 &= f(t+(3+\sqrt{3})h_n/6, y_n+h_n((1+2\sqrt{3})k_1+5k_2)/12) \\ k_3 &= f(t+(3-\sqrt{3})h_n/6, y_n+h_n(3k_3-\sqrt{3}k_4)/6) \\ k_4 &= f(t+(3+\sqrt{3})h_n/6, y_n+h_n(\sqrt{3}k_3+3k_4)/6) \end{aligned} \tag{7}$$

9-stages, 3-processor, 4th-order, A-stable PSDIRK [2]

$$\begin{aligned} y_{n+1} &= y_n + h_n(0.9702k_7+0.0307k_8-0.0009k_9) \\ k_1 &= f(t+0.4443h_n, y_n+h_n(-0.6243f(y_n)+1.0686k_1)) \\ k_2 &= f(t+2.4516h_n, y_n+h_n(1.3830f(y_n)+1.0686k_2)) \\ k_3 &= f(t+6.7213h_n, y_n+h_n(5.6527f(y_n)+1.0686k_3)) \\ k_4 &= f(t+0.4443h_n, y_n+h_n(-0.5571k_1-0.0757k_2+0.0085k_3+1.0686k_4)) \\ k_5 &= f(t+2.4516h_n, y_n+h_n(1.4082k_1+0.0167k_2-0.0418k_3+1.0686k_5)) \\ k_6 &= f(t+6.7213h_n, y_n+h_n(0.3786k_1+4.7337k_2+0.5404k_3+1.0686k_6)) \\ k_7 &= f(t+0.4443h_n, y_n+h_n(-0.5571k_4-0.0757k_5+0.0085k_6+1.0686k_7)) \\ k_8 &= f(t+2.4516h_n, y_n+h_n(1.4082k_4+0.0167k_5-0.0418k_6+1.0686k_8)) \\ k_9 &= f(t+6.7213h_n, y_n+h_n(0.3786k_4+4.7337k_5+0.5404k_6+1.0686k_9)) \end{aligned} \tag{8}$$

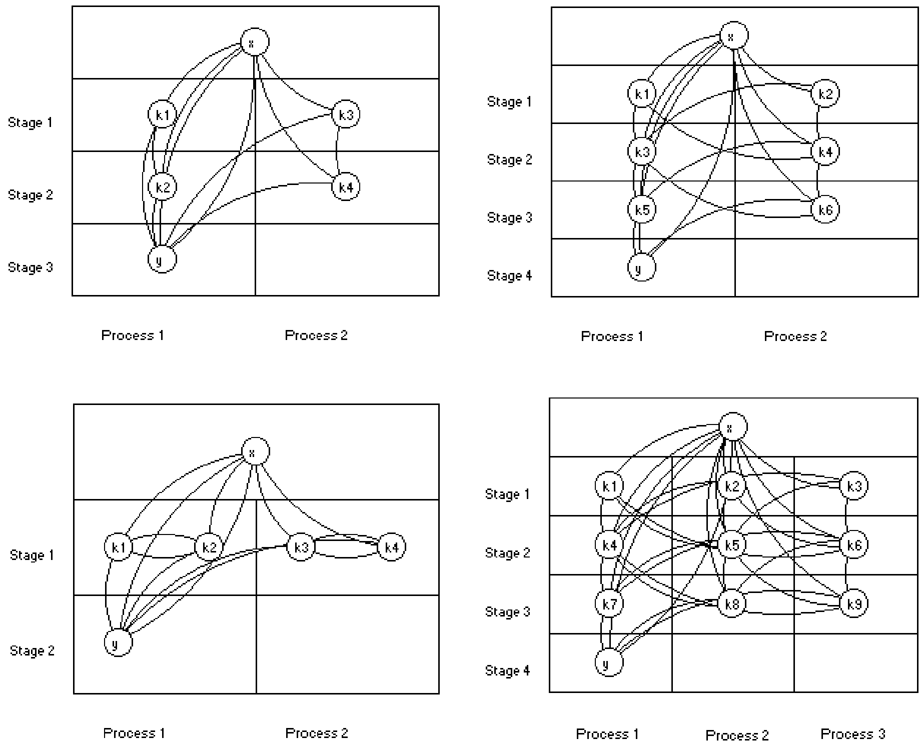

Fig. 1. Data-flow graphs reported by `EpODE` (x denote y_n and y denote y_{n+1}): from left to right and from top to bottom, the methods (5)-(8) – An arc from the left-part of a circle means a dependency from the top variable to the bottom variable, an arc from the right-part of a circle means a dependency from the bottom to the top variable when starting the next integration step, and an almost horizontal arc indicates a interdependence between the two linked variables. A k-labeled node refers the solving procedure for obtaining the value of the variable k using the previous (above) computed labeled node-values. More than one labeled-node in a computational cell indicates that a system formed with those variables must be solved. Two processes can run on the same or on different processors.

$$\frac{\partial u(x,t)}{\partial t} = u(x,t)\frac{\partial^2 u(x,t)}{\partial x^2} - x\cos(t)\frac{\partial u(x,t)}{\partial x} - x^2\sin(t) \qquad (9)$$

with $0 \le x \le 1,\ 0 \le t \le 1$. The initial and Dirichlet boundary conditions are such that the exact solution is given by $u(t,x) = x^2\cos(t)$: $u(0,t) = 0,\ u(1,t) = \cos(t),\ 0 < t \le 1,\ u(x,0) = x^2,\ 0 < x < 1$.

In order to solve both problems, we carry out a semi-discretization on the spatial variable by using second-order symmetric differences on a uniform grid with mesh size $\Delta x = 1/(m+1)$. This method (of lines) leads to IVPs with m ODEs.

As the third problem we take a real one. The PLEI [3] problem (28 ODEs) is a celestial mechanics problem of seven stars in the plane of coordinates x_i, y_i and masses $m_i = i$ $(i = 1 \ldots 7)$. It has the form $z" = f(z),\ z(0) = z_0,\ z'(0) = z'_0$ with $z = (x, y)^T \in R^{14},\ 0 \leq t \leq 3$ and

$$x"_i = \sum_{j \neq i} m_j (x_j - x_i)/r_{ij}, \qquad y"_i = \sum_{j \neq i} m_j (y_j - y_i)/r_{ij}, \tag{10}$$

where $m_i = i$ and $r_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{3/2}$.

Figure 2 obtained by using the visualize procedures from `Distributed Maple` shows the ratio between sequential and distributed time measurements corresponding to one arbitrary integration step. First vertical block of each figure corresponds to one sequential-integration, and the second one to the distributed-integration. Horizontal lines corresponding to a local or remote processor indicates the time when that processor is busy (continuous tasks). The time in seconds reported in the bottom-right corner of each figure represent the total time, the sequential one plus the distributed one.

The difference between the lengths of a local and a remote task is due to the fact that the local processor must compute explicitly the final value y_{n+1} from y_n and the known values k_i, must send the tasks to the other processors and the must prepare the nonlinear algebraic systems to be solved by the other processors. We see that in the case of the nonlinear problem the time reductions using a distributed environment are greater than the ones obtained in the case of the three-times greater linear problem.

Figure 3 also produced by `Distributed Maple` offers more details about the load-balancing between the running processes. Analyzing the top images we conclude that small linear IVPs, respectively problem (4) with $m = 10 \div 20$ equations, cannot be integrated in a distributed computational environment faster than using a sequential computer, since the distributed task are small relative to the overall time spent in one distributed integration step (including the necessary communications). In the case of nonlinear problems of similar dimensions, almost all computation time is spent on computing stage solutions.

The efficiency measurements of the distributed implementation of the selected method are shortly presented in Table 2. The horizontal lines split the inefficient values (top) from the efficient values (bottom), i.e. they indicate when the distributed solution computation time becomes smaller than the sequential solution computation time.

We can arrange the analyzed methods in a increasing trust order depending on the order in which they attain the horizontal lines: DIRK, PDIRK, BDIRK (we must prefer the BDIRK method). These methods appear in the reverse order if we sort them by the moment when they complete a time-step (we must prefer the DIRK method). Therefore supplementary parameters (like recommended step-size) must be take into account when we select a distributed methods.

We see also that the three-processor PSDIRK method can be almost so efficient as a two-processor method when we solve a nonlinear problem.

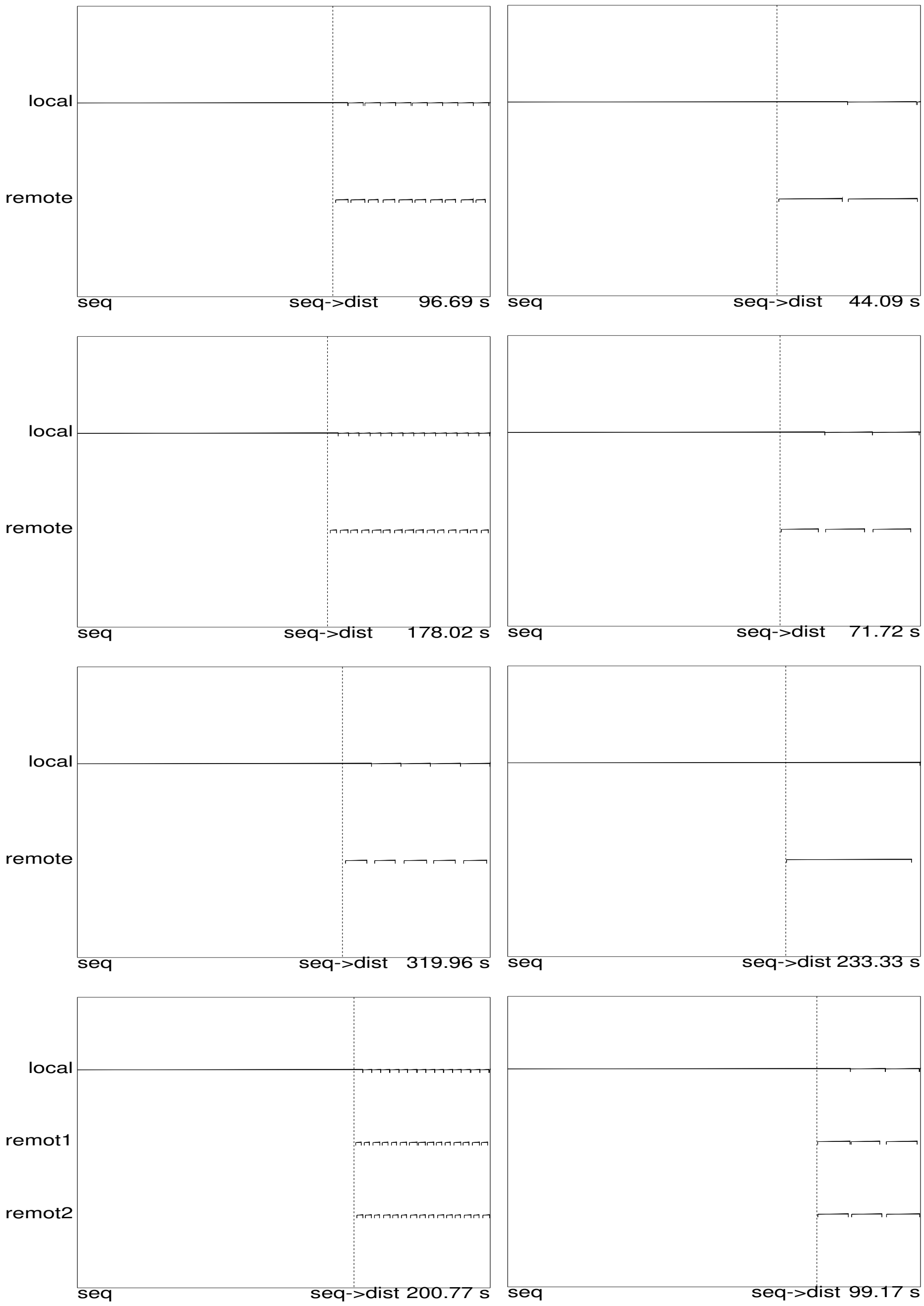

Fig. 2. Time measurements and processor load per k integration-steps: left column for linear problem (4) with $m = 60$ and $k = 5$, right column for nonlinear problem (9) with $m = 20$ and $k = 1$, first line for method (5), second line for (6), third line for (7), fourth line for (8)

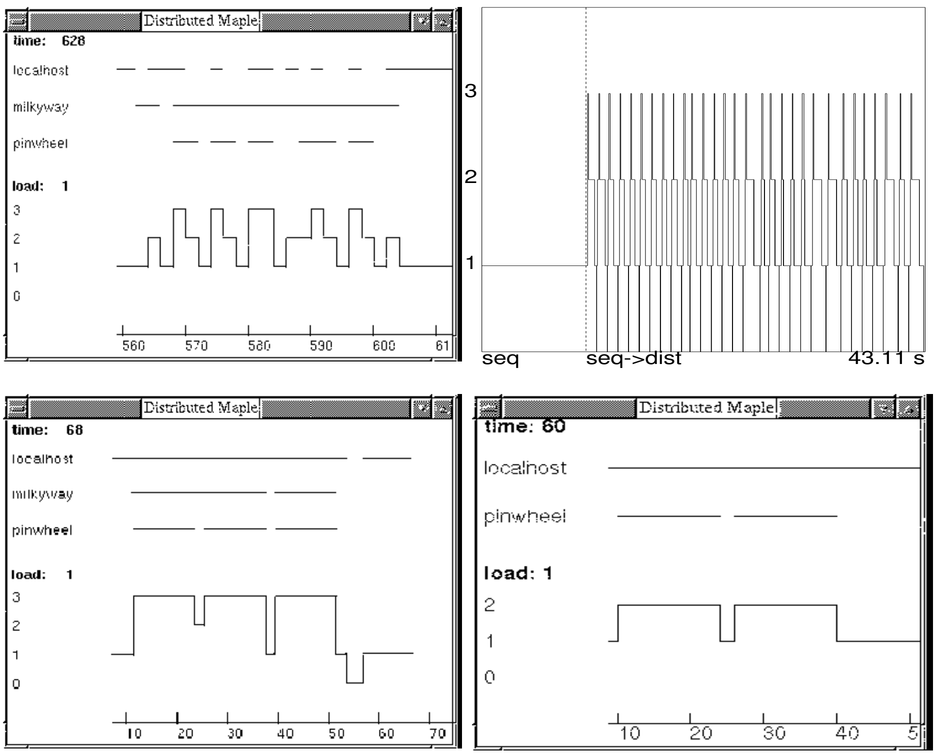

Fig. 3. Load balancing for k-integration steps: method (8) for linear test problem (4) with $m = 20$ and $k = 10$ (top-left), with $m = 10$ and $k = 10$ (top-right), and for nonlinear real problem (10) with $m = 28$ and $k = 1$ (bottom-left), respectively method (5) for nonlinear test problem (9) with $m = 25$ and $k = 1$ (bottom-right) – Short-time remote tasks cannot be combined efficiently, but large-time tasks can have the chance to overcome the communication time (the images are produced using the `Distributed Maple` facilities for the online visualization of load distribution)

We cannot expect to obtain similar efficiency results when we use explicit Runge-Kutta methods (or explicit multistep methods), since the solution of a stage equation involves only a small number of function evaluations and vector operations.

Due to the fact that the implicit equation solver implemented in `Maple` is more time-consumer than some modified Newton iterations written in programming language like `C`, we can obtain a lowest barrier in IVP sizes between efficient and inefficient implementation of distributed solvers. On other hand we can have more trust in the `Maple` solution of implicit equation system than in the numerical solution produced by a simple iterative solving procedure (this one must include a starting strategy and an error control mechanism which are not easy to be programmed). The overall solution computation time can be greater than that obtained by using ODE expert tools like `EpODE`, but the numerical solution

Table 2. Efficiency results: $E_p = T_s/(pT_p)$ where T_s is the mean time necessary to perform one-integration step using one processor, p is the number of processors, T_p is the mean time necessary to perform one-integration step using p processors

Linear problem (4)

Method	(5)	(6)	(7)	(8)
p	2	2	2	3
$m = 10$	9.84%	13.96%	34.48%	10.18%
$m = 20$	25.38%	29.05%	63.08%	22.67%
$m = 40$	53.85%	66.66%	86.57%	50.43%
$m = 60$	81.25%	76.72%	88.32%	67.70%

Nonlinear problem (9)

Method	(5)	(6)	(7)	(8)
p	2	2	2	3
$m = 5$	24.62%	28.26%	63.08%	24.31%
$m = 10$	49.97%	51.43%	92.00%	51.88%
$m = 15$	86.11%	87.38%	99.98%	81.96%
$m = 20$	94.12%	97.00%	99.99%	99.10%

Real problem (10)

Method	(5)	(6)	(7)	(8)
p	2	2	2	3
$m = 28$	97.00%	94.12%	95.54%	97.33%

can be more accurate. Using the accurate solution of implicit stage-equations produced by `Maple` we can apply the error control strategies for ODE solvers often reported in literature (usually these strategies do not consider the great influence of the implicit equation solver on the global error of the numerical ODE solution).

5 Conclusions

`D-NODE`, a `Maple` package written using `Distributed Maple` can extend the numerical solving capabilities of `Maple` from small initial value problems to ODE systems of order tens or order hundreds by exploiting the computational power of a local network of workstations. The solution computation time can be drastically reduced using a small number of worker processors. A strategy was adopted in which parts of some nonlinear systems to be solved at each time-step are send in algebraic forms to the workers. The solution accuracy compensates the supplementary time required by this non-classical procedure. Efficiency measurements indicate that the parallel implicit Runge-Kutta methods are fitted with this strategy.

Acknowledgments

This work was supported by Austrian Science Foundation project SFB P1303, and by the ERASMUS EU-grant SOCRATES 1999/ 2000. The author would like to express her appreciation to Wolfgang Schreiner, the creator of `Distributed Maple` and to thank him for the fruitful discussions.

References

1. Cong, N., A Parallel DIRK Method for Stiff Initial-Value Problems. *J. Comp. Appl. Math.* **54** (1994), 121-127.
2. Franco, J.M., Gomez, I., Two Three-Parallel and Three-Processor SDIRK Methods for Stiff Initial-Value Problems, *J. Comp. Appl. Math.* **87** (1997), 119-134.
3. Lioen, W.M., de Swart, J.J.B., van der Veen, W.A., Test Set for IVP Solvers, Report NM-R9615, CWI, August 1996, Available at http:// www.cwi.nl/ cwi/ projects/ IVPtestest.
4. Lioen, W.M., On the diagonal approximation of full matrices, *J. Comp. Appl. Math.* **75** (1996), 35-42.
5. Iserles, A., Nørsett, S.P., On the Theory of Parallel Runge-Kutta Methods. *IMA J. Numer. Anal.* **10** (1990), 463-488.
6. Jackson, K.R., Nørsett, S.P., The Potential for Parallelism in Runge-Kutta Methods. Part I: Runge-Kutta formulas in standard form. *SIAM J. Numer. Anal.* **32**, No. 1 (1995), 49-82.
7. Kahaner, D.K, Ng, E., Schiesser, W.E., Thompson, S., Experiments with an Ordinary Differential Equation Solver in the Parallel Solution of Method of Lines Problems on a Shared-Memory Parallel Computer, *Journal of Computational and Applied Mathematics* **38** (1991), 231–253.
8. Petcu, D., Implementations of Some Multiprocessor Algorithms for ODEs Using PVM. In *LNCS* **1332** (1997): Recent Advances in Parallel Virtual Machine and Message Passing Interface, eds. M. Bubak, J. Dongarra, J. Waśniewski, Springer Verlag, Berlin, 375-382.
9. Petcu, D., Parallelism in Solving Ordinary Differential Equations, Mathematical Monographs, The Press of Western University of Timişoara (1998), 7-126.
10. Petcu, D., Solving Initial Value Problems with a Multiprocessor Code. In *LNCS* **1662** (1999): Parallel Computing Technologies, ed. Victor Malyshkin, Springer Verlag, Berlin, 452-466.
11. Petcu, D., Drăgan, M., Designing an ODE Solving Environment. In *LNCSE* **10** (2000): Advances in Software Tools for Scientific Computing, eds. H.P. Langtangen, A. M. Bruaset, E. Quak, Springer-Verlag, Berlin, 319-338.
12. Petcu, D., The Potential for Distributing Computing in Large Initial Value Problems. In *Notes Num. Fluid Mech.* **73** (2000): Large Scale Scientific Computations of Engineering and Environmental Problems, eds. M. Griebel, S. Margenov, P. Yalamov, Springer Verlag, Berlin, 180-195.
13. Schreiner, W., Distributed Maple – User and Reference Manual. Technical Report 98-05 (1998), Research Institute for Symbolic Computation, Linz, and http://www.risc.uni-linz.ac.at/software/distmaple.
14. Shampine, L.F., Reichelt, M.W., The Matlab ODE Suite. *SIAM J. Sci. Comput.* **18**, No. 1 (1997), 1-22.
15. Sommeijer, B.P., Parallel Iterated Runge-Kutta Methods for Stiff Ordinary Differential Equations. *J. Comp. Appl. Math.* **45** (1993), 151-168.
16. Van der Houwen, P.J., Parallel Step-by-Step Methods. *Applied Numerical Mathematics* **11** (1983), 69-81.