Performance of a Parallel Technique for Solving Stiff ODEs Using a Block-diagonal Implicit Runge-Kutta Method

Kartawidjaja, M. A.; Suhartanto, H.; and Basaruddin, T.

Abstract—*Differential equations arise in many fields of application, such as in the simulation of phenomena in chemistry, physics, biology, medicine and so forth. These equations are generally in the form of initial value problems (IVPs), which can be extremely costly to solve when they are stiff due to the requirement of working with implicit methods. Implicit methods are costly because at each time step we need to solve implicit equations, which are nonlinear in general. Therefore, in such cases parallelization becomes an attractive approach.*

In this article we propose a parallel implementation of an ODE solver based on implicit Runge-Kutta framework. The parallelization is performed in two levels, i.e. across the method in solving the arising nonlinear systems and across the system in solving the associated linear systems. We use two kinds of test problems, the Brusselator and the Dense problems. The experiment was run on a cluster of PCs with PVM message-passing environment.

Our observations show that for the Brusselator problem, using parallelization will result in a better performance in terms of speedup for sufficiently large data. However for the Dense problem, the maximum attainable speedup is only two. We conclude that our two levels parallelization technique is only suitable for Brusselator type problems.

Index Terms—*message-passing, performance, PVM, stiff.*

1. INTRODUCTION

Many of mathematical models can be expressed in the form of IVPs for ODEs given in the following form [1]:

$$y'(x) = f(x, y(x)), \quad x \in [a, b], \qquad y(x_0) = y_0. \tag{1}$$

Manuscript received March 31, 2005. Revised July 18, 2005.

Kartawidjaja, M. A., Faculty of Electrical and Engineering, University of Atma Jaya, Jakarta, Indonesia (e-mail: maria.kw@atmajaya.ac.id).

Suhartanto, H., Faculty of Computer Science, University of Indonesia, Depok, Indonesia (email: heru@cs.ui.ac.id).

Basaruddin, T., Faculty of Computer Science, University of Indonesia, Depok, Indonesia (email: chan@cs.ui.ac.id).

where $y \to \Re^m$ and $f : \Re \times \Re^m \to \Re^m$. Higher order systems can always be written as systems of first order differential equations by introducing an additional vector and representing all derivatives accordingly. The solution of these ODEs can be extremely expensive due to the following factors:

- The dimension of the ODE is very large;
- The evaluation of the right-hand side factor is expensive;
- The interval of the integration is very long;
- The ODEs must be solved repeatedly.

It is widely believed that computationally intensive problems can be solved effectively using parallel computation.

There are three different techniques of parallelism. The first technique is parallelism across the method [2] that exploits concurrent function evaluations within a step or computes blocks of values simultaneously. The second technique is parallelism across the system [2] that involves decomposition of the problem into subproblems and solution of the subproblems concurrently. The last technique is parallelism across the step. This consists of strategy such as parallel solution of linear and nonlinear recurrences which are solved simultaneously over a large number of steps [3].

Several authors have designed and implemented ODE solvers. Their works are mainly based on construction of new integration formula which accommodates parallelism. For example, Suhartanto designed a parallel iterated technique based on multistep Runge-Kutta formula and implemented it using HPF90 technology [4]. Bendtsen designed a new formula based on Multiply Implicit Runge-Kutta (MIRK) methods which exploit parallelism across the method and implemented it using MPI (Message Passing Interface) technology [5]. De Swart designed a parallel iterated technique based on Implicit Runge-Kutta methods [6]. Bryne and Hindmarsh created MPI versions of PVODE from CVODE [7], a solver which is generated from two earlier solvers, VODE [8] and VODPK [9], by accommodating some parallel techniques [10]. The common part of the algorithms is the stepsize iteration (outer most loop) uses

nonlinear iteration (Newton loop) to solve the nonlinear equation, and the nonlinear iteration uses some linear solvers. It means that there will be a linear solver loop if an iteration technique is used. We refer to it as inner most loop. However, none of those works investigate the effect of parallelism inside the inner most loop to the next outer loop (Newton loop) and further to the outer most loop (stepsize loop). Our research is aim to investigate this effect of parallelization on the overall performance of the ODE solver.

In this article we propose a parallel implementation of an ODE solver based on implicit Runge-Kutta framework. The parallelization is performed in two levels, i.e. across the method in solving the arising nonlinear systems and across the system in solving the associated linear systems. We use two kinds of test problems, the Brusselator problem which has a banded matrix structure and the Dense problem which has a full matrix structure. Because in general numerical methods have either banded or full matrix structures, it is reasonable to take those two models as test problems. Furthermore, stiffness of the problem, a factor that influences the rate of convergence of the solution, can be adjusted, and so can be the dimension of the problem.

2. A BRIEF OVERVIEW OF NUMERICAL CONCEPTS

2.1. Runge-Kutta method

An autonomous form of an *s*-stage Runge-Kutta method [3] is expressed as

$$\begin{aligned} Y &= e \otimes y_n + h(A \otimes I_m)F(Y), \\ y_{n+1} &= y_n + h(b^T \otimes I_m)F(Y) \end{aligned} \qquad (2)$$

where Y represents the intermediate approximation vectors, *A* denotes the Runge-Kutta matrix and *h* is the step size. Using Newton iteration scheme results in the following linear system:

$$(I_s \otimes I_m - hA \otimes J)\Delta = G, \qquad (3)$$

where J is a Jacobian matrix. In order to reduce the computational cost, the Jacobian is evaluated at a single point [3], at y_0 for instance, and the method is called modified Newton method [3][1]. This Jacobian is computed either using analytical or numerical approach. If the users do not provide the analytical Jacobian, a numerical Jacobian is computed using finite difference method.

There are two diverse approaches to solve the linear systems, direct and iterative methods. Time complexity of direct methods equals $O(n^3)$ and of iterative methods $O(kxn^2)$, where *k* is the iteration step and *n* is the problem size. Thus, it is reasonable to consider the use of iterative methods to solve large linear systems. In our work we consider a widely used iterative method called GMRES (Generalized Minimal Residual) proposed by Saad [11] to solve the linear systems.

2.2. Error estimation

The estimation and control of errors are extremely important features in determining the accuracy of a numerical solution. A numerical method is supposed to produce a solution within a prescribed error tolerance. There are two measures of error commonly used, global error and local error. Although we are usually interested in global error which is the difference between the true solution and the numerical approximation, this error is not computable since we do not know the true solution in advance, besides it is difficult to estimate [12]. Therefore, most numerical codes estimate the local error and adjust the step size *h* in order to control the global error indirectly.

One technique to measure the local error is to use an embedding method introduced by Merson [13]. The idea of this method is to construct Runge-Kutta formulas which contain a second approximation $\hat{y}_n$ besides the numerical solution $\hat{y}$ and the order of those two formulas differs by one. The embedding process can be modeled by the following Butcher's tableau.

c	A
y	b^T
$\hat{y}$	$\hat{b}^T$
	e^T

The method defined by *c*, *A* and b^T is our numerical approximation that has order *q,* and that defined by *c*, *A* and $\hat{b}^T$ is the second approximation with one order higher or lower than the original method. Normally the embedded formula uses a lower order, i.e. $\hat{q} = q - 1$ [14]. The difference between the two approximations, $\hat{y}_n$ and y_n is used to estimate the local error

$$e_{n+1} = h\left\| \left(b^T - \hat{b}^T\right) \otimes I_m \right\| F(Y). \qquad (4)$$

There are many ways to measure the size of the estimated error vector. One way is to use a weighted root mean square norm [15] where the error is defined by

$$r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{e_{n,i}}{ewt_i}\right)^2}, \qquad (5)$$

in which $e_{n,i}$ is the i^{th} component of the error vector and *ewt* is an error weighting factor written as

$$ewt_i = atol_i + \max\left(\left|y_{0,i}\right|, \left|y_{1,i}\right|\right) rtol_i \qquad (6)$$

The parameter *rtol* and *atol* are respectively the relative and absolute tolerances specified by the user. The former indicates the number of digits in accordance with relative accuracy taken at each time step, and the latter designates the value of

the corresponding component of the solution vector. The estimated error at each time step varies with the stepsize with the aim of taking minimum number of steps while satisfying the error boundary.

2.3. Stepsize selection

An ODE integrator generates the solution by marching forward in time using one of the stepsize selection strategies, fixed stepsize or variable (adaptive) stepsize.

The simplest algorithms use fixed stepsize in the computation, meaning that a constant stepsize is used throughout the entire integration. However, this strategy can lead to excessive computation time especially when dealing with stiff problems where solutions have sharp changes. Therefore, modern solvers usually use algorithms that monitor the accuracy of the solution continuously and adjust the stepsize adaptively according to local truncation error calculated at each integration step. Stepsize should be chosen as such that it should be sufficiently small to cope with the sharp changes in the solution and must be as large as possible to minimize the computation cost and the round-off error.

A standard stepsize selection formula in integration of IVPs [15][16] is written as

$$h_{n+1} = \left(\frac{\varepsilon}{r_{n+1}}\right)^{1/k} h_n , \quad (7)$$

where r is the estimated error, k is a constant related to the order of the method and $\varepsilon = \theta . TOL$. The value of θ < 1 is a safety factor to reduce the risk of stepsize rejection. Although determining the stepsize using standard stepsize strategy is widely used and successfully implemented, for certain cases more sophisticated methods are required.

Based on the knowledge in linear digital control theory, Söderlind introduced a PID (Predictive Integral Derivative) controller to control the stepsize. The general form of the PID controller [17] is given as

$$h_{n+1} = \left(\frac{\varepsilon}{r_n}\right)^{(k_I + k_P + k_D)} \left(\frac{\varepsilon}{r_{n-1}}\right)^{-(k_P + 2k_D)} \left(\frac{\varepsilon}{r_{n-2}}\right)^{k_D} h_n , \quad (8)$$

where k_I, k_P and k_D are parameters of integral, proportional and derivative gains respectively. The family of filter based controllers can be classified using a notation of $Hp_Dp_Ap_F$, where p_D, p_A and p_F denote the dynamic order, adaptivity and filter order respectively. One class of this controller, *H211b*, is appropriate to solve medium to non-smooth problems as stated in [17]. This controller is defined by

$$h_{n+1} = \left(\frac{\varepsilon}{r_n}\right)^{1/(bk)} \left(\frac{\varepsilon}{r_{n-1}}\right)^{1/(bk)} \left(\frac{\varepsilon}{r_{n-2}}\right)^{-1/b} h_n . \quad (9)$$

In practice the value of b in equation (9) may be chosen in the range of 2 – 8, where larger values of b offer more smoothness of stepsize. Detail discussion of these controllers can be found in [17].

3. PARALLEL PERFORMANCE

There are various metrics to evaluate performance of parallel systems, such as run time, speedup and efficiency.

A sequential algorithm is usually evaluated in terms of its execution time, and expressed as a function of the problem size. On the contrary, the execution time of a parallel algorithm depends not only on the problem size, but also on the architecture of the parallel computer and the number of processors involved in the computation.

A general model of parallel execution time [18] is formulated as

$$T_p = T_{comp} + T_{comm}, \quad (10)$$

where T_{comp} and T_{comm} denote computation time and communication time respectively. This model is an ideal model where only computation and communication processes are assumed to take place. However, in parallel environment the execution time will also incorporate waiting time of a processor, synchronization time or even collision time if exists. Thus, a more realistic model of a parallel execution time is

$$T_p = T_{comp} + T_{comm} + \delta , \quad (11)$$

where δ is the time a processor neither in the state of computing nor in the state of communicating. This parameter denotes the overhead in parallel computation and thus limits the speedup. The overhead can be caused by several factors [19]:

- imbalanced workload;
- the time spent waiting at synchronization level;
- extra computations in the parallel version which are not needed in the sequential version.

Unfortunately those issues are conflicting with one another and thus must be traded off. Unlike the computation and communication time that contribute to the parallel execution time in a straightforward manner, the parameter δ is difficult to be determined.

4. PROBLEM STATEMENT

We use two types of test problems, 1-D diffusion Brusselator problem and the Dense problem. The dimension of ODEs used in these two problems ranges from 100 to 700.

4.1. Brusselator problem

The Brusselator problem is modeled as [20]

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$
$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2} \quad , \qquad (12)$$

where *u* and *v* denote the concentration of reaction products, *A* and *B* denote the concentration of input reagents. The parameter $\alpha = d/L^2$, where *d* is the diffusion coefficient and *L* is the reactor length. The value of α determines the stiffness of the problem. In our work we chose *A* = 1, *B* = 3 and α = 0.02. The initial conditions are

$$u(x,0) = 1 + \sin(2\pi x),$$
$$v(x,0) = 3. \qquad (13)$$

Setting $x_i = i/(N+1)$ $(1 \le i \le N)$, $\Delta x = 1/(N+1)$ and discretizing the derivatives in equation (12) over a grid of *N* points yields

$$u'_i = 1 + u_i^2 v_i - 4u_i + \frac{\alpha}{(\Delta x)^2}(u_{i-1} - 2u_i + u_{i+1})$$
$$v'_i = 3u_i - u_i^2 v_i - 4u_i + \frac{\alpha}{(\Delta x)^2}(v_{i-1} - 2v_i + v_{i+1}) \quad , \qquad (14)$$

with $u_i(0) = 1 + \sin(2\pi x_i), v_i(0) = 3$ and $i = 1, \ldots N$.

4.2. Dense problem

The Dense problem [4][21]] is defined by

$$y_i' = (QY)_i + e^{-\sum_{j=1}^{i} y_j^2}, \; i = 1, \ldots, m, \qquad (15)$$

where $Y \in \Re^m, Y(0) = e, \; x \in [0,1]$ and $Q = Q_1^T D Q_1$. Matrix Q has orthonormal columns of a random dense matrix. The eigenvalues of matrix D can be adjusted and hence the eigenvalue spectrum of matrix Q can be adjusted as well. In our work we use matrix D with an eigenvalue spectrum that ranges from 1 to 10.

5. IMPLEMENTATION ISSUES

In our work we use a block-diagonal Runge-Kutta matrix proposed by Iserles and Norsett [22] which is a method of order 4. For the second approximation we use a third order method. The construction of the third order method has to obey a certain condition as described in [15]:

$$\sum_j b_j = 1,$$
$$2\sum_{j,k} b_j a_{jk} = 1,$$
$$3\sum_{j,k,l} b_j a_{jk} a_{jl} = 1, \qquad (16)$$
$$6\sum_{j,k,l} b_j a_{jk} a_{kl} = 1.$$

Because it is impossible to find a pair of order 4(3) for a 4-stage Runge-Kutta method, we use a method called FSAL (First Same As Last) as suggested in [15]. This method adds *y* as a fifth stage of the process. As a result we have four linear systems with five unknowns, and thus we must take an arbitrary value for one of the unknown. In order to have a relative small difference between vector $\hat{b}^T$ and b^T we set $\hat{b}_5^T = 6 \times 10^{-16}$. The modified Runge-Kutta matrix is thus given in Table 3.

TABLE 3 MODIFIED ISERLES AND NØRSETT PAIR FORMULA.

$\frac{3-\sqrt{3}}{6}$	$\frac{5}{12}$	$\frac{1-2\sqrt{3}}{12}$	0	0
$\frac{3+\sqrt{3}}{6}$	$\frac{1+2\sqrt{3}}{12}$	$\frac{5}{12}$	0	0
$\frac{3-\sqrt{3}}{6}$	0	0	$\frac{1}{2}$	$\frac{-\sqrt{3}}{6}$
$\frac{3+\sqrt{3}}{6}$	0	0	$\frac{\sqrt{3}}{6}$	$\frac{1}{2}$
	$\frac{3}{2}$	$\frac{3}{2}$	-1	-1
	$\frac{9.857143}{6}$	$\frac{9.857143}{6}$	$-\frac{6.857143}{6}$	$-\frac{6.857143}{6}$ $6\text{x}10^{-16}$
	$\frac{0.857143}{6}$	$\frac{0.857143}{6}$	$\frac{0.857143}{6}$	$\frac{0.857143}{6}$ $6\text{x}10^{-16}$

To control the error we consider the stepsize controller introduced by Söderlind that belongs to *H211b* class and set *b* = 4 as recommended in [17].

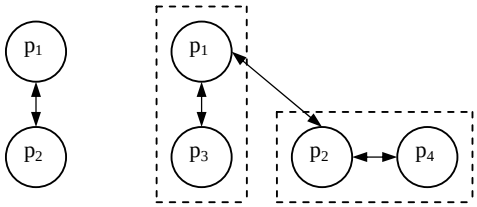
6. PARALLEL IMPLEMENTATION

In our experiment we exploit parallelism across the method as well as parallelism across the system. Parallelism across the method is implemented for solving the nonlinear systems using Newton's scheme as discussed in [23], and parallelism across the system is employed for solving the linear system using GMRES method as presented in [24]. If there are only two processors available, parallelization is performed only using across the method to solve the two nonlinear systems simultaneously. If more processors are available, the parallelization across the system is also done to solve the linear systems. Figure 5 gives an illustration of the parallelization process.

If there are only two processors, after initialization p_1 sends vector Y_2 and stepsize *h* to p_2. Afterwards each processor builds its own Jacobian, and the succeeding computation can be carried out concurrently. If there are four processors or more, the resulting linear systems can further be parallelized, meaning that the two groups of processors, $p_1 - p_3$ and $p_2 - p_4$ as depicted in Figure 6, work independently in solving the linear systems and the exchange of data between those two groups only occurs at the beginning and the end of the computation, or if synchronization is needed, such as if convergence is not achieved after a predefined maximum Newton iteration and time step *h* needs to be reduced as a consequence.

```
Preprocessing
t_n = t_0
while (t_n < t_fin)
initialize Y
  if p > 1
    p_1 sends Y_2 and h to p_2
  end if
  Newton_converge := false
  while (not converge)
   call nonlinear solver -> parallelism across
                            the method [23]
      GMRES_converge := false
      while (not converge)
         call linear solver ->  parallelism across
                                the system [24]
      end while
 end while
 estimate error
 compute h_new
 t_n = t_n + h_new
end while
```

FIGURE 5 STRUCTURE OF PARALLEL ODE.



(a) Two processors (b) Four processors

FIGURE 6 PROCESSOR WORKING-RELATIONSHIP.

To ensure load balancing, the number of processors involved in parallel environment should be an even number. Since only four processors are available in our working platform, we could use only two and four processors in parallel and each process is mapped onto each processor.

7. NUMERICAL EXPERIMENTS AND RESULTS

The experiment was performed on a cluster of PCs, consisting of four processors with similar characteristics, connected through a 100 Mbit Ethernet link. Each PC has an INTEL Pentium III 450 MHz processor and 512 MB RAM. They run Linux Mandrake 8 and the test code was written in C. The measurement was performed in PVM environment that provides a collection of library routines for message-passing. Further details concerning PVM can be found in [25].

7.1. Brusselator problem

The performance curve of the solver for the Brusselator problem with finite difference and analytical Jacobian is given in Figure 7.

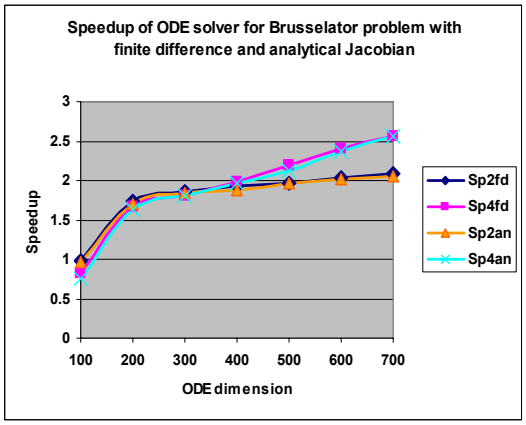


FIGURE 7 SPEEDUP OF ODE SOLVER FOR THE BRUSSELATOR PROBLEM WITH FINITE-DIFFERENCE AND ANALYTICAL JACOBIAN.

We note from Figure 7 that using multiprocessors will result in a better performance despite how the Jacobian matrix is provided except for ODE with small dimension, i.e. n = 100, where applying parallelization will result in a performance degradation because of the domination of communication time over computation time. We also observe that a superlinear speedup occurs for large size of ODE solved using two processors, i.e. ODE of dimension $n \geq 600$. This superlinear speedup occurs for ODE with either finite difference or analytical Jacobian. Superlinerar speedup does not reflect the actual performance gain, it is usually caused by the size of data which might be too large to fit into the main memory of a single processor.

Our observation indicates that solving the linear system requires approximately 50 to 70% of the overall execution time in each integration step. This large percentage of time will contribute to a better performance when more processors are available except for small data size, i.e. for ODE with dimension $n \leq 200$.

7.2. Dense problem

The performance of the Dense problem is illustrated in Figure 8. We note from this figure that a better performance can be achieved by using 2 and 4 processors.

For solver with finite difference Jacobian, we observe that approximately 95 to 99% of computation time is required to compute the Jacobian matrix and less than 5% to solve the linear systems. Because only two processors are

involved in solving nonlinear systems, the maximum attainable speedup is two, and employing more processors will not contribute to speedup because of the very small amount of time is needed to solve the linear system.

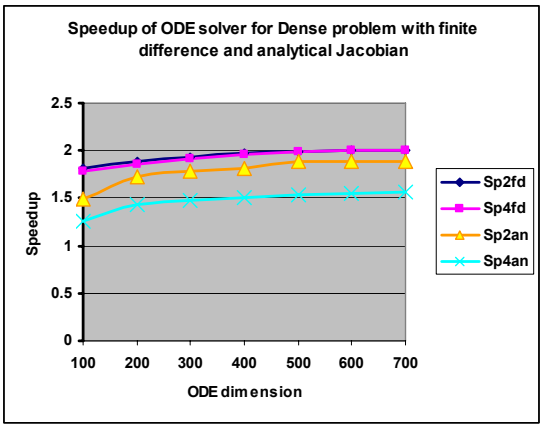


FIGURE 8 SPEEDUP OF ODE SOLVER FOR THE DENSE PROBLEM WITH FINITE-DIFFERENCE AND ANALYTICAL JACOBIAN.

For solver with analytical Jacobian, the computation time required to solve the linear system at each integration step is approximately 25 to 30% of the overall execution time. However, this amount of time is not large enough to contribute to the speedup by parallelizing the linear system, in contrary the performance will degrade. In other words, for solver with analytical Jacobian, the maximum performance is achieved by using two processors where only parallelization across the method is employed.

8. CONCLUSION

From the experiment we conclude that if solving the linear systems requires a large percentage of time as in the Brusselator problem, two levels parallelization will result in a better performance in terms of speedup. However, if solving the linear systems only consumes about 30% of the overall execution time as in the Dense problem, parallelizing the linear systems will not contribute to a better performance. Further experiment by incorporating more processors will be performed.

REFERENCES

[1] J. D. Lambert, "Numerical methods for ordinary differential equations: the initial value problems," John Wiley and Sons, New York, U.S.A., 1999.

[2] C. W. Gear, "The potential of parallelism in ordinary differential equations," Comp. Sci. Dept., Univ. of Illinois at Urbana Campaign, U.S.A., Tech. Rep. UIUC-DCS-R-86-1246, 1986.

[3] K. Burrage, "*Parallel and sequential methods for ordinary differential equations*," Oxford University Press, New York, U.S.A., 1995.

[4] H. Suhartanto, "Parallel iterated techniques based on multistep Runge-Kutta methods of Radau type," Ph.D. Thesis, Dept. of Mathematics, University of Queensland Brisbane, Australia, 1997.

[5] C. Bendtsen, "ParSODES: a parallel stiff ODE solver version 1.0, user's guide". Available: http://www.netlib.org/ode.

[6] J. J. B. de Swart, W. M. Lioen, and W. A. van der Veen, "Specification of PSIDE," CWI Amsterdam, Tech. Rep. MAS-R9833, 1998. Available: http://www.cwi.nl/cwi/projects/PSIDE.

[7] S. D. Cohen and A. C. Hindmarsh, "CVODE, a stiff/nonstiff ODE solver in C," *Computers in Physics*, 10 (2), pp. 138-143, 1996.

[8] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, "VODE: a variable coefficient ODE solver," *SIAM J. Sci. Statist. Comput. 10*, pp. 1038-1051, 1989.

[9] G. D. Byrne, "Pragmatic experiments with Krylov methods in the stiff ODE setting," In J.R. Cash and I. Gladwell, editors, *Computational Ordinary Differential Equations*, pp. 323-356, Oxford, 1992. Oxford University Press.

[10] G. D. Byrne and A. C. Hindmarsh, "PVODE, an ODE solver for parallel computers," *Int. J. High. Perf. Comput. Apps.* 13(4), pp. 354–365, 1999.

[11] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM, J. Sci. Stat. Comput.* 7 (3), pp. 856–869, 1986

[12] R. D. Skeel, "Thirteen ways to estimate global error," *Numerical Mathematics* 48, pp. 1-20, 1986.

[13] R. H. Merson, "An operational method for the study of integration processes," *In Proc. Symp. Data Processing, Weapons Research Establishment*, Salisbury, S. Australia, 1957.

[14] J. R. Dormand, J. P. Gillmore, and P. J. Prince,, "Globally Embedded Runge-Kutta Schemes," *Annals of Numerical Mathematics* 1, pp. 97-106, 1994.

[15] E. Hairer, S. P. Nørsett, and G. Wanner, "*Solving Ordinary Differential Equations I: Nonstiff Problems*," Springer-Verlag, Springer Series in Comp. Math., Berlin, 1993.

[16] C. W. Gear, "*Numerical initial value problems in ordinary differential equations*," Prentice Hall, Englewood Cliffs, NJ, 1971.

[17] G. Söderlind, "Digital Filters in Adaptive Time-Stepping," *ACM, TOM* 29 (1), pp. 1-26, 2003.

[18] I. Foster, "*Designing and building parallel programs: concepts and tools for parallel software engineering*," Addison Wesley Publishing Company, 1995.

[19] D. E. Culler, J. P. Singh, and A. Gupta, "*Parallel computer architecture: a hardware/software approach*," Morgan Kaufmann Publishers, Inc., San Francisco, CA, U.S.A., 1999.

[20] E. Hairer and G. Wanner, "*Solving Ordinary Differential Equations II: Stiff and Differential Algebraic Problems*," Springer-Verlag, Springer Series in Comp. Math., Berlin, 1996.

[21] K. Burrage, R. B. Sidje, and C. Eldershaw, "RadPk: a matrix free version of Radau5," unpublished, 1997.

[22] A. Iserles, and S. P. Nørsett, "On the theory of parallel Runge-Kutta methods," *IMAJNA* 10, 1990, 463-488.

[23] M. A. Kartawidjaja, H. Suhartanto, and T. Basaruddin, "Performance of a parallel technique for solving nonlinear systems arising from ODEs", *IEEE TENCON2004 Conference*, Chiang Mai, Thailand, 2004.

[24] M. A. Kartawidjaja, H. Suhartanto, and T. Basaruddin, "Performance of parallel iterative solution of linear systems using GMRES", *IPSI-2004 Kopaonik Conference*, Serbia, 2004.

[25] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "*PVM: Parallel Virtual Machine – A users' guide and tutorial for networked parallel computing*," MIT Press, Cambridge, Massachusetts, London, England, 1994.