Asian Journal of Information Technology 8 (3): 88-93, 2009
ISSN: 1682-3915


# Parallel Performance on Solving Lengyel-Epstein Model Using Runge-Kutta Method

H.A. Bastian and M.A. Kartawidjaja
Faculty of Engineering, Catholic University of Atma Jaya, Jakarta 12930, Indonesia

**Abstract:** The Lengyel-Epstein model is an Ordinary Differential Equation (ODE) model developed from Chlorite Iodide Malonic Acid (CIMA) chemical reaction in the Initial Value Problem (IVP) form. The stiffness of this model is determined by the rescaling parameter, which depends on the starch concentration. A stiff system is harder to solve and requires a larger computation time and thus, parallelization is applied. In this study, the ODE model is solved using Runge-Kutta implicit parallel scheme on a number of Personal Computers (PCs) connected in a network. The parallelization is implemented in two levels-one is parallelization across the method to solve the nonlinear equations and the other is parallelization across the system to solve the associated linear equations. The parallel environment is emulated using Parallel Virtual Machine (PVM) software. The performance of the parallel system is quantified by the speedup compared to a sequential system. The experiments showed that a significant speedup was achieved by implementing two level parallelization on the Lengyel-Epstein model.

**Key words:** Parallel, performance, PVM, Runge-Kutta, speedup, stiffness

## INTRODUCTION

Numerical modeling is commonly used in simulating natural phenomena, for example: weather predictions, air pollution, earth quakes, chemical reactions, ozone depletion, DNA structure analysis, et cetera. Modeling a system usually requires a large number of parameters and complex computational method, which in turn requires a large amount of time. One way to circumvent this problem is to use parallel computing where, the computation process is broken down into smaller processes to be solved simultaneously using a number of processors.

One model that needs intensive computational power is the Chlorite Iodide Malonic Acid (CIMA) chemical reaction introduced by Lengyel and Epstein. This model is an Initial Value Problem (IVP) involving a number of parameters related to the reactance of CIMA reaction.

Several researchers have designed and implemented ODE solver. Their research are mainly based on construction of new integration formula that accommodates parallelism. For example, Suhartanto (2007) designed a parallel iterated technique based on multistep Runge-Kutta formula. Bendtsen (1997) introduced a new formula based on Multiply Implicit Runge-Kutta (MIRK) methods that exploit parallelism across the method. De Swart *et al.* (1998) designed a parallel iterated technique based on implicit Runge-Kutta method. Cohen and Hindmarsh (1994, 1996) created PVODE from CVODE, which was a solver generated from two earlier solvers, VODE (Brown *et al.*, 1989) and VODPK (Byrne, 1992), by accommodating some parallel techniques. The common part of those algorithms is the step size iteration (outer most loop) uses nonlinear iteration (Newton loop) to solve the nonlinear system and the nonlinear iterations uses some linear solvers. It means that there will be a linear solver loop if an iteration technique is used. We name the loop as inner most loop. However, none of those research investigate the effect of parallelism inside the inner most loop to the next outer loop (Newton loop) and further to the outer most loop (step size loop). This study aims to investigate this effect of parallelization on the overall performance of the ODE solver.

We propose a parallel implementation of an ODE solver based on Runge-Kutta method. The parallelization was performed in two levels, across the method to solve the arising nonlinear system and across the system to solve the related linear system.

## MATERIALS AND METHODS

**Lengyel-epstein model:** The Lengyel-Epstein model can be declared in an Ordinary Differential Equation (ODE) system form as follows (Fengqi *et al.*, 2008):

$$\begin{cases} \dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2} + a - u - \dfrac{4uv}{1+u^2} \\ \dfrac{\partial v}{\partial t} = \sigma\left[c\dfrac{\partial^2 v}{\partial x^2} + b\left(u - \dfrac{uv}{1+u^2}\right)\right] \\ u(x,0) = u_0(x) > 0 \\ v(x,0) = v_0(x) > 0 \end{cases} \quad (1)$$

$$t>0,\ x\in\Omega$$

**Corresponding Author:** H.A. Bastian, Faculty of Engineering, Catholic University of Atma Jaya, Jakarta 12930, Indonesia

The parameter $\Omega \in R^n$ is the reactor, parameters u and v each declares the particle concentration of activator Iodide (I-) and inhibitor Chlorite (ClO2-). Parameters a and b are parameters concerning with feed concentrations. Parameter c is the ratio of diffusion coefficient where as $\sigma$ is the rescaling parameter determined by the starch concentration. The values for all parameters a, b, c and $\sigma$ must be positive. The stability of the Lengyel-Epstein model is determined by parameters a, b and initial condition of the system, where as the stiffness of the system is determined by the rescaling parameter $\sigma$.

**Runge-Kutta method:** Generally speaking there are two classes of numerical methods that can be used to solve an IVP problem, which are one step method and multistep method. A famous class of multistage one step method is the family of Runge-Kutta methods.

The Runge-Kutta method has two different forms, i.e., explicit and implicit. Implicit form is commonly used to solve a stiff problem. The general equation of an implicit autonomous Runge-Kutta method can be declared as (Burrage, 1995):

$$\begin{cases} y_{n+1} = y_n + h\left(b^T \otimes I_m\right)F(Y) \\ Y = e \otimes y_n + h\left(A \otimes I_m\right)F(Y) \end{cases} \quad (2)$$

Where,

$A$ and $b^T$ = Butcher table components
$Y$ = The intermediate approximation vectors
$h$ = The step size

Solving an ODE system for each integration step requires solving the non-linear system iteratively. The iterative method that is used here is the Newton method, which results in the following linear equation:

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

$$G = -Y + e \otimes y_n + h(A \otimes I_m)F(Y) \quad (4)$$

with parameter J declaring the Jacobian matrix. To reduce the computational cost, the Jacobian matrix is evaluated at a certain point, for example, at initial integration point, $y_0$. This is a modified Newton method as mentioned in Burrage (1995). To simplify the discussion, the operation $I_s \otimes I_m - hA \otimes J$ is symbolized with parameter M, so that (Eq. 4) can be declared as:

$$M \times \Delta = G \quad (5)$$

This linear equation is then solved with the iterative method known as GMRES (Generalized Minimal Residual) which was introduced by Saad (1986).

In advancing through the integration time, we used adaptive step size instead of fixed step size. The step size is dependent on the error from the previous step. Therefore error estimation is required in each time step. Because, global error is difficult to calculate or estimate (Skeel, 1986), the estimation was done for local error. Controlling the size of the local error will indirectly affect the size of the global error.

One way to calculate local error is Merson's (1957) embedding method. For that purpose, we need to build a Runge-Kutta formula that differed in one order from the Runge-Kutta formula that was used. Usually the embedded formula uses a lower order (Dormand *et al*., 1994). If the approximation results with the first and second Runge-Kutta formula is $y_n$ and $\hat{y}_n$, then local error can be declared as:

$$e_{n+1} = h\left\|\left(b^T - \hat{b}^T\right) \otimes I_m\right\|F(Y) \quad (6)$$

The error can be calculated by using weighted root mean square norm (Hairer *et al*., 1993):

$$r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{e_{n,i}}{ewt_i}\right)^2} \quad (7)$$

with $e_{n,i}$ declaring the ith error vector component and ewt declaring the error weighting factor, which size is declared in the following equation:

$$ewt_i = atol_i + \left|y_{n,i}\right| \times rtol_i \quad (8)$$

Parameter atol is the absolute tolerance value and parameter rtol is the relative tolerance value.

**Parallel system speedup:** A parallel system can be viewed as a collection of processing elements that communicate and cooperate to solve large problems faster than using a single processing element. In essence, a parallel system is expected to provide a better performance than that provided by a sequential system. As in single processor, most performance issues in multiprocessor can be addressed by programming or architectural techniques or both. The focus of this study is the performance issues addressed by programming techniques. There are various metrics to evaluate performance of a parallel system, such as execution time, speedup, efficiency and cost. Nevertheless, the most general metric is speedup (Wu, 1999).

Speedup denotes the performance gain of a parallel system in terms of reduced execution time compared to the execution time of the sequential system. By defining

$T_1$ and $T_p$ as execution time of a parallel algorithm on one and on p processors respectively, speedup is formulated as:

$$Sp = \frac{T_1}{T_p} \tag{9}$$

This speedup is termed as relative speedup. If the execution time on one processor is substituted by execution time of the best sequential algorithm, then an absolute speedup is identified.

The lower bound of the speedup is equal to one and the upper bound is equal to the number of processors in the system. A speedup that equals to this upper bound is termed as ideal speedup. In practice, sometimes the performance of a parallel system degrades by using more than one processor and thus speedup becomes <1. This fact is usually caused by excessive communication or by the small data size. In some cases, a speedup greater than the number of processors is observed. This super-linear speedup is commonly caused by the size of the data that might be too large to fit in the main memory of a single processor.

There are two distinct programming models in parallel computing platform, shred memory and distributed memory models. Two popular message passing softwares are commonly used to emulate parallel platform, i.e., Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The research was performed using PVM software (Geist *et al*., 1994).

**Parallel Virtual Machine (PVM):** PVM is software that makes a collection of computers appear as one large virtual machine. The set of computers used in the parallel environment must be defined prior to running the programs. PVM can be used on homogeneous or heterogeneous platforms. It handles transparently all message routing, data conversion and task scheduling across a network of computers.

PVM system is composed of two parts. The first part is a daemon that resides in all computers forming the virtual machine. The second part is a library of PVM interface routines that contains a complete collection of primitives that are required for cooperating processes.

PVM allows creation of any number of processes, independent on the number of processors used in the system. Each process is identified by a task ID. The processes will be mapped on to processors automatically unless overridden by the programmer.

PVM programs are generally organized in a master-slave arrangement, where a single process, referred as master process, is first executed and all other processes, child processes, are created by master process when necessary.

**Parallelization strategy:** The discretization process in the Lengyel-Epstein model resulted in the following equation:

$$\begin{aligned}
u_i' &= \frac{1}{(\Delta x)^2}\left[u_{i-1} - 2u_i + u_{i+1}\right] + a - u_i - \frac{4u_i v_i}{1+u_i^2} \\
v_i' &= \frac{\sigma c}{(\Delta x)^2}\left[v_{i-1} - 2v_i + v_{i+1}\right] + \sigma b u_i - \frac{\sigma b u_i v_i}{1+u_i^2} \\
\Delta x &= 1/(N+1) \\
x_i &= i\,\Delta x, \quad (1 \le i \le N)
\end{aligned} \tag{10}$$

The parameter value used here are a = 20, b = 1.2 and c = 2, following the recommendation (Fengqi *et al*., 2008) for a stable system. The stiff condition was acquired by taking the rescaling parameter $\sigma = 8\times10^3$. The initial condition uses two function types. The constant functions are declared as $u_0(x) = 2.0$ and $v_0(x) = 11.0$. The sinusoidal functions are declared as $u_0(x) = 3 + \sin(x)$ and $v_0(x) = 10 + \cos(x)$. The Runge-Kutta matrix used in the investigation is a fourth order Runge-Kutta matrix introduced by Iserles and NØrsett (1990). This matrix has decoupled sub-matrix structures to as shown in Fig. 1.

The error estimation here uses embedding technique by forming second approximation, which has a lower order from the Runge-Kutta formula used to solve the ODE. The modified matrix can be shown in Fig. 2 (Kartawidjaja, 2004).

The absolute tolerance and relative tolerance in Eq. 8 is set to $10^{-5}$. Integration step size in this ODE system is calculated from Predictive Integral Derivative (PID) formula controller from H211b class introduced by Soderlind (2003) with the following equation:

$$h_{n+1} = \left(\frac{\varepsilon}{r_n}\right)^{1/(bk)} \left(\frac{\varepsilon}{r_{n-1}}\right)^{1/(bk)} \left(\frac{h_n}{h_{n-1}}\right)^{-1/b} h_n \tag{11}$$

Parameter h represents the integration step size, parameter r denotes error prediction and parameter ε is the multiplication of safety factor with the allowed tolerance. Tolerance value is between 0<TOL<1. Parameter k declares the order from the ODE system. To acquire a smooth integration step, parameter b = 4 is used following the recommendation by Soderlind (2003).

In order to balance the workload among processors, the number of processors used in the parallel environment must be even, which is two, four, six and eight processors. For clarity of discussion, we provide a working relationship of 2, 4 and 6 processors in Fig. 3, where $p_1$ denotes root processor, whilst $p_2$, $p_3$, $p_4$, $p_5$ and denote child processors. The eight processors interrelationship can be constructed analogously. Parallel environment is emulated with PVM software.

| | | | | |
|---|---|---|---|---|
| $\frac{3-\sqrt{3}}{6}$ | $\frac{5}{12}$ | $\frac{1-2\sqrt{3}}{12}$ | 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 |

Fig. 1: Runge-Kutta table introduced by Iserles and Nørsett (1990)

| | | | | | |
|---|---|---|---|---|---|
| $\frac{3-\sqrt{3}}{6}$ | $\frac{5}{12}$ | $\frac{1-2\sqrt{3}}{12}$ | 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 | |
| | $\frac{9.857143}{6}$ | $\frac{9.857143}{6}$ | $-\frac{6.857143}{6}$ | $-\frac{6.857143}{6}$ | $6\cdot10^{-16}$ |
| | $\frac{0.857143}{6}$ | $\frac{0.857143}{6}$ | $-\frac{0.857143}{6}$ | $-\frac{0.857143}{6}$ | $6\cdot10^{-16}$ |

Fig. 2: Modified Runge-Kutta table for embedding process

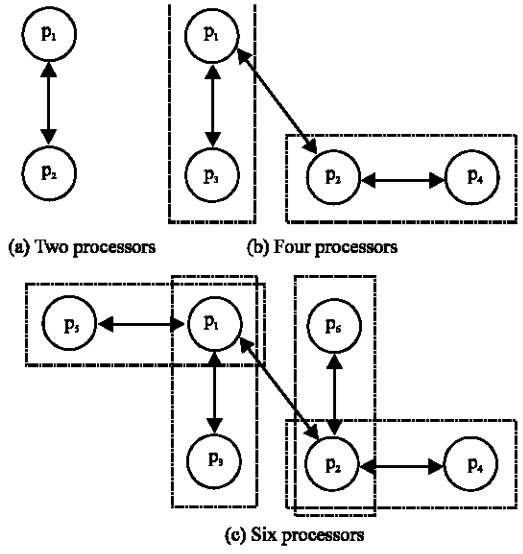


Fig. 3: Working relationships of the processors

The parallelization in this study is done in two steps, across the method to solve the nonlinear system using Newton method (Kartawidjaja *et al*., 2004a) and across the system to solve linear system using GMRES method (Kartawidjaja *et al*., 2004b). If only two processors are available, then parallelization is only done to solve nonlinear system. But if more than two processors are available, parallelization is also implemented to solve the linear system.

Generally, there are three basic algebra operations in GMRES method, which are Inner Product (IP) operation, Vector Update (VU) operation and Matrix Vector product (MV) operation.

IP and VU are BLAS 1 operations, where as MV is BLAS 2 operation, which means MV operation requires a much larger computation time than the computation time needed by IP or VU operations. By that reason, parallelization in the linear system is only implemented in MV operation.

## RESULTS AND DISCUSSION

The experiment was performed on a cluster of PCs, consisting of eight processors with similar characteristics, connected through a 100 Mbit Each PC has a CPU of 1.5 GHz and 512 MB RAM and the test code was written in C.

The performance of parallel computing is evaluated using speedup metric. Testing has been done for two initial conditions, initial condition in constant form and initial condition in sinusoid function. Test result of execution time for initial condition in constant form is presented in Table 1.

It is noted from Table 1 that the execution time increases with more processors for ODE dimension of 300. But for ODE dimension above 300, the execution time decreases as the number of processor increases. The same phenomenon also appears for initial condition in sinusoidal form as shown in Table 2.

By comparing Table 1 and 2, we see that ODE with initial condition in sinusoid form requires larger computation time compared to ODE with initial condition in constant form.

We can calculate the speedup from the execution time in Table 1 and 2 using Eq. 9 and the results are provided in Table 3 and 4.

It was shown that in general speedup occurs for ODE with dimension greater or equal to 400, both for initial condition in constant form (Table 3) and for initial condition in sinusoid form (Table 4). In addition, the speedup for initial condition in constant form is larger compared to the speedup for initial condition in sinusoid form, as illustrated in Fig. 4 and 5 for two and eight processors, respectively.

Our observation on single processor revealed that solving the linear system inside the Newton loop required

Table 1: Execution time of ODE with initial condition in constant form

| ODE dimension/n | Execution time (sec) $T_1$ | $T_2$ | $T_4$ | $T_6$ | $T_8$ |
|---|---|---|---|---|---|
| 300 | 241 | 201 | 344 | 358 | 375 |
| 400 | 654 | 492 | 492 | 409 | 391 |
| 500 | 1687 | 1163 | 922 | 743 | 570 |
| 600 | 3338 | 2086 | 1314 | 1048 | 835 |
| 700 | 6182 | 3237 | 1938 | 1614 | 1247 |
| 800 | 7732 | 3987 | 2358 | 1943 | 1501 |

Table 2: Execution time of ODE with initial condition in sinusoid form

| ODE dimension/n | Execution time (sec) $T_1$ | $T_2$ | $T_4$ | $T_6$ | $T_8$ |
|---|---|---|---|---|---|
| 300 | 277 | 283 | 413 | 477 | 512 |
| 400 | 726 | 622 | 572 | 537 | 569 |
| 500 | 1831 | 1387 | 1071 | 885 | 799 |
| 600 | 3592 | 2500 | 1561 | 1294 | 1148 |
| 700 | 6493 | 4008 | 2143 | 1809 | 1627 |
| 800 | 8159 | 4859 | 2593 | 2189 | 1963 |

Table 3: Speedup of ODE with initial condition in constant form

| ODE dimension/n | Speedup $Sp_2$ | $Sp_4$ | $Sp_6$ | $Sp_8$ |
|---|---|---|---|---|
| 300 | 1.20 | 0.70 | 0.67 | 0.64 |
| 400 | 1.33 | 1.33 | 1.60 | 1.67 |
| 500 | 1.45 | 1.83 | 2.27 | 2.96 |
| 600 | 1.60 | 2.54 | 3.19 | 4.00 |
| 700 | 1.91 | 3.19 | 3.83 | 4.96 |
| 800 | 1.94 | 3.28 | 3.98 | 5.15 |

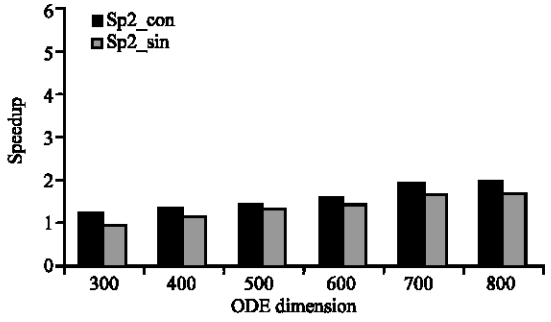


Fig. 4: Speedup of ODE with initial condition in constant and sinusoid form on two processors

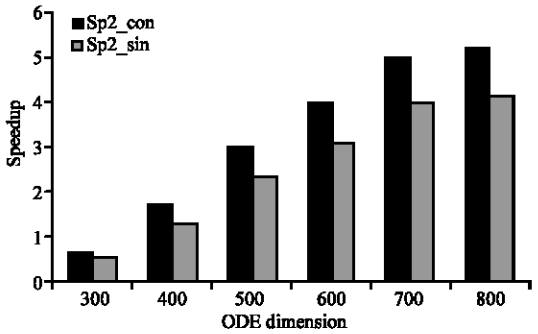


Fig. 5: Speedup of ODE with initial condition in constant and sinusoid form on eight processors

Table 4: Speedup of ODE with initial condition in sinusoid form

| ODE dimension/nn | Speedup $Sp_2$ | $Sp_4$ | $Sp_6$ | $Sp_8$ |
|---|---|---|---|---|
| 300 | 0.98 | 0.67 | 0.58 | 0.54 |
| 400 | 1.17 | 1.27 | 1.35 | 1.28 |
| 500 | 1.32 | 1.71 | 2.07 | 2.29 |
| 600 | 1.44 | 2.30 | 2.78 | 3.13 |
| 700 | 1.62 | 3.03 | 3.59 | 3.99 |
| 800 | 1.68 | 3.15 | 3.73 | 4.16 |

approximately 60-70% 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., 300.

## CONCLUSION

From the test results, we can draw the following conclusions:

- For ODE with dimension greater or equal 400 we can acquire speedup by implementing parallelization in two levels
- Speedup for ODE with initial condition in constant form is larger compared to speedup for ODE with initial condition in sinusoid form

## REFERENCES

Bendtsen, C., 1997. ParSodes: A Parallel Stiff ODE Solver Version 1.0 User's Guide. http://www.netlib.org/ode/parsodes.tar.gz.

Brown, P.N., G.D. Byrne and A.C. Hindmarsh, 1989. VODE, a variable-coefficient ODE solver. SIAM J. Scient. Statist. Comput., 10 (5): 1038-1051. DOI: 10.1137/0910062.

Burrage, K., 1995. Parallel and Sequential Methods for Ordinary Differential Equations. Numerical Mathematics and Scientific Computation Series. Oxford University Press, New York. ISBN: 0-19-853432-9.

Byrne, G.D., 1992. Pragmatic Experiments with Krylov Methods in the Stiff ODE Setting. In: Cash, J.R. and I. Gladwell (Eds.). Computational Ordinary Differential Equations. Oxford University Press, Oxford, pp: 323-356.

Cohen, S.D. and A.C. Hindmarsh, 1994. CVODE User Guide. Numerical Mathematics Group, UCRL-MA-118618. https://computation.llnl.gov/casc/nsde/pubs/u118618.pdf.

Cohen, S.D. and A.C. Hindmarsh, 1996. CVODE, A Stiff/Nonstiff ODE Solver in C. Computers in Physics, 10 (2): 138-143. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.5594.

De Swart, J.J.B., W.M. Lioen and W.A. Van Der Veen, 1998. Specification of PSIDE. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.1762.

Dormand, J.R. *et al*., 1994. Globally embedded Runge-Kutta schemes. Ann. Numer. Mathe., 1: 97-106.

Fengqi, Y. *et al*., 2008. Diffusion-driven instability and bifurcation in the lengyel-epstein system. Nonlinear Anal. Real World Appl., 9 (3): 1038-1051. DOI: 10.1016/j.nonrwa.2007.02.005.

Geist, A. *et al*., 1994. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. Scientific and Engineering Computation Series. Massachusetts: MIT Press, Cambridge. ISBN: 0-262-57108-0. http://www.netlib.org/pvm3/book/pvm-book.html.

Hairer, E., S.P. NØrsett and G. Wanner, 1993. Solving Ordinary Differential Equations I: Nonstiff Problems. 2nd Edn. Springer Series in Computational Mathematics. Springer-Verlag, Berlin Heidelberg. ISBN: 978-3-540-56670-0. DOI: 10.1007/978-3-540-78862-1.

Iserles, A. and S.P. NØrsett, 1990. On the theory of parallel Runge-Kutta methods. IMA J. Numer. Anal., 10 (4): 463-488. DOI: 10.1093/imanum/10.4.463.

Kartawidjaja, M.A., 2004. Performance of Parallel ODE Solver Using Semi-Implicit Runge-Kutta Method. http://lontar.cs.ui.ac.id/Lontar/opac/themes/ng/hasilcari.jsp?method=similar&query=8609&lokasi=lokal.

Kartawidjaja, M.A., H. Suhartanto and T. Basaruddin, 2004a. Performance of a parallel technique for solving nonlinear systems arising from ODEs. IEEE. TenCon, Chiang Mai, Thailand, 4: 403-406. ISBN: 0-7803-8560-8. DOI: 10.1109/TENCON.2004.1414955.

Kartawidjaja, M.A., H. Suhartanto and T. Basaruddin, 2004b. Performance of parallel iterative solution of linear systems using GMRES. Proceedings of the International IPSI-2004k Conference, Kopaonik, Published on CD, Serbia, ISBN: 86-7466-117-3. http://kopaonik.internetconferences.net/BookOfAbstractsBody.doc.

Merson, R.H., 1957. An operational method for the study of integration processes. In Proceedings of a Symposium on Data Processing. Weapons Research Establishment, Salisbury, South Australia.

Saad, Y. and M.H. Schultz, 1986. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM J. Scient. Statist. Comput., 7 (3): 856-869.

Skeel, R.D., 1986. Thirteen ways to estimate global error. Numerische Mathematik, 48 (1): 1-20. DOI: 10.1007/BF01389440.

Soderlind, G., 2003. Digital filters in adaptive time-stepping. ACM Trans. Mathe. Software, 29 (1): 1-26. http://doi.acm.org/10.1145/641876.641877.

Suhartanto, H., 2007. VSMRK: A parallel implementation and the performance of variable stepsize multistep Runge-Kutta methods for stiff ODEs. Asian J. Inform. Technol., 6 (11): 1110-1116.

Wu, X., 1999. Performance Evaluation, Prediction and Visualization of Parallel Systems. Kluwer Academic Publishers, The Netherlands. ISBN: 0-7923-8462-8.