3

# Parallel direct integration variable step block method for solving large system of higher order ordinary differential equations

Zanariah Abdul Majid and Mohamed Suleiman
*Universiti Putra Malaysia*
*Malaysia*

## 1. Introduction

The performance of the developed two point block method designed for two processors for solving directly non stiff large systems of higher order ordinary differential equations (ODEs) has been investigated. The method calculates the numerical solution at two points simultaneously and produces two new equally spaced solution values within a block and it is possible to assign the computational tasks at each time step to a single processor. The algorithm of the method was developed in C language and the parallel computation was done on a parallel shared memory environment. Numerical results are given to compare the efficiency of the developed method to the sequential timing. For large problems, the parallel implementation produced 1.95 speed up and 98% efficiency for the two processors.

## 2. Background

The ever-increasing advancement in computer technology has enabled many in science and engineering sectors to apply numerical methods to solve mathematical model that involve ODEs. The numerical solution of large ODEs systems requires a large amount of computing power. Users of parallel computing tend to be those with large mathematical problems to solve with the desire to obtain faster and more accurate results.
In this paper, we consider solving directly the higher order IVPs for system of ODEs of the form

$$y'' = f(x, y, y'),\ y(a) = y_0\ ,\ y'(a) = y'_0,\ x \in [a, b]\,. \tag{1}$$

Equation (1) can be reduced to the equivalent first order system of twice the dimension equations and then solved using any numerical method. This approach is very well established but it obviously will enlarge the dimension of the equations. The approach for solving the system of higher order ODEs directly has been suggested by several researchers such as in (Suleiman, 1989); (Fatunla, 1990); (Omar, 1999); (Cong et al., 1999) and (Majid & Suleiman, 2006). In previous work of (Omar, 1999), a general $r$ block implicit method of multistep method for solving problems of the form (1) has been investigated. The code used

a repetitive computation of the divided differences and integration coefficients that can be very costly. The worked in (Majid & Suleiman, 2007) has presented a direct block method (2PFDIR) for solving higher order ODEs in variable step size which is faster in terms of timing and comparable or better in terms of accuracy to the existence direct non block method in (Omar, 1999). The 2PFDIR method will store all the coefficients in the code and there will be no calculations that involved the divided difference and integration coefficients. In this paper, we would like to extend the discussions in (Majid & Suleiman, 2007) on the performance of 2PFDIR method using parallel environment particularly focus on the cost of time computation by comparing the execution time of sequential and parallel implementation for solving large problem.

## 3. Formulation of the method

In Figure 1, the two values of $y_{n+1}$ and $y_{n+2}$ are simultaneously computed in a block using the same back values. The computed block has the step size $h$ and the previous back block has the step size $rh$. The idea of having the ratio *r* is for variable step size implementation.

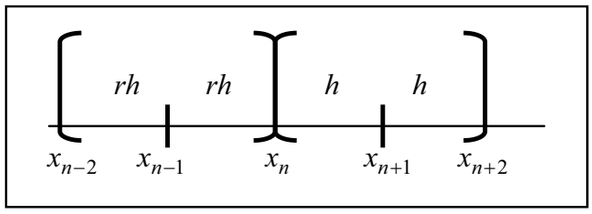


Fig. 1. Two Point Block Method.

In Eqn. (1), the $f(x,y,y')$ will be replaced with Lagrange interpolation polynomial and the interpolation points involved were $(x_{n-2},f_{n-2}),\ldots,(x_{n+2},f_{n+2})$. These polynomial will be integrate once and twice over the interval $[x_n,x_{n+1}]$ and $[x_n,x_{n+2}]$, and the following corrector formulae were obtained,

*Integrate once*
First point:

$$y'(x_{n+1}) = y'(x_n) + \frac{h}{240(r+1)(r+2)(2r+1)r^2}\Big[-(2r+1)r^2(3+15r+20r^2)f_{n+2} + 4r^2(r+2)(18+75r+80r^2)f_{n+1} + (r+1)(r+2)(2r+1)(7+45r+100r^2)f_n - 4(2r+1)(7+30r)f_{n-1} + (r+2)(7+15r)f_{n-2}\Big] \quad . \tag{2}$$

Second point:

$$y'(x_{n+2}) = y'(x_n) + \frac{h}{15r^2(2r+1)(r+2)(r+1)}\Big[r^2(2r+1)(5r^2+15r+9)f_{n+2} + 4r^2(r+2)(10r^2+15r+6)f_{n+1} + (r+2)(r+1)(2r+1)(5r^2-1)f_n + 4(2r+1)f_{n-1} - (r+2)f_{n-2}\Big]. \tag{3}$$

*Integrate twice*
First point:

$$y(x_{n+1}) - y(x_n) - hy'(x_n) = \frac{h^2}{240r^2(r+1)(r+2)(2r+1)}\Big[-r^2(2r+1)(1+6r+10r^2)f_{n+2} + 4r^2(r+2)(4+21r+30r^2)f_{n+1} + (r+1)(r+2)(2r+1)(3+24r+70r^2)f_n - 4(2r+1)(3+16r)f_{n-1} + (r+2)(3+8r)f_{n-2}\Big] \quad . \tag{4}$$

Second point:

$$y(x_{n+2}) - y(x_n) - 2hy'(x_n) = \frac{h^2}{15r(r+1)(r+2)(2r+1)}\Big[r(2+3r)(2r+1)f_{n+2} + 8r(2+6r+5r^2)(r+2)f_{n+1} + (3+10r)(r+1)(r+2)(2r+1)f_n - 8(2r+1)f_{n-1} + (r+2)f_{n-2}\Big] \quad . \tag{5}$$

During the implementation of the method, the choices of the next step size will be restricted to half, double or the same as the previous step size and the successful step size will remain constant for at least two blocks before considered it to be doubled. This step size strategy helps to minimize the choices of the ratio *r*. In the code developed, when the next successful step size is doubled, the ratio *r* is 0.5 and if the next successful step size remain constant, *r* is 1.0. In case of step size failure, *r* is 2.0. Substituting the ratios of *r* in (2) – (5) will give the corrector formulae for the two point block direct integration method. For detail see (Majid & Suleiman, 2007).
For example, taking $r = 1$ in Eqn. (2) – (5) will produce the following corrector formulae:
Integrate once:

$$y'_{n+1} = y'_n - \frac{h}{720}(19f_{n+2} - 346f_{n+1} - 456f_n + 74f_{n-1} - 11f_{n-2}) \tag{6}$$

$$y'_{n+2} = y'_n + \frac{h}{90}(29f_{n+2} + 124f_{n+1} + 24f_n + 4f_{n-1} - f_{n-2}) \tag{7}$$

Integrate twice:

$$y_{n+1} = y_n + hy'_n - \frac{h^2}{1440}(17f_{n+2} - 220f_{n+1} - 582f_n + 76f_{n-1} - 11f_{n-2}) \tag{8}$$

$$y_{n+2} = y_n + 2hy'_n + \frac{h^2}{90}(5f_{n+2} + 104f_{n+1} + 78f_n - 8f_{n-1} + f_{n-2}) \tag{9}$$

This block method was applied using predictor and corrector mode. The developed method is the combination of predictor of order 4 and the corrector of order 5. The predictor formulae was derived similar as the corrector formulae and the interpolation points involved are $x_{n-3},\ldots,x_n$ .

## 4. Parallelism Implementation

The code starts by finding the initial points in the starting block for the method. Initially we use the sequential direct Euler method to find the three starting points for the first block using constant *h*. The Euler method will be used only once at the beginning of the code. Once we find the points for the first starting blocks, then we could apply the block method until the end of the interval.

Within a block in the parallel two point direct block method for two processors (P2PFDIR), it is possible to assign both the predictor and the corrector computations to a single processor and to perform the computations simultaneously in parallel. Each application of the block method generates a collection of approximation to the solution within the block.

In Eqn. (2) – (5), each computed block consists of two steps, i.e n+1 and n+2. The predictor equation are dependent on values taken from the previous block of n, n-1, n-2, n-3 and the corrector values depend on the current blocks at the points n+1 and n+2. The predictor and corrector equations at each point are independent of each other. Thus, the equation can be easily mapped onto one processor each. In the shared memory machine this synchronisation point takes the form of a barrier. Below are given the parallel algorithm of the block method in the code:

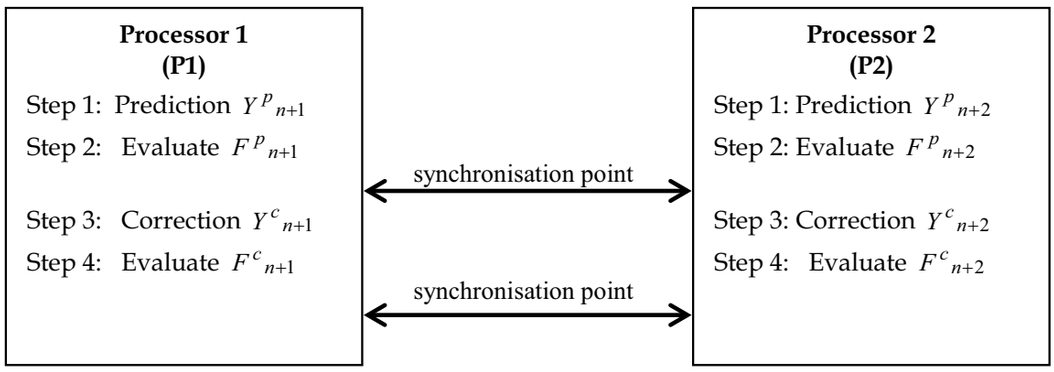


Fig. 2. The parallel process of P2PFDIR.

In Figure 2, both processors have to exchange information after the evaluation of the terms $F^p{}_{n+1}$ and $F^p{}_{n+2}$ before continue to *Step* 3. The same process happen at *Step* 4 after the evaluation of the terms $F^c{}_{n+1}$ and $F^c{}_{n+2}$. The parallelism is achieved when the code computes at *Step* 3 – 4**,** particularly the $Y^c{}_{n+m}$ and $F^c{}_{n+m}$, m=1,2. *Step* 1 – 2 and *Step* 3 – 4 can be done concurrently as they are independent to each other. Parallelization in P2PFDIR is achieved by sharing the f-evaluations.

The sequential programs were executed on DYNIX/ptx operating system. The parallel programs of the method employed were run on a shared memory Sequent Symmetry parallel computer at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. The choice of an implementation on a shared memory parallel computer is due to the fact that such a computer can consists of several processors sharing a common memory with fast data access and requiring less communication times, which is suited to the features of the P2PFDIR method.

The algorithm for P2PFDIR is executed in C language. In order to see a possible speed up of the parallel code, the test problems in Section 5 should be expensive. Therefore, the relatively small problems have been enlarge by scaling. The computation cost increased when solving large systems of higher order ODEs because the function evaluations continue to increase. Using two processors to do the work simultaneously can help to reduce the computation time when solving large problem.

## 5. Results and Discussions

The following two problems were tested using P2PFDIR code and compare the sequential and parallel timing for N=1000, 2000 and 4000 in Problem 1 and N=101 for interval [0, 20] and [0, 40] in Problem 2.

Problem 1: (Lagrange equation for the hanging spring)

$$y_1'' = K^2(-y_1 + y_2)$$
$$y_2'' = K^2(y_1 - 3y_2 + 2y_3)$$
$$y_3'' = K^2(2y_2 - 5y_3 + 3y_4)$$
$$y_N'' = K^2((N-1)y_{N-1} - (2N-1)y_N)$$

$N$ = number of equations, $0 \le x \le b$, $b$ = end of the interval.

$K = 1$, the initial values $y_i(0) = y_i'(0) = 0$ except $y_{N-2}(0) = y_{N-2}'(0) = 1$,

Source: (Hairer et al., 1993).

Problem 2: (Moon – the second celestial mechanics problem.)

$$x_i'' = \gamma \sum_{j=0, j\neq i}^{N} m_j \frac{(x_j - x_i)}{r_{ij}^3}$$

$$y_i'' = \gamma \sum_{j=0, j\neq i}^{N} m_j \frac{(y_j - y_i)}{r_{ij}^3} \quad \text{where } i = 0, \ldots, N$$

$$r_{ij} = \left((x_i - x_j)^2 + (y_i - y_j)^2\right)^{\frac{1}{2}}, \qquad i, j = 0, \ldots, N$$
$$\gamma = 6.672, m_0 = 60, m_i = 7 \times 10^{-3} \quad i = 1, \ldots, N$$

Initial data: $x_0(0) = y_0(0) = x_0'(0) = y_0'(0) = 0$.

$$x_i(0) = 30\cos\left(\frac{2\pi}{100i}\right) + 400, x_i'(0) = 0.8\sin\left(\frac{2\pi}{100i}\right)$$
$$y_i(0) = 30\sin\left(\frac{2\pi}{100i}\right), y_i'(0) = -0.8\cos\left(\frac{2\pi}{100i}\right) + 1$$

$N = 101$, $0 \le t \le b$, $b$ = end of the interval.

Source: (Cong et al., 2000).
The performance of the sequential and parallel execution times for every problem is shown in Table 1 – 5 while Table 6 shows the speed up and efficiency performance for the problems. The notations are defined as follows:

| | |
|---|---|
| TOL | Tolerances |
| MTD | Method employed |
| TS | Total number of steps |
| FS | Failure steps |
| FCN | Total function calls |
| MAXE | Magnitude of the global error $(\max\left\|y_n - y(x_n)\right\|$ |
| TIME(min) | The execution time in minutes |
| TIME(sec) | The execution time in seconds |
| S2PFDIR | Sequential implementation of the two point implicit block method |
| P2PFDIR | Parallel implementation of the two point implicit block method |

In the code, we iterate the corrector to convergence. The convergence test employed were

$$abs(y^{(s+1)}{}_{n+2} - y^{(s)}{}_{n+2}) < 0.1 \times \text{TOL},\ s = 0,1,2,\ldots \tag{10}$$

where $s$ is the number of iteration. After the successful convergence test of (10), local errors estimated at the point $x_{n+2}$ will be performed to control the error for the block. The error controls were at the second point in the block because in general it had given us better results. The local errors estimates will be obtain by comparing the absolute difference of the corrector formula derived of order $k$ and a similar corrector formula of order $k$-1.
In these problems we recall that *speed up* is a measure of the relative benefits of parallelising a given application over sequential implementation. The speed up ratio on two processors that we use is defined as

$$S_p = \frac{T_s}{T_p} \tag{11}$$

where $T_s$ is the time for the fastest serial algorithm for a given problem and $T_p$ is the execution time of a parallel program on multiprocessors and in this research we used $p = 2$. The maximum speed up possible is usually $p$ with processors $p$, normally referred to as *linear speed up*. The *efficiency of the parallel algorithm,* $E_p$, is defined as

$$E_p = \frac{S_p}{p} \times 100 \tag{12}$$

which is the ratio of speed up compared to the number of processors used. In an ideal parallel system, speed up is equal to the number of processors $p$ being used and efficiency is equal to 100%. In practice, speed up is less than $p$ and efficiency is between 0% and 100%,

depending on the degree of effectiveness with which the processors are utilised. The speed up shows the speed gain of the parallel computation and it can describe the increase of performance in the parallel system.
The two problems above were run without exact reference solution in a closed form, so we used the reference solution obtained by the same program using tolerance at two order lower from the current tolerance. The tested problems were run without calculating the maximum error for the execution time of the sequential and parallel. The values of maximum errors were computed in a separate program.
In Table 1 – 5 show the numerical results for the tested problems. For sequential S2PFDIR only one processor was used and two processors were employed for the parallel algorithms of P2PFDIR. The numerical results show that the parallel execution time is faster than the sequential execution time for large ODEs systems. In Table 1 – 3, without loss of generality, we only compute the MAXE at TOL = $10^{-2}$ since the execution time is grossly increased with a finer tolerance.

| TOL | MTD | N=1000, [0, 5] | | | |
|---|---|---|---|---|---|
| | | TS | FS | FCN | TIME (min) |
| $10^{-2}$ | S2PFDIR | 239 | 0 | 1762 | 0.195781 |
| | P2PFDIR | | | 883 | 0.179703 |
| | MAXE=1.15083(-2) | | | | |
| $10^{-4}$ | S2PFDIR | 570 | 1 | 3384 | 0.381467 |
| | P2PFDIR | | | 1698 | 0.354987 |
| $10^{-6}$ | S2PFDIR | 714 | 0 | 5584 | 0.609685 |
| | P2PFDIR | | | 2797 | 0.601066 |
| $10^{-8}$ | S2PFDIR | 1743 | 0 | 10402 | 1.167327 |
| | P2PFDIR | | | 5207 | 1.087472 |
| $10^{-10}$ | S2PFDIR | 4298 | 0 | 25722 | 2.882636 |
| | P2PFDIR | | | 12867 | 2.682821 |

Table 1. Numerical results of 2PFDIR Method for Solving Problem 1 When N=1000.

| TOL | MTD | N=2000, [0, 5] | | | |
|---|---|---|---|---|---|
| | | TS | FS | FCN | TIME (min) |
| $10^{-2}$ | S2PFDIR | 329 | 0 | 2300 | 0.649994 |
| | P2PFDIR | | 0 | 1150 | 0.436222 |
| | MAXE=2.36506(-2) | | | | |
| $10^{-4}$ | S2PFDIR | 796 | 0 | 4734 | 1.360806 |
| | P2PFDIR | | 0 | 2373 | 0.936770 |
| $10^{-6}$ | S2PFDIR | 997 | 0 | 7642 | 2.128587 |
| | P2PFDIR | | 0 | 3801 | 1.449660 |
| $10^{-8}$ | S2PFDIR | 2451 | 0 | 14648 | 4.082667 |
| | P2PFDIR | | 0 | 7330 | 2.874696 |
| $10^{-10}$ | S2PFDIR | 6066 | 0 | 36330 | 10.672470 |
| | P2PFDIR | | 0 | 18171 | 7.236281 |

Table 2. Numerical results of 2PFDIR Method for Solving Problem 1 When N=2000.

| TOL | MTD | N=4000, [0, 5] | | | |
|---|---|---|---|---|---|
| | | TS | FS | FCN | TIME (min) |
| $10^{-2}$ | S2PFDIR | 457 | 0 | 3070 | 1.278904 |
| | P2PFDIR | | | 1530 | 0.683906 |
| | MAXE=4.23114(-2) | | | | |
| $10^{-4}$ | S2PFDIR | 1116 | 1 | 6654 | 2.812752 |
| | P2PFDIR | | | 3323 | 1.488229 |
| $10^{-6}$ | S2PFDIR | 1397 | 0 | 10032 | 4.127480 |
| | P2PFDIR | | | 5012 | 2.195468 |
| $10^{-8}$ | S2PFDIR | 3459 | 0 | 20644 | 8.778264 |
| | P2PFDIR | | | 10318 | 4.524878 |
| $10^{-10}$ | S2PFDIR | 8566 | 0 | 51328 | 21.953364 |
| | P2PFDIR | | | 25658 | 11.258135 |

Table 3. Numerical results of 2PFDIR Method for Solving Problem 1 When N=4000.

| TOL | MTD | N=101, [0, 20] | | | |
|---|---|---|---|---|---|
| | | TS | FS | FCN | TIME (sec) |
| $10^{-2}$ | S2PFDIR | 29 | 0 | 128 | 2.079400 |
| | P2PFDIR | | | 70 | 1.329228 |
| | MAXE=3.78992(-4) | | | | |
| $10^{-4}$ | S2PFDIR | 36 | 0 | 158 | 2.542150 |
| | P2PFDIR | | | 85 | 1.543892 |
| | MAXE=2.40610(-6) | | | | |
| $10^{-6}$ | S2PFDIR | 44 | 0 | 198 | 3.175919 |
| | P2PFDIR | | | 105 | 1.902361 |
| | MAXE=8.76138(-7) | | | | |
| $10^{-8}$ | S2PFDIR | 52 | 0 | 238 | 3.808737 |
| | P2PFDIR | | | 125 | 2.260861 |
| | MAXE=4.36732(-9) | | | | |
| $10^{-10}$ | S2PFDIR | 70 | 0 | 336 | 5.360910 |
| | P2PFDIR | | | 174 | 3.137757 |
| | MAXE=6.78177(-11) | | | | |

Table 4. Numerical results of 2PFDIR Method for Solving Problem 2 When N=101, interval [0, 20].

| TOL | MTD | N=101, [0, 40] | | | |
|---|---|---|---|---|---|
| | | TS | FS | FCN | TIME (sec) |
| $10^{-2}$ | S2PFDIR | 31 | 0 | 136 | 2.201868 |
| | P2PFDIR | | | 74 | 1.395546 |
| | MAXE=1.77673(-4) | | | | |
| $10^{-4}$ | S2PFDIR | 39 | 0 | 176 | 2.835758 |
| | P2PFDIR | | | 94 | 1.704369 |
| | MAXE=1.53896(-6) | | | | |
| $10^{-6}$ | S2PFDIR | 48 | 0 | 224 | 3.674788 |
| | P2PFDIR | | | 118 | 2.133841 |
| | MAXE=1.39440(-7) | | | | |
| $10^{-8}$ | S2PFDIR | 62 | 0 | 296 | 4.720133 |
| | P2PFDIR | | | 154 | 2.779889 |
| | MAXE=1.29065(-8) | | | | |
| $10^{-10}$ | S2PFDIR | 94 | 0 | 478 | 7.599380 |
| | P2PFDIR | | | 245 | 4.409488 |
| | MAXE=1.54553(-10) | | | | |

Table 5. Numerical results of 2PFDIR Method for Solving Problem 2 When N=101, interval [0, 40].

| PROB | N | Interval | TOL | | | | |
|---|---|---|---|---|---|---|---|
| | | | $10^{-2}$ | $10^{-4}$ | $10^{-6}$ | $10^{-8}$ | $10^{-10}$ |
| 1 | 1000 | [0, 5] | 1.09 [55] | 1.07 [54] | 1.01 [51] | 1.07 [54] | 1.07 [54] |
| | 2000 | [0, 5] | 1.49 [75] | 1.45 [73] | 1.47 [74] | 1.42 [71] | 1.52 [76] |
| | 4000 | [0, 5] | 1.87 [94] | 1.89 [95] | 1.88 [94] | 1.94 [97] | 1.95 [98] |
| 2 | 101 | [0, 20] | 1.56 [78] | 1.65 [83] | 1.67 [84] | 1.68 [84] | 1.71 [86] |
| | 101 | [0, 40] | 1.58 [79] | 1.66 [83] | 1.72 [86] | 1.70 [85] | 1.72 [86] |

Table 6. The Speed Up and Efficiency of the 2PFDIR Method for Solving Problem 1 and 2.
Note. For each tolerance the values in the square brackets give the results of the efficiency in percentage.

In Table 6, the speed up ranging between 1.87 and 1.95 for solving Problem 1 when $N = 4000$ and the efficiency is between 94% and 98%. Better speed up and efficiency can be achieved by increasing the dimension of the ODEs. In Problem 2, the speed up ranging between 1.58 and 1.72 as the interval increased at the same number of equations. The number of function evaluations is almost half in the parallel mode compared to the sequential mode. In term of accuracy, numerical results are within the given tolerances. The performance of parallel implementation of an integration method depends heavily on the machine, the size of the problem and the costs of the function evaluation.

## 6. Conclusion

In this paper, we have considered the performances of the parallel direct block code (P2PFDIR) based on the two point implicit block method in the form of Adams Moulton type. By using large problems and by implementing the code on the shared memory computer, we have shown the superiority of the parallel code over the sequential code. The P2PFDIR achieved better speed up and efficiency when the dimension of ODEs systems increased and hence the parallel code developed are suitable for solving large problems of ODEs.

## 7. Acknowledgements

This research was supported by Institute of Mathematical Research, Universiti Putra Malaysia under RU Grant 05-01-07-0232RU.

## 8. Future Work

Instead of implemented the block method using variable step size, it is possible to vary the step size and order of the method in solving the higher order ODEs. Therefore, the approximations for the tested problems will be better in terms of accuracy and less number of solving steps. It would also be interesting to implement the parallel computation to new three and four point direct block method.

## 9. References

Cong, N.H.; Strehmel, K.; Weiner, R. & Podhaisky, H. (1999). Runge–Kutta-Nystrom-type parallel block predictor-corrector methods, *Advances in Computational Mathematics,* Vol 10, No. 2., ( February 1999), pp. 115–133, ISSN 1019-7168.

Cong, N.H.; Podhaisky, H. & Weiner, R. (2000). Performance of explicit pseudo two-step RKN methods on a shared memory computer, *Reports on Numerical Mathematics*, No. 00-21, Dept. Math & Computer Science, Martin Luther University Halle-Wittehberg. (http://www.mathematik.uni-halle.de/reports/rep-num.html)

Fatunla, S.O. (1990). Block Methods for Second Order ODEs. *Intern. J. Computer Math,* Vol 40, pp. 55-63, ISSN 0020-7160.

Hairer, E.; Norsett, S.P. & Wanner, G. (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems,* Berlin: Springer-Verlag, ISSN 3-540-17145-2, Berlin Heidelberg New York.

Majid, Z.A. & Suleiman, M. (2006). Direct Integration Implicit Variable Steps Method for Solving Higher Order Systems of Ordinary Differential Equations Directly, *Jurnal Sains Malaysiana*, Vol 35, No. 2., (December 2006), pp 63-68. ISSN 0126-6039.

Majid, Z.A. & Suleiman, M. (2007). Two point block direct integration implicit variable steps method for solving higher order systems of ordinary differential equations, *Proceeding of the World Congress on Engineering 2007,* pp. 812-815, ISBN 978-988-98671-2-6, London, July 2007, Newswood Limited, Hong Kong.

Omar, Z. (1999). Developing Parallel Block Methods For Solving Higher Order ODEs Directly, Ph.D. Thesis, University Putra Malaysia, Malaysia.

Suleiman, M.B. (1989). Solving Higher Order ODEs Directly by the Direct Integration Method, *Applied Mathematics and Computation*, Vol. 33, No. 3., (October 1989), pp 197-219, ISSN 0096-3003.


**Advanced Technologies**

Edited by Kankesu Jayanthakumaran

ISBN 978-953-307-009-4
Hard cover, 698 pages
**Publisher** InTech
**Published online** 01, October, 2009
**Published in print edition** October, 2009

This book, edited by the Intech committee, combines several hotly debated topics in science, engineering, medicine, information technology, environment, economics and management, and provides a scholarly contribution to its further development. In view of the topical importance of, and the great emphasis placed by the emerging needs of the changing world, it was decided to have this special book publication comprise thirty six chapters which focus on multi-disciplinary and inter-disciplinary topics. The inter-disciplinary works were limited in their capacity so a more coherent and constructive alternative was needed. Our expectation is that this book will help fill this gap because it has crossed the disciplinary divide to incorporate contributions from scientists and other specialists. The Intech committee hopes that its book chapters, journal articles, and other activities will help increase knowledge across disciplines and around the world. To that end the committee invites readers to contribute ideas on how best this objective could be accomplished.

**How to reference**

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Zanariah Abdul Majid and Mohamed Suleiman (2009). Parallel Direct Integration Variable Step Block Method for Solving Large System of Higher Order Ordinary Differential Equations, Advanced Technologies, Kankesu Jayanthakumaran (Ed.), ISBN: 978-953-307-009-4, InTech, Available from: http://www.intechopen.com/books/advanced-technologies/parallel-direct-integration-variable-step-block-method-for-solving-large-system-of-higher-order-ordi

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen