Malaysian Journal of Mathematical Sciences 5(2): 143-160 (2011)

Parallel Implementation of Explicit 2 and 3-Point Block Method for Solving System of Special Second Order ODEs Directly

[1]Yap Lee Ken, [2,3]Fudziah Ismail,
[2,3]Zanariah Majid and [3,4]Mohamed Othman

[1]Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, 53300 Setapak, Kuala Lumpur, Malaysia

[2]Faculty of Science,Universiti Putra Malaysia 43400 UPM Serdang Selangor, Malaysia

[3]Institute for Mathematical Research, Universiti Putra Malaysia 43400 UPM Serdang, Selangor, Malaysia

[4]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

E-mail: fudziah@math.upm.edu.my

ABSTRACT

In this paper the explicit 2 and 3-point block method for solving large systems of special second order ODEs directly is discussed. Codes based on the methods are executed in sequential and parallel. The numerical results show that parallel execution of the methods is more efficient compared to sequential counterpart for solving the large system of special second order ODEs.

INTRODUCTION

The desire for parallel Initial Value Problem (IVP) solvers arises from the need to solve many significant problems faster than is currently possible. This is because the computational time on conventional sequential machine is so large that it affects the productivity of engineers and researchers.

According to Gear and Xuhai (1993) parallelism in IVP solvers can be classified into two main categories

- Parallelism across the method or equivalently parallelism across time
- Parallelism across the system or equivalently parallelism across space.

Parallelism across the method is the possibility of distributing the computational effort of each integration step among the processors. Parallelism across the problem is the possibility of partitioning the system of equations by assigning one single equation or block of them to a processor for concurrent integration. Here we are focusing on the first type of parallelism that is parallelism across the method. However most methods for solving IVPs are by nature step by step method the approximation to the solution of the IVP by methods like Runge-Kutta (RK) and Multistep method evolves one point at a time, which makes them less favourable for parallel implementation. This situation requires the construction of new methods specifically designed for parallel execution.

Burrage (1993) and Cash (1985) have proposed the used of parallel Runge-Kutta methods for solving first order ODEs. Cash (1985) derived explicit and diagonally implicit block Runge-Kutta method which can be exploited for the purpose of parallel implementation. Majid and Suleiman (2009) proposed block method for solving first order ODEs which can be executed in parallel. In this paper we have developed sequential and parallel algorithms based on the method developed by Yap *et. al* (2008) and used them to solve special second order initial value problems. We hope that by parallelizing the algorithms, a more effective code can be developed.

PROBLEM DESCRIPTION AND OBJECTIVES

Birta and Abou-Rabia (1987) stated that parallelism across the method for the solution of ODEs has its basis in a class of techniques referred to as block methods. Block methods have been discussed in detail in Majid and Suleiman (2009) and Omar *et al.* (2002).

Assume that the system to be solved is described by N (the number of equations in the system) special second order ODEs of the form

$$y'' = f(x, y), \quad y(a) = \eta, \quad y'(a) = \eta', \quad a \le x \le b. \tag{1}$$

The *r*-point *k*-block method for Equation (1) is represented by the computation scheme

$$A^{(0)}Y_m = \sum_{i=1}^{k} A^{(i)}Y_{m-i} + h^2 \sum_{i=0}^{k} B^{(i)}F_{m-i} \tag{2}$$

where $Y_m = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_{n+r} \end{bmatrix}$, $F_m = \begin{bmatrix} f_{n+1} \\ f_{n+2} \\ \vdots \\ f_{n+r} \end{bmatrix}$ (for $n = mr, m = 0,1\ldots$), $A^{(i)}, B^{(i)}$ are *r* by *r* matrices. The block scheme is explicit if the coefficient matrix $B^{(0)}$ is a null matrix.

In the explicit *r*-point block method, the interval $[a,b]$ is divided into series of blocks with each block containing *r* points. Each application of the formulae generates a block of *r* new equally spaced solution values simultaneously. The computational tasks at each point within a block are sufficiently independent and considered as separate task. Therefore, the computation tasks can be performed simultaneously by assigning the tasks to different processors.

Yap *et al.* (2008), have derived the explicit block methods of order four based on Newton-Gregory backward difference formula designed for special second order ODEs. The methods are:

Explicit 2-Point Block Method:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_{n+1} \\ y_{n+2} \end{bmatrix} = \begin{bmatrix} -1 & 2 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} y_{n-1} \\ y_n \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} y_{n-3} \\ y_{n-2} \end{bmatrix} + h^2\left(\begin{bmatrix} -\frac{5}{12} & \frac{7}{6} \\ -\frac{20}{3} & \frac{20}{3} \end{bmatrix}\begin{bmatrix} f_{n-1} \\ f_n \end{bmatrix} + \begin{bmatrix} -\frac{1}{12} & \frac{1}{3} \\ -\frac{4}{3} & \frac{16}{3} \end{bmatrix}\begin{bmatrix} f_{n-3} \\ f_{n-2} \end{bmatrix}\right)$$

Explicit 3-Point Block Method:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ y_{n+3} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} y_{n-5} \\ y_{n-4} \\ y_{n-3} \end{bmatrix}$$
$$+ h^2 \left(\begin{bmatrix} \frac{1}{3} & -\frac{5}{12} & \frac{7}{6} \\ \frac{16}{3} & -\frac{20}{3} & \frac{20}{3} \\ 27 & -\frac{135}{4} & \frac{45}{2} \end{bmatrix} \begin{bmatrix} f_{n-2} \\ f_{n-1} \\ f_n \end{bmatrix} + \begin{bmatrix} 0 & 0 & -\frac{1}{12} \\ 0 & 0 & -\frac{4}{3} \\ 0 & 0 & -\frac{27}{4} \end{bmatrix} \begin{bmatrix} f_{n-5} \\ f_{n-4} \\ f_{n-3} \end{bmatrix} \right)$$

PARALLEL IMPLEMENTATION

Parallel Implementation of the Explicit 2-Point Block Method

The sequential implementation of the explicit 2-point block method is shown in Figure 1. In the sequential algorithm, one processor P_0 (the first processor is always denoted as P_0) is used to calculate the first point y_{n+1} and followed by the evaluation of the second point y_{n+2}. The function evaluation f_{n+1} is then computed using y_{n+1} and followed by function evaluation f_{n+2} using y_{n+2}.

$$P_0 \longrightarrow y_{n+1} \longrightarrow y_{n+2} \longrightarrow f_{n+1} \longrightarrow f_{n+2} \longrightarrow \text{Results}$$

Figure 1: Sequential Implementation of Explicit 2-Point Block Method

On the other hand, in the parallel algorithm, y_{n+1} and y_{n+2} are assigned to two processors P_0 and P_1 as shown in Figure 2. Each processor is responsible for computing the assigned y. All function evaluations are performed simultaneously.

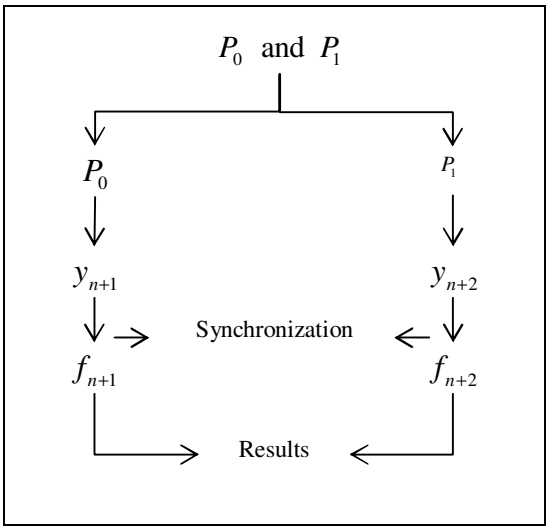

Figure 2: Parallel Implementation of Explicit 2-Point Block Method

Master (*Processor 1)*
Compute *N* equations of *y* at first point
Send *N* equations of *y* at first point to worker
Receive *N* equations of *y* at second point from worker
Call *Function Evaluation* at first point
Send function evaluation at first point to worker
Receive function evaluation at second point from worker

Worker (*Processor 2)*
Compute *N* equations of *y* at second point
Send *N* equations of *y* at second point to master
Receive *N* equations of *y* at first point from master
Call *Function Evaluation* at second point
Send function evaluation at second point to master
Receive function evaluation at first point from master

Figure 3: Pseudo-code of the Parallel Implementation of the Explicit 2-Point Block Method

Parallel Implementation of Explicit 3-Point Block Method

In the sequential algorithm, explicit 3-point block method is implemented in the manner as shown in Figure 4. One processor P_0 is used to calculate

value y at the first point, followed by y at the second point and at the third point. Then, function evaluations f_{n+1}, f_{n+2} and f_{n+3} are computed.

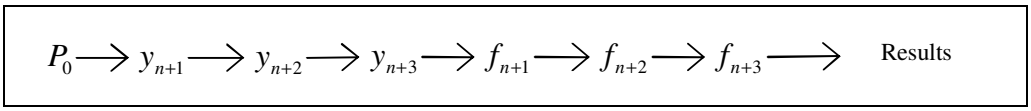

Figure 4: Sequential Implementation of Explicit 3-Point Block Method

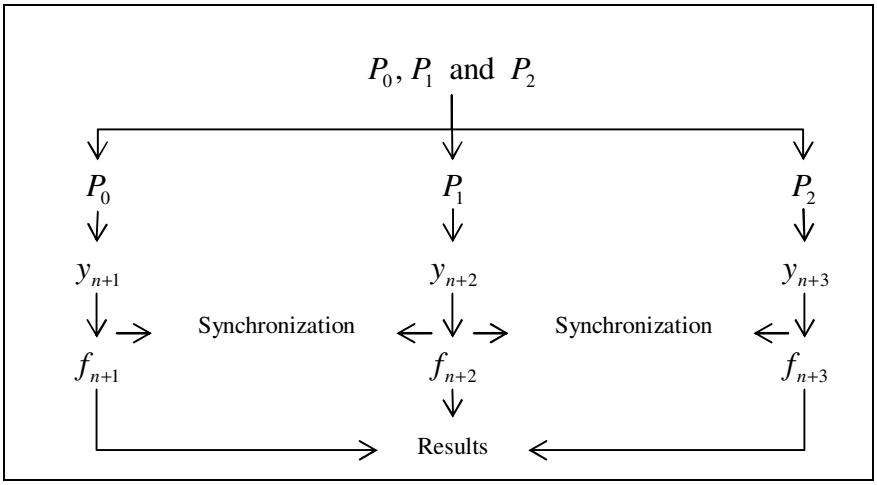

Figure 5: Parallel Implementation of Explicit 3-Point Block Method

Master (*Processor 1)*
Compute *N* equations of *y* at first point
Send *N* equations of *y* at first point to worker 1
Send *N* equations of *y* at first point to worker 2
Receive *N* equations of *y* at second point from worker 1
Receive *N* equations of *y* at third point from worker 2
Call *Function Evaluation* at first point
Send function evaluation at first point to worker 1
Send function evaluation at first point to worker 2
Receive function evaluation at second point from worker 1
Receive function evaluation at third point from worker 2

Figure 6: Pseudo-code of the Parallel Implementation of the Explicit 3-Point Block Method

Worker 1 (*Processor 2)*
Compute *N* equations of *y* at second point
Send *N* equations of *y* at second point to master
Send *N* equations of *y* at second point to worker 2
Receive *N* equations of *y* at first point from master
Receive *N* equations of *y* at third point from worker 2

Call *Function Evaluation* at second point
Send function evaluation at second point to master
Send function evaluation at second point to worker 2
Receive function evaluation at first point from master
Receive function evaluation at third point from worker 2

Worker 2 (*Processor 3)*
Compute *N* equations of *y* at third point
Send *N* equations of *y* at third point to master
Send *N* equations of *y* at third point to worker 1
Receive *N* equations of *y* at first point from master
Receive *N* equations of *y* at second point from worker 1
Call *Function Evaluation* at third point
Send function evaluation at third point to master
Send function evaluation at third point to worker 1
Receive function evaluation at first point from master
Receive function evaluation at second point from worker 1

Figure 6 (continued): Pseudo-code of the Parallel Implementation of the Explicit 3-Point Block Method

The parallel explicit 3-point block method is implemented by assigning the computational tasks for each point to three processors. Each processor calculates the value y and the function evaluation f simultaneously as shown in Figure 5. In the parallel program, the computed y and f must be made available to all participating processors by the blocking send and receive operations.

TEST PROBLEMS

In this section, we present numerical results when a standard set of problems are solved using the methods given in previous section. Before tabulating the numerical results, let us review the metric used to measure the performance of the sequential and parallel algorithms.

For sequential programs, the metric to measure the performance is the sequential time, t_s, which is the time period elapsed between the beginning and the end of the algorithm execution.

For parallel programs, the metrics for measuring performances are as follows:

1. The number of processors, p used.
2. Parallel time, t_p that is the time period elapsed between the beginning of the first processor and the end of the last processor during the execution of the algorithm.
3. Speed-up, S_p compares the parallel running time, t_p of an algorithm that uses p processors to solve a particular problem, to the sequential running time, t_s, which is given by:

$$S_p = \frac{t_s}{t_p}.$$

It can also be defined as the ratio of the execution time of the parallel algorithm on a single processor and the execution time of the parallel algorithm on p processors, that is:

$$S_p = \frac{t_{p=1}}{t_p}.$$

4. $E_p = \frac{S}{p} = \frac{t_s}{pt_p} = \frac{t_{p=1}}{pt_p}.$

E_p is the efficiency of the parallel algorithm and it must be less or equal to one ($E_p \leq 1$). If $E_p = 1$, the speed-up is said to be perfect. Perfect speed-up is rarely ever achievable.

To evaluate the performance of the parallel explicit block methods, an extensive set of numerical experiments has been carried out on a Sun Fire V1280 system equipped with 8 UltraSPARC III Cu processors at 1.2 GHz. These experiments were based on a collection of two test problems as given below.

Problem 1: Lagrange equation for the hanging string

$$
\begin{aligned}
y_1'' &= K^2\left(-y_1 + y_2\right)\\
y_2'' &= K^2\left(y_1 - 3y_2 + 2y_3\right)\\
y_3'' &= K^2\left(2y_2 - 5y_3 + 3y_4\right)\\
&\vdots\\
y_N'' &= K^2\left((N-1)\,y_{N-1} - (2N-1)\,y_N\right)
\end{aligned}
$$

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

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

Source: Majid and Suleiman (2009).

Problem 2: Moon – the celestial mechanics problem

$$
\begin{aligned}
x_i'' &= \gamma \sum_{j=0,\, j\neq i}^{N} m_j \frac{\left(x_j - x_i\right)}{r_{ij}^{\,3}}\\
y_i'' &= \gamma \sum_{j=0,\, j\neq i}^{N} m_j \frac{\left(y_j - y_i\right)}{r_{ij}^{\,3}}
\end{aligned}
\qquad \text{where } i = 0,1,\ldots,N
$$

$$
r_{ij} = \left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)^{\frac{1}{2}}, \qquad i, j = 0,1,\ldots,N
$$

$$
\gamma = 6.672, \quad m_0 = 60, \quad m_i = 7\times 10^{-3},\ i = 1,2,\ldots,N.
$$

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

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

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

$N =$ Number of equations, $0 \le t \le b$, $b =$ end of the interval

Source: Majid and Suleiman (2009).

NUMERICAL RESULTS

Tables 1 and 2 give the performance comparison between the sequential and parallel block methods for solving large system of equations in terms of the total number of steps taken and execution time. The notations used in the tables are as follows:

h	Step size used.
METHOD	Method employed.
TSTEP	Total number of steps taken to obtain the solution.
TIME	Execution time taken in seconds.
SE1PN	Sequential implementation of explicit 1-point method based on *Newton-Gregory backward interpolation formula*.
SE2PBN	Sequential implementation of explicit 2-point block method based on *Newton-Gregory backward interpolation formula*.
PE2PBN	Parallel implementation of explicit 2-point block method based on *Newton-Gregory backward interpolation formula*.
SE3PBN	Sequential implementation of explicit 3-point block method based on *Newton-Gregory backward interpolation formula*.

PE3PBN Parallel implementation of explicit 3-point block method based on *Newton-Gregory backward interpolation formula*.

Figures 7 to 8 demonstrate the speedup comparison between parallel 2-point and 3-point block methods as the number of equations increase. The speedups versus the number of processors with explicit block methods are shown in Figures 9 to 10. On the other hand, Figures 11 to 12 show the efficiency comparison between parallel 2-point and 3-point block methods as the number of equations increase. For the speedup and efficiency comparisons, we do it only for $h=10^{-5}$ because the results are quite similar for other value of *h*.

TABLE 1: Performance comparison between sequential and parallel explicit block methods for solving Problem 1 when N=3000, b=1

h	METHOD	TSTEP	TIME (seconds)
10^{-2}	E1PN	100	0.136
	SE2PBN	52	0.125
	PE2PBN	52	0.087
	SE3PBN	36	0.136
	PE3PBN	36	0.079
10^{-3}	E1PN	1000	1.089
	SE2PBN	502	0.884
	PE2PBN	502	0.614
	SE3PBN	336	0.865
	PE3PBN	336	0.489
10^{-4}	E1PN	10000	10.800
	SE2PBN	5002	8.647
	PE2PBN	5002	5.979
	SE3PBN	3336	8.328
	PE3PBN	3336	4.615
10^{-5}	E1PN	100000	107.903
	SE2PBN	50002	86.288
	PE2PBN	50002	59.532
	SE3PBN	33336	82.976
	PE3PBN	33336	45.440

TABLE 2: Performance comparison between sequential and parallel explicit block methods for solving Problem 2 when N=100, b=1

h	METHOD	TSTEP	TIME (seconds)
10^{-2}	E1PN	100	1.960
	SE2PBN	52	1.956
	PE2PBN	52	1.072
	SE3PBN	36	1.988
	PE3PBN	36	0.804
10^{-3}	E1PN	1000	18.959
	SE2PBN	502	18.285
	PE2PBN	502	9.208
	SE3PBN	336	18.279
	PE3PBN	336	6.230
10^{-4}	E1PN	10000	181.641
	SE2PBN	5002	181.327
	PE2PBN	5002	91.024
	SE3PBN	3336	181.239
	PE3PBN	3336	60.679
10^{-5}	E1PN	100000	1815.485
	SE2PBN	50002	1810.101
	PE2PBN	50002	906.866
	SE3PBN	33336	1809.905
	PE3PBN	33336	605.816

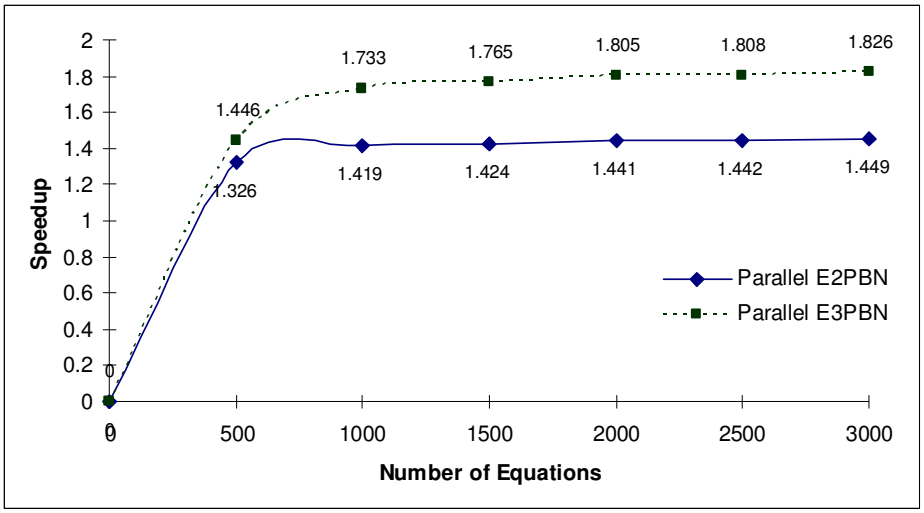

Figure 7: Speedup Comparison between PE2PBN and PE3PBN for Solving Problem 1 when $h = 10^{-5}$

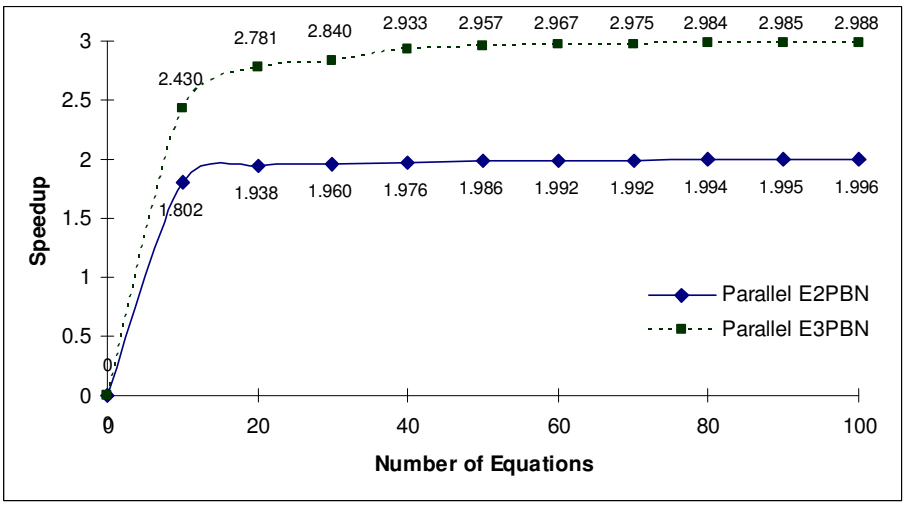

Figure 8: Speedup Comparison between PE2PBN and PE3PBN for Solving Problem 2 when $h = 10^{-5}$

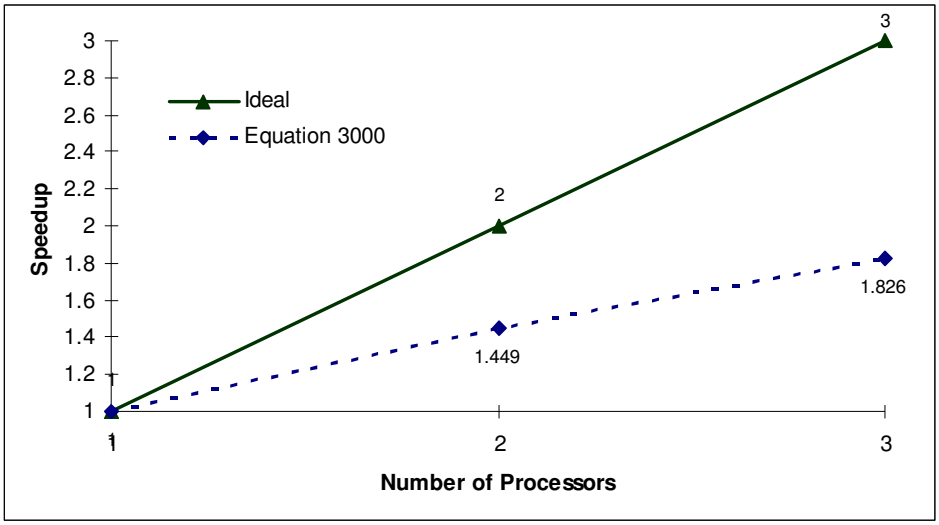

Figure 9: Speedup versus Number of Processors with Explicit Block Methods for Solving Problem 1 when $h = 10^{-5}$

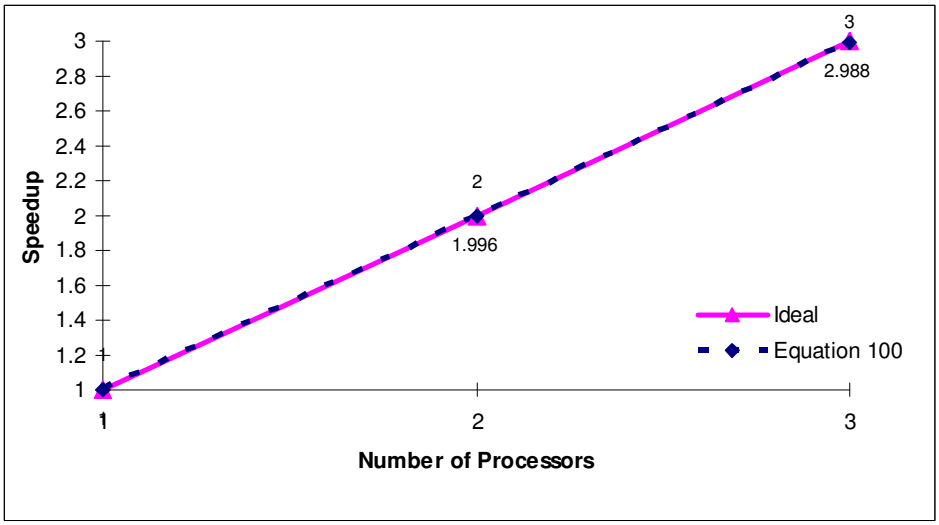

Figure 10: Speedup versus Number of Processors with Explicit Block Methods for Solving Problem 2 when $h = 10^{-5}$

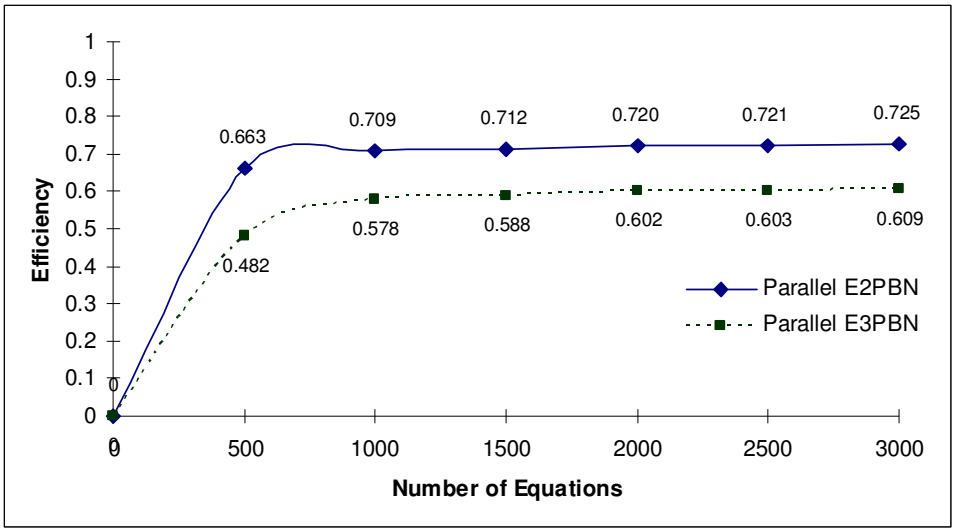

Figure 11: Efficiency Comparison between PE2PBN and PE3PBN for Solving Problem 1 when $h = 10^{-5}$

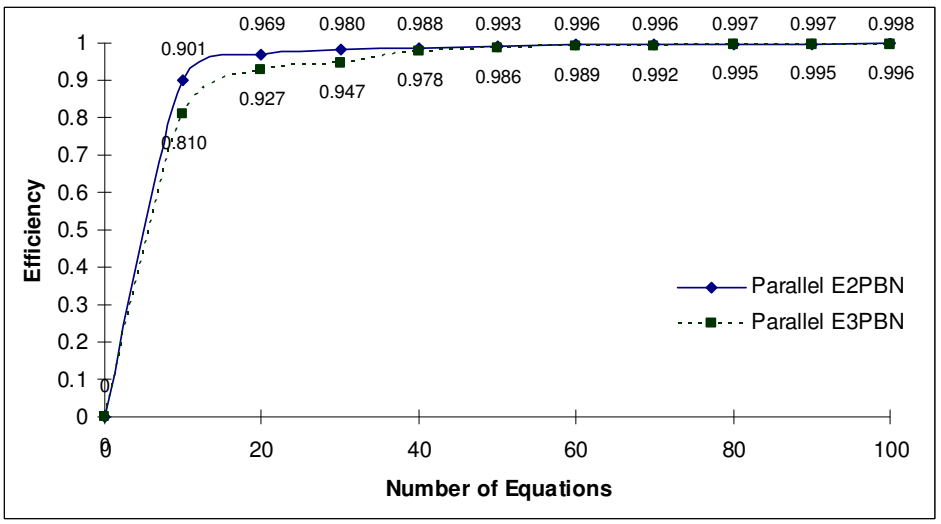

Figure 12: Efficiency Comparison between PE2PBN and PE3PBN for Solving Problem 2 when $h = 10^{-5}$

DISCUSSION AND CONCLUSION

The focus here is the performance metrics of the parallel and sequential codes namely the total number of steps taken, execution time, speedup and efficiency.

The results in the Tables 1 and 2 indicate that the sequential and parallel implementation for explicit block methods are superior compared to the non-block counterparts in term of total number of steps taken to obtain the solution. The 2-point and 3-point block methods reduce the total steps to almost one half and one third respectively compared to 1-point method.

The experiments with the block and non-block methods are performed using the interval $[0,1]$. Since the integration interval does not influence the outcome of the performance comparison, this interval is chosen such that all experiments could be completed within a reasonable amount of time.

The parallel implementations achieve better execution time than the sequential implementations when tested on large systems of equations. The primary reason for the better execution time is the computation tasks within a block are carried out simultaneously on separate processors. The parallel implementations achieve the improvement greater than 30% when solving Problem 1 and 50% when solving Problem 2.

Figures 7 and 8 show the speedup values measured with the parallel implementations. The best execution time of the sequential codes is used as a reference for the speedup calculation.

The speedup values gained by the parallel algorithms with respect to number of equations for step size $h = 10^{-5}$ are shown in Figures 7 and 8. The results indicated that for a fixed number of processors, as the number of equations increases, the speedup increases.

In general, the speedups gained by the parallel explicit 3-point block (PE3PB) are higher than the parallel explicit 2-point block (PE2PB) when solving large systems of equations.

Figures 9 and 10 demonstrate that the speedup increases linearly with the number of processors for the case $N = 100$ in problem 2. Overall, we can conclude that the algorithm is highly parallel and has clear

superiority over sequential approach particularly as the number of equations increase as in Problem 2. The reason why the speedup is better for Problem 2 compared to Problem 1 is that in Problem 2 the function evaluation involved the calculation of distance *r,* hence more computational tasks are needed which are assigned to different processors concurrently. Thus, the parallel implementations work efficiently compared to the sequential one.

Figures 11 to 12 indicated that large number of equations lead to better efficiency. It is apparent that the PE2PB methods have better efficiency compared to PE3PB methods when solving Problem 1. The principal reason for the efficiency in PE3PB code is parallel overhead. This could be time spent in process startup and interprocess communication when solving Problem 1.

However the efficiencies of parallel implementation using two and three processors are similar when solving Problem 2. Both the parallel 2-point block and 3-point block methods gain the efficiency to almost one. Therefore, more computation tasks are involved when solving Problem 2 compared to Problem 1.

In the parallel algorithms, the computational tasks are assigned to different processors concurrently. Hence, the parallel implementations work efficiently when solving Problem 2 compared to sequential implementations.

Hence we can conclude that the parallel 2-point and 3-point block methods have shown superiority in terms of total steps, execution time, speedup and efficiency over the 1-point method for solving large system of equations.

REFERENCES

Birta, L.G. and Abou-Rabia, O. 1987. Parallel Block Predictor-Corrector Methods for ODEs. *IEEE Transactions on Computers.* **C-36**(3): 299-311.

Burrage, K. 1993. Parallel Methods for Initial Value Problems. *Applied Numerical Mathematics.* **11**: 5-25.

Cash, J. R. 1985. Block embedded explicit Runge-Kutta methods. *Comput. Math Appl.* **11**: 395-409.

Gear, C. W. and Xuhai, X. 1993. Parallelism Across Time in ODEs. *Applied Numerical Mathematic.* **11**: 45-68.

Majid, Z. and Suleiman, M. 2009. Parallel Direct Integration Variable Step Block Method for Solving Large Systems of Higher Order Ordinary Differential Equations, *International Journal of Math and Statistical Sciences.* **1**(1): 9-12.

Omar, Z., Suleiman, M., Saman, M. Y. and Evans, D. J. 2002. Parallel R-point Explicit Block Methods for Solving Second Order ODEs directly. *International Journal of Computer Math.* **79**(3): 289-298.

Yap Lee Ken, Ismail, F., Suleiman M and Md. Amin, S. 2008. Block Methods Based on Newton Interpolations for Solving Special Second Order Ordinary Differential Equations Directly, *International Journal of Mathematics and Statistics*. **4**(3): 174-180.