Implementation of MPI environment for solving large systems of ODEs using block method

[1] Z. A. MAJID, [2] S. MEHRKANOON, [3] K. I. OTHMAN, [4] M. SULEIMAN

[1,2,4] Mathematics Department, Faculty of Science
Universiti Putra Malaysia
43400 UPM Serdang, Selangor
MALAYSIA

[1,3,4] Institute Mathematical Research
Universiti Putra Malaysia
43400 UPM Serdang, Selangor
MALAYSIA

[3] Mathematics Department, Faculty of Information Technology and Science Quantitative
Universiti Technology MARA
40450 Shah Alam, Selangor
MALAYSIA
zanariah@math.upm.edu.my http://profile.upm.edu.my/am_zana/en/profail.html

Abstract: - Parallel 2-point and 3-point block method will simultaneously compute the numerical solutions at two and three points respectively are suitable for solving large system of first order ordinary differential equations (ODEs) using variable step size. The Gauss Seidel iteration will be implemented for the block methods. The parallelism across the system is considered for the parallelization of the proposed methods using the Message Passing Interface (MPI) communication environment which runs on High Performance Computing (HPC). Numerical examples are given to illustrate the efficiency of the parallel implementation of the proposed method using 2, 4 and 6 processors with respect to the sequential one.

Key-Words: - 2-point block, 3-point block, ODEs, parallel method, block method, parallelism

1 Introduction

The parallel solution of ODEs has received interest from many researchers due to the possibility of using parallel computing platforms. This paper is concerned with development of parallel block code for solving system of non-stiff first order ODEs of the form

$$Y' = F(x, Y), \quad Y(a) = Y_0, \quad a \le x \le b \qquad (1)$$

where a and b are finite and

$$Y' = [y_1', y_2', \ldots, y_n']^T,$$
$$Y = [y_1, y_2, \ldots, y_n]^T,$$
$$F = [f_1, f_2, \ldots, f_n]^T.$$

Block methods for numerical solution of first order ODEs have been proposed by several authors such as in [1], [2],[5] and [6]. There exist many parallel approaches in solving real world problem such as in [3], [4], [5], [7], [8], [9], [11], [12], [13] and [14]. The parallel block code for solving first and higher order ODEs using variable step size and order has been developed in [14]. Alternating group method was proposed in [8] to solve fourth order parabolic equation in parallel environment. The parallel fifth order diagonally Runge-Kutta method mentioned in [11] was implemented by constant step size to solve ordinary differential equations using three processors. Reference in [13] have proposed three point block methods for solving large system of ODEs and Jacobi iteration was used for the implementation of the block method. These codes utilized the same approach as in [14] for parallelization i.e parallelism across the method.

The aim of this paper is to introduce the parallel two point and three point block methods for solving large system of ODEs and the Gauss Seidel iteration

is considered for the implementation of the methods. The parallelism across the system is applied to the proposed codes.

2 Formulation of the block methods

2.1 Derivation of 2-point block method

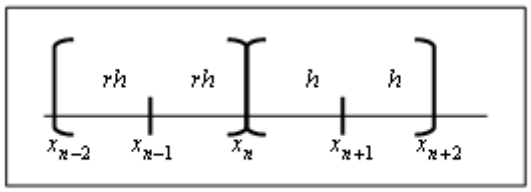

Fig. 1:2- point block method

In Fig. 1, the solutions of y_{n+1} and y_{n+2} with step size h at the points x_{n+1} and x_{n+2} respectively were approximated simultaneously using three back values at the points x_n, x_{n-1} and x_{n-2} of the previous two steps with step size rh. The method computes two points concurrently using two earlier steps. The interpolation points involved for obtaining the corrector formulae are $\{(x_{n-2}, f_{n-2}), \ldots, (x_{n+2}, f_{n+2})\}$.

The two values of $\{y_{n+j}\}_{j=1}^{2}$ can be derived by integrating (1) over the interval $[x_n, x_{n+1}]$ and $[x_{n+1}, x_{n+2}]$. After simplifying using MATHEMATICA one can obtain the following corrector formulae in terms of r:

1 st point:

$$y(x_{n+1}) = y(x_n) + h\left[\frac{(3+15r+20r^2)}{240(r+1)(r+2)} f_{n+2} \right.$$
$$\frac{(18+75r+80r^2)}{60(r+1)(2r+1)} f_{n+1} + \frac{(7+45r+100r^2)}{240r^2} f_n$$
$$\left. - \frac{(7+30r)}{60(r+1)(r+2)} f_{n-1} + \frac{(7+15r)}{240(r+1)(2r+1)r^2} f_{n-2} \right]. \quad (2)$$

2 nd point:

$$y(x_{n+2}) = y(x_{n+1}) + h\left[\frac{(147+255r+100r^2)}{240(r+1)(r+2)} f_{n+2} + \right.$$
$$\frac{(78+165r+80r^2)}{60(2r+1)(r+1)} f_{n+1} + \frac{(23+45r+20r^2)}{240r^2} f_n$$
$$\left. + \frac{(23+30r)}{60r^2(r+1)(r+2)} f_{n-1} - \frac{(23+15r)}{240r^2(2r+1)(r+1)} f_{n-2} \right] \quad (3)$$

The predictor formulae were derived similarly as above and the order is one less than the corrector.

2.2 Derivation of 3-point block method

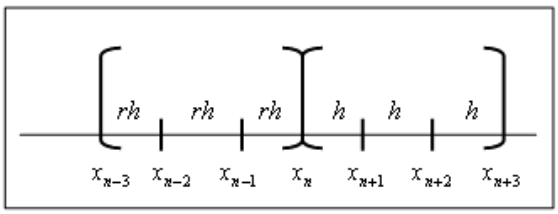

Fig. 2: 3-point block method

In Fig. 2, the solutions of y_{n+1}, y_{n+2} and y_{n+3} with step size h at the points x_{n+1}, x_{n+2} and x_{n+3} respectively were approximated simultaneously using four back values at the points x_n, x_{n-1}, x_{n-2} and x_{n-3} of the previous three steps with step size rh. The interpolation points involved for obtaining the corrector formulae are $\{(x_{n-3}, f_{n-3}), \ldots, (x_{n+3}, f_{n+3})\}$.

The three values of $\{y_{n+j}\}_{j=1}^{3}$ can be derived by integrating (1) over the interval $[x_n, x_{n+1}]$, $[x_{n+1}, x_{n+2}]$ and $[x_{n+2}, x_{n+3}]$. After simplifying using MATHEMATICA one can obtain the following corrector formulae in terms of r:

1 st point:

$$y_{n+1} = y_n + h\left[-\frac{66+357r+532r^2}{7560r^3(1+r)(1+3r)(2+3r)} f_{n-3} + \right.$$
$$\frac{33+238r+399r^2}{840r^3(1+r)(1+2r)(3+2r)} f_{n-2} -$$
$$\frac{66+595r+1596r^2}{840r^3(1+r)(2+r)(3+r)} f_{n-1} +$$
$$\frac{33+357r+1463r^2+2835r^3}{7560r^3} f_n +$$
$$\frac{214+1680r+4389r^2+3990r^3}{840(1+r)(1+2r)(1+3r)} f_{n+1} -$$
$$\frac{16+147r+462r^2+525r^3}{840(1+r)(2+r)(2+3r)} f_{n+2} +$$
$$\left. \frac{18+168r+539r^2+630r^3}{7560(1+r)(3+r)(3+2r)} f_{n+3} \right]. \quad (4)$$

2 nd point:

$$y_{n+2} = y_{n+1} + h\left[\frac{354+693r+308r^2}{7560r^3(1+r)(1+3r)(2+3r)}f_{n-3} - \frac{59+154r+77r^2}{280r^3(1+r)(1+2r)(3+2r)}f_{n-2} + \frac{118+385r+308r^2}{280r^3(1+r)(2+r)(3+r)}f_{n-1} - \frac{177+693r+847r^2+315r^3}{7560r^3}f_n + \frac{398+1680r+2233r^2+910r^3}{280(1+r)(1+2r)(1+3r)}f_{n+1} + \frac{368+1281r+1386r^2+455r^3}{280(1+r)(2+r)(2+3r)}f_{n+2} - \frac{402+1512r+1771r^2+630r^3}{7560(1+r)(3+r)(3+2r)}f_{n+3}\right]. \quad (5)$$

3 rd point:

$$y_{n+3} = y_{n+2} + h\left[-\frac{1746+2037r+532r^2}{7560r^3(1+r)(1+3r)(2+3r)}f_{n-3} + \frac{873+1358r+399r^2}{840r^3(1+r)(1+2r)(3+2r)}f_{n-2} - \frac{1746+3395r+1596r^2}{840r^3(1+r)(2+r)(3+r)}f_{n-1} + \frac{873+2037r+1463r^2+315r^3}{7560r^3}f_n - \frac{2866+6720r+4851r^2+1050r^3}{840(1+r)(1+2r)(1+3r)}f_{n+1} + \frac{4744+11613r+8778r^2+1995r^3}{840(1+r)(2+r)(2+3r)}f_{n+2} + \frac{19338+42168r+28259r^2+5670r^3}{7560(1+r)(3+r)(3+2r)}f_{n+3}\right]. \quad (6)$$

The order of the predictor formulae were one less than the corrector.

3 Implementation

During the implementation of the method, the choice for next step size will be limited to half, double or the same as the current step size. In the developed code, when the next step size is double, the ratio r is 0.5 and q can be 0.5 or 0.25, but if the next step sizes remain constant, r is 1 and q can be 1, 2 or 0.5. In case of step size failure, r is 2, and q is 2. In order to reduce the computational cost, all the coefficients of the formula are stored in the developed code.

The method are implemented in $PE(CE)^m$ mode where P stands for an application of the predictor, E stands for an evaluation of the function F, and C stands for an application of the corrector. During the implementation, the iteration will involve the Gauss Seidel style.

The error calculated are defined as

$$(e_i)_t = \left|\frac{(y_i)_t - (y(x_i))_t}{A + B(y(x_i))_t}\right|$$

where $(y)_t$, is the t^{th} component of the approximate y. A = 1, B = 0 corresponds to the absolute error test, A = 0, B = 1 corresponds to the relative error test and finally A = 1, B = 1 corresponds to the mixed error test. The mixed error test is used for all tested problems.

The maximum error is defined as follow:

$$\text{MAXE} = \max_{1\le i\le SSTEP}\left(\max_{1\le i\le N}(e_i)_t\right)$$

where N is the number of equation in the system.

In the code, we iterate the corrector to convergence using the convergence criteria:

$$\left|y^{t+1}{}_{n+i} - y^{t}{}_{n+i}\right| < 0.1 \times \text{Tolerance}$$

where $i = 2,3$ and t is the number of iteration.

4 Stability region

The absolute stability of the above methods is obtained using a linear first order test problem

$$y' = f = \lambda y \quad (7)$$

The stability region is plotted when the step size ratio is constant, doubled and halved for the method. The test equation (7) is substituted into the corrector formula of the above methods. Setting the determinant of the corrector formula written in matrix form to zero will give the stability polynomial corresponding to each method.

The stability polynomials of 2p1b method at r = 1, 2 and 0.5 are as follow,

For $r = 1$ we have,

$$Q_1(\bar{h}) = t^4\left(1 - \frac{289}{360}\bar{h} + \frac{413}{2160}\bar{h}^2\right) + t^3\left(-1 - \frac{191}{180}\bar{h} - \frac{559}{720}\bar{h}^2\right) + t^2\left(-\frac{49}{360}\bar{h} - \frac{59}{720}\bar{h}^2\right) + \frac{1}{2160}\bar{h}^2 t = 0.$$

For $r = 2$ we have,

$$Q_1(\bar{h}) = t^4\left(1 - \frac{87}{100}\bar{h} + \frac{623}{2700}\bar{h}^2\right) + t^3\left(-1 - \frac{5291}{4800}\bar{h} - \frac{289}{540}\bar{h}^2\right) + t^2\left(-\frac{133}{4800}\bar{h} - \frac{1237}{86400}\bar{h}^2\right) + \frac{1}{86400}\bar{h}^2 t = 0.$$

Finally, for $r = 0.5$ we have,

$$Q_1(\bar{h}) = t^4\left(1 - \frac{147}{200}\bar{h} + \frac{847}{5400}\bar{h}^2\right) + t^3\left(-1 - \frac{407}{600}\bar{h} - \frac{1493}{1080}\bar{h}^2\right) + t^2\left(-\frac{44}{75}\bar{h} - \frac{589}{1350}\bar{h}^2\right) + \frac{8}{675}\bar{h}^2 t = 0.$$

where $\bar{h} = h\lambda$ and the stability region is shown in Fig. 2.

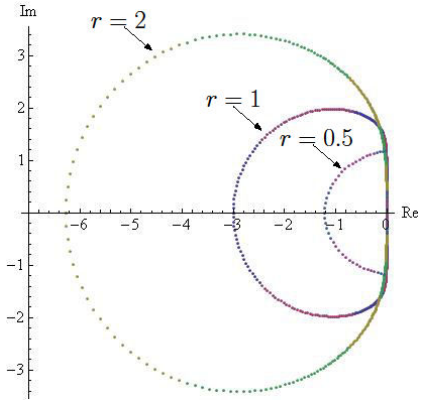

Fig. 2: Stability region for two block method

The stability polynomials of 3p1b method at r = 1, 2 and 0.5 are as follow,

For $r = 1$ we have,

$$Q_1(\bar{h}) = t^6\left(1 - \frac{11947}{10080}\bar{h} + \frac{63859}{120960}\bar{h}^2 - \frac{547411}{6048000}\bar{h}^3\right) + t^5\left(-1 - \frac{659}{315}\bar{h} - \frac{24337}{25200}\bar{h}^2 - \frac{796417}{756000}\bar{h}^3\right) + t^4\left(\frac{559}{2016}\bar{h} + \frac{13063}{403200}\bar{h}^2 + \frac{7879}{63000}\bar{h}^3\right) + t^3\left(\frac{101}{75600}\bar{h}^2 + \frac{593}{756000}\bar{h}^3\right) - \frac{1}{6048000}\bar{h}^3 t^2 = 0$$

For $r = 2$ we have,

$$Q_2(\bar{h}) = t^6\left(1 - \frac{507}{392}\bar{h} + \frac{23795}{37044}\bar{h}^2 - \frac{7024939}{55566000}\bar{h}^3\right) + t^5\left(-1 - \frac{1633693}{94080}\bar{h} - \frac{706621}{604800}\bar{h}^2 - \frac{4281161}{7408800}\bar{h}^3\right) + t^4\left(\frac{3133}{94080}\bar{h} + \frac{455029}{13547520}\bar{h}^2 + \frac{523433}{50803200}\bar{h}^3\right) + t^3\left(\frac{5809}{474163200}\bar{h}^2 - \frac{8848213}{18966528000}\bar{h}^3\right) + \frac{1}{14224896000}\bar{h}^3 t^2 = 0$$

Finally, for $r = 0.5$ we have,

$$Q_1(\bar{h}) = t^6\left(1 - \frac{843}{784}\bar{h} + \frac{79099}{185220}\bar{h}^2 - \frac{3545993}{55566000}\bar{h}^3\right) + t^5\left(-1 - \frac{46723}{11760}\bar{h} + \frac{306023}{264600}\bar{h}^2 - \frac{9633853}{3704400}\bar{h}^3\right) + t^4\left(\frac{3011}{1470}\bar{h} + \frac{3685}{1323}\bar{h}^2 + \frac{252761}{158760}\bar{h}^3\right) + t^3\left(\frac{21248}{231525}\bar{h}^2 + \frac{15412}{231525}\bar{h}^3\right) - \frac{512}{3472875}\bar{h}^3 t^2 = 0$$

where $\bar{h} = h\lambda$ and the stability region is shown in Fig. 3.

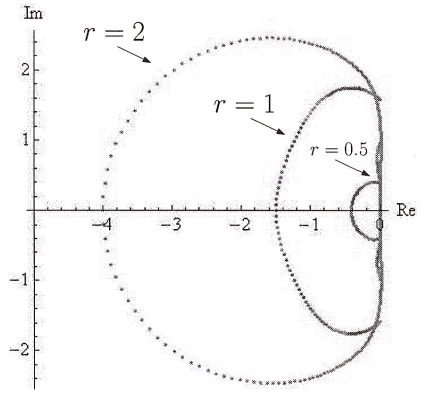

Fig. 3: Stability region for three point block

The stability regions of 2p1b and 3p1b methods at r = 1, 2 and 0.5 are plotted in Fig. 2 and Fig. 3 respectively. The stability region is inside the boundary of the dotted points. In Fig. 2 and 3, the stability region is larger when the step size is half (r = 2) compare to the step size being double (r = 0.5) or constant (r = 1). This is expected since the region should get larger with smaller step size. The stability region for 2p1b is larger compared to the stability region of 3p1b.

4 Parallel techniques using MPI

There are several approaches to parallelizing the proposed sequential programs. In this paper, it is done by distributing the equations among the processors (parallelism across the system). Each processor runs essentially the same program on its share of the equations. Let N be the number of equations in system (1), and P is the number of processors. Therefore, the share of equations for the i^{th} processor is from $(i * L) + (i + 1) * L$, where $L = \frac{N}{P}$.

The parallel algorithm of the proposed 2p1b method is discussed in Table 1 and the same implementation will be applied in 3p1b method. All

the processors have to exchange their required computed values at Step 1 before calling the evaluation function. In the code, we avoid to exchange unnecessary computed values in order to reach a peak speed-up of the parallel code.

Table 1: The parallel algorithm based on the propose method

Step 1	**for** k=1 to 2 **do**
	Processor 0: Prediction $\{y_{n+k,j}\}_{j=1}^{L}$
	Processor 1: Prediction $\{y_{n+k,j}\}_{j=L+1}^{2L}$
	⋮
	Processor P-1: Prediction $\{y_{n+k,j}\}_{j=((P-1)*L)+1}^{P*L}$
	Exchanging of computed values between processors
	Processor 0: Evaluation $\{f_{n+k,j}\}_{j=1}^{L}$
	Processor 1: Evaluation $\{f_{n+k,j}\}_{j=L+1}^{2L}$
	⋮
	Processor P-1: Evaluation $\{f_{n+k,j}\}_{j=((P-1)*L)+1}^{P*L}$
	end for

Step 2	**for** k=1 to 2 **do**
	Processor 0: Correction $\{y_{n+k,j}^{c}\}_{j=1}^{L}$
	Processor 1: Correction $\{y_{n+k,j}^{c}\}_{j=L+1}^{2L}$
	⋮
	Processor P-1: Correction $\{y_{n+k,j}^{c}\}_{j=((P-1)*L)+1}^{P*L}$
	Exchanging of computed values between processors
	Processor 0: Evaluation $\{f_{n+k,j}\}_{j=1}^{L}$
	Processor 1: Evaluation $\{f_{n+k,j}\}_{j=L+1}^{2L}$
	⋮
	Processor P-1: Evaluation $\{f_{n+k,j}\}_{j=((P-1)*L)+1}^{P*L}$
	end for
	Convergent test: **if** yes **then**
	Each processor calculates its own local truncation error (LTE)
	The maximum LTE will be sent to processor 0
	go to Step 3
	else
	go to Step 2
	end if

Step 3	Processor 0 computes the next step size Send this value to all processors in communicator

At Step 2, the procedure is executed until convergence. Each processor does the convergent test on its computed values. All stages at Step 1 and 2 are done simultaneously.

At Step 3, the processor 0 will receive the necessary data from the rest of the processors at the same time to calculate the next step size. By sending this computed value to all processors they will be ready to commence integrating their equations for the next block.

The parallelism is achieved from the beginning of the code. The message passing calls between processors are made through Message Passing Interface (MPI).

4 Numerical results

In order to show the efficiency of the presented method, we present some numerical experiments for two given problems. The following notations are used in the tables:

TOL	Tolerance
MTD	Method employed
TS	Total successful steps
FS	Total failure steps
Maxe	Absolute value of the maximum error of the computed solution
FN	Total function calls
S2p1b	Sequential implementation of 2-point block method using one processor
P2p1b	Parallel implementation of the 2-point block method
S3p1b	Sequential implementation of 3-point block method using one processor
P3p1b	Parallel implementation of the 3-point block method
N	Number of equations in system (1)
P	Number of processors used

Speed-up and efficiency are the measures of relative benefit of parallelizing a given application over sequential implementation. The speed-up

$S_p(n)$ and efficiency $E_p(n)$ on P processors that we used are defined as:

$$S_p(n) = \frac{T_s(n)}{T_p(n)},$$

$$E_p(n) = \frac{S_p(n)}{P}$$

where $T_s(n)$ is equal to execution time of the code on a single processor for a given problem with size n; $T_p(n)$ is the execution time on P processors for solving the problem with size n.

The speed-up shows the speed gain of parallel method developed. In an ideal parallel system, speed-up is equal to the number of processor used and efficiency is equal to 100%. In practice, speed-up is less than P and efficiency is between 0% and 100%.

Problem 1: A radioactive decay chain.

$$\begin{bmatrix} y_1' \\ y_2' \\ \cdot \\ \cdot \\ \cdot \\ y_N' \end{bmatrix} = \begin{bmatrix} -1 & 0 & . & . & . & . & 0 \\ 1 & -1 & 0 & . & . & . & . \\ 0 & 1 & -1 & . & . & . & . \\ . & 0 & . & . & 0 & . & . \\ . & . & 0 & . & -1 & 0 & . \\ . & . & . & . & 1 & -1 & 0 \\ 0 & . & . & . & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{bmatrix}, \; y(0) = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

$N =$ Number of equations, $0 \le x \le b$, $b =$ end of the interval.
Source: [10]

Problem 2: A parabolic partial differential equations. (The Heat Equation)

$$\begin{bmatrix} y_1' \\ y_2' \\ \cdot \\ \cdot \\ \cdot \\ y_N' \end{bmatrix} = \begin{bmatrix} -2 & 1 & 0 & . & . & . & 0 \\ 1 & -2 & 1 & 0 & . & . & . \\ 0 & 1 & -2 & 1 & . & . & . \\ . & 0 & 1 & . & . & 0 & . \\ . & . & & . & -2 & 1 & 0 \\ . & . & . & 0 & 1 & -2 & 1 \\ 0 & . & . & . & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{bmatrix}, \; y(0) = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

$N =$ Number of equations, $0 \le x \le b$, $b =$ end of the interval.
Source: [10]

Problem 1 and 2 are solved without exact reference solution. To measure the accuracy of the codes, the solutions obtained by the methods are compared with the true solutions which are estimated by solving the problems using very small tolerance (10^{-14}). This has been done in separate program. The performance of the code was measured by implementation both sequential and parallel versions in C.

The parallel implementation is supported by Message Passing Interface (MPI) which is a message passing standard library and is widely used to write message passing programs on high performance computing (HPC) platforms. Both sequential and parallel algorithms were carried out on *Sunfire V1280* with eight homogeneous processors located at Institute of Mathematical Research (INSPEM), University Putra Malaysia.

The numerical results in term of total step, failure step, function calls and maximum error are tabulated in Table 2 – 5. The performance of the sequential and parallel execution times for each problem is shown in Table 6 – 9 while Table 10 – 17show the speed-up and efficiency performance for the 2p1b and 3p1b methods.

Table 2: Results of 2p1b solving Problem 1

TOL		TS	FS	FN	Maxe
10^{-2}	2p1b	19	0	133	5.58e-4
10^{-4}	2p1b	33	0	213	2.34e-6
10^{-6}	2p1b	68	0	421	1.91e-8
10^{-8}	2p1b	151	0	909	2.42e-10
10^{-10}	2p1b	356	0	2111	1.97e-12

Table 3: Results of 3p1b for solving Problem 1

TOL		TS	FS	FN	Maxe
10^{-2}	3p1b	20	0	185	2.03e-4
10^{-4}	3p1b	24	0	239	1.97e-6
10^{-6}	3p1b	38	0	356	1.37e-8
10^{-8}	3p1b	60	0	566	1.46e-10
10^{-10}	3p1b	99	0	947	1.34e-12

Table 4: Results of 2p1b for solving Problem 2

TOL		TS	FS	FN	Maxe
10^{-2}	2p1b	17	0	121	3.07e-4
10^{-4}	2p1b	29	0	183	1.13e-6
10^{-6}	2p1b	56	0	349	1.30e-8
10^{-8}	2p1b	123	0	739	1.45e-10
10^{-10}	2p1b	285	0	1693	1.32e-12

Table 5: Results of 3p1b for solving Problem 2

TOL		TS	FS	FN	Maxe
10^{-2}	3p1b	19	0	167	2.07e-4
10^{-4}	3p1b	22	0	218	1.29e-6
10^{-6}	3p1b	32	0	311	7.49e-9
10^{-8}	3p1b	52	0	488	7.02e-11
10^{-10}	3p1b	85	0	806	6.78e-13

The reported results in Table 2 – 3 were obtained when N=6000 and for each different N we will get the same numerical results. The maximum errors for solving the two tested problems are within the given tolerances for both methods.

In Table 6 - 9, the parallel execution times are faster compared to the sequential timing, particularly for large ODEs systems and the number of processors employed for the parallel algorithm. The execution times are larger at smaller tolerance because at this stage the workload increases.

We also could observed that the 3p1b managed to solve faster compared to the 2p1b method. For example in Problem 1, 3p1b and 2p1b need 2.19 and 4.96 seconds respectively to solve 30000 equations at TOL= 10^{-10}.

Table 6: Execution times (seconds) of 2p1b for solving Problem 1 for interval [0, 10]

N	P	MTD	TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	1	S2p1b	0.25	0.37	0.83	1.95	4.68
	2	P2p1b	0.15	0.23	0.54	1.35	3.34
	4	P2p1b	0.07	0.11	0.32	0.86	2.20
	6	P2p1b	0.05	0.08	0.25	0.71	1.85
18000	1	S2p1b	0.91	1.41	2.89	6.44	15.18
	2	P2p1b	0.47	0.71	1.52	3.45	8.29
	4	P2p1b	0.24	0.36	0.80	1.88	4.57
	6	P2p1b	0.17	0.26	0.56	1.39	3.42
30000	1	S2p1b	1.47	2.29	4.62	10.17	23.83
	2	P2p1b	0.74	1.15	2.37	5.30	12.53
	4	P2p1b	0.38	0.58	1.23	2.83	6.79
	6	P2p1b	0.27	0.41	0.87	2.04	4.95

Table 7: Execution times (seconds) of 3p1b for solving Problem 1 for interval [0, 10]

N	P	MTD	TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	1	S3p1b	0.37	0.47	0.75	1.25	2.17
	2	P3p1b	0.23	0.27	0.47	1.82	1.48
	4	P3p1b	0.11	0.13	0.27	0.51	0.97
	6	P3p1b	0.08	0.09	0.20	0.41	0.80
18000	1	S3p1b	1.37	1.67	2.53	4.03	6.80
	2	P3p1b	0.71	0.86	1.34	2.21	3.73
	4	P3p1b	0.36	0.42	0.68	1.16	2.04
	6	P3p1b	0.24	0.29	0.48	0.84	1.52
30000	1	S3p1b	2.31	2.82	4.21	6.66	11.13
	2	P3p1b	1.16	1.41	2.12	3.35	5.66
	4	P3p1b	0.59	0.71	1.07	1.76	3.03
	6	P3p1b	0.40	0.48	0.74	1.24	2.19

Table 8: Execution times (seconds) of 2p1b for solving Problem 2 for interval [0, 5]

N	P	MTD	TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	1	S2p1b	0.22	0.33	0.69	1.60	3.81
	2	P2p1b	0.14	0.21	0.46	1.13	2.75
	4	P2p1b	0.07	0.10	0.27	0.72	1.81
	6	P2p1b	0.05	0.07	0.22	0.60	1.54
18000	1	S2p1b	0.85	1.25	2.47	5.38	12.50
	2	P2p1b	0.44	0.64	1.26	2.78	6.48
	4	P2p1b	0.23	0.32	0.68	1.58	3.79
	6	P2p1b	0.16	0.23	0.49	1.18	2.86
30000	1	S2p1b	1.38	2.05	3.95	8.52	19.68
	2	P2p1b	0.70	1.03	2.00	4.39	10.25
	4	P2p1b	0.37	0.52	1.05	2.38	5.59
	6	P2p1b	0.25	0.36	0.74	1.71	4.09

Table 9: Execution times (seconds) of 3p1b for solving Problem 2 for interval [0, 5]

N	P	MTD	TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	1	S2p1b	0.35	0.44	0.67	1.10	1.88
	2	P2p1b	0.21	0.26	0.42	0.73	1.29
	4	P2p1b	0.11	0.13	0.24	0.45	0.83
	6	P2p1b	0.08	0.10	0.19	0.37	0.70
18000	1	S2p1b	1.28	1.58	2.27	3.58	5.94
	2	P2p1b	0.65	0.80	1.16	1.88	3.15
	4	P2p1b	0.33	0.40	0.62	1.03	1.78
	6	P2p1b	0.24	0.29	0.44	0.77	1.34
30000	1	S2p1b	2.14	2.66	3.78	5.93	9.75
	2	P2p1b	1.08	1.34	1.90	3.00	4.95
	4	P2p1b	0.54	0.67	0.97	1.57	2.65
	6	P2p1b	0.38	0.45	0.68	1.12	1.92

Table 10: Speed-up of 2p1b for solving Problem 1

N	P	TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	1.66	1.60	1.54	1.44	1.40
	4	3.51	3.36	2.59	2.27	2.13
	6	5.00	4.62	3.32	2.75	2.53
18000	2	1.93	1.99	1.90	1.87	1.83
	4	3.79	3.92	3.61	3.43	3.32
	6	5.35	5.42	5.16	4.63	4.44
30000	2	1.99	1.99	1.94	1.92	1.90
	4	3.87	3.95	3.76	3.59	3.51
	6	5.44	5.59	5.31	4.99	4.81

It is noted in Table 10 – 15 that the speed-up and efficiency by using two, four and six processors have improved as the number of equations increased.

Fig. 4 – 5 shown that the speed-up improved as the number of equations increased.

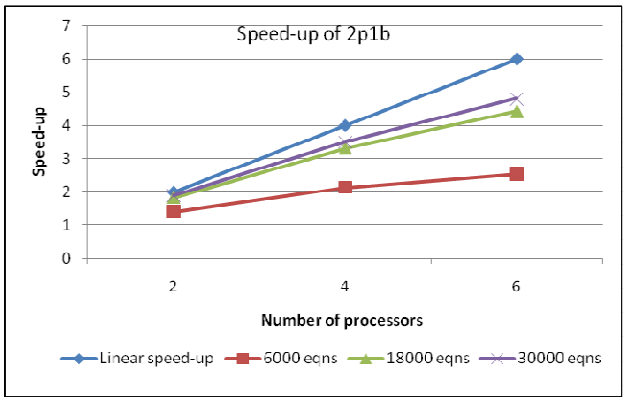

Fig. 4 Speed-up of 2p1b at TOL=10^{-10} for solving Problem 1

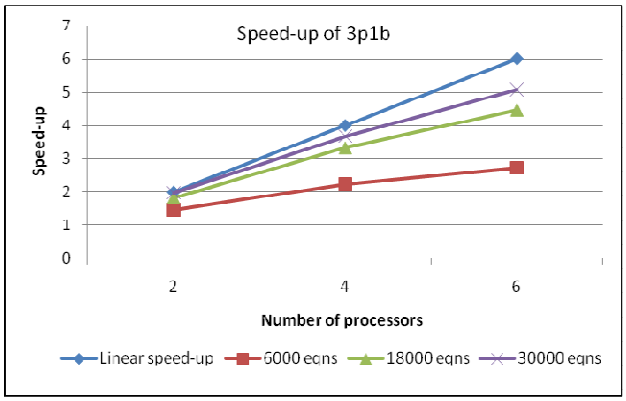

Fig. 4 Speed-up of 3p1b at TOL=10^{-10} for solving Problem 1

Table 11: Speed-up of 3p1b for solving Problem 1

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	1.60	1.74	1.59	1.52	1.46
	4	3.36	3.61	2.77	2.45	2.23
	6	4.62	5.22	3.75	3.04	2.71
18000	2	1.92	1.94	1.88	1.82	1.82
	4	3.80	3.97	3.72	3.47	3.33
	6	5.70	5.75	5.27	4.79	4.47
30000	2	1.99	1.98	1.98	1.98	1.96
	4	3.91	3.97	3.93	3.78	3.67
	6	5.77	5.87	5.68	5.37	5.08

In general, the speed-up varying for 2p1b from 1.39 to 1.99 when using two processors, 2.10 to 3.95 when using four processors and while using six processors the speed-up varying from 2.47 to 5.69 for solving Problem 1 and 2. For example, in solving Problem 1 the maximum speed-up and efficiency gained by the parallel performance of the P2p1b using 6 processors is 2.75 [46%], 4.63 [77%] and 4.99 [83%] respectively at TOL $=10^{-8}$ when N=6000, 18000 and 30000.

Table 12: Speed-up of 2p1b for solving Problem 2

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	1.57	1.57	1.50	1.41	1.39
	4	3.14	3.30	2.56	2.22	2.10
	6	4.44	4.71	3.14	2.67	2.47
18000	2	1.93	1.95	1.96	1.94	1.93
	4	3.70	3.90	3.63	3.41	3.30
	6	5.31	5.43	5.04	4.56	4.37
30000	2	1.97	1.99	1.98	1.94	1.92
	4	3.73	3.94	3.76	3.58	3.52
	6	5.52	5.69	5.34	4.98	4.81

Table 13: Speed-up of 3p1b for solving Problem 2

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	1.66	1.69	1.59	1.50	1.45
	4	3.18	3.38	2.79	2.44	2.26
	6	4.37	4.40	3.52	2.97	2.68
18000	2	1.96	1.97	1.95	1.90	1.88
	4	3.87	3.95	3.66	3.47	3.33
	6	5.33	5.44	5.15	4.64	4.43
30000	2	1.98	1.98	1.98	1.97	1.96
	4	3.96	3.97	3.89	3.77	3.67
	6	5.63	5.91	5.55	5.29	5.07

In 3p1b, the speed-up varying from 1.45 to 1.98 when using two processors, 2.23 to 3.97 when using four processors and while using six processors the speed-up varying from 2.47 to 5.91 for solving Problem 1 and 2.

Better speed-up and efficiency can be achieved by increasing the dimensions of ODEs.

Table 14: Efficiency (%) of 2p1b for solving Problem 1

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	94	80	77	72	70
	4	100	84	65	57	53
	6	93	77	55	46	42
18000	2	97	99	95	94	92
	4	95	98	90	86	83
	6	89	90	86	77	74
30000	2	99	99	97	96	95
	4	97	99	94	90	88
	6	91	93	89	83	80

Table 15: Efficiency (%) of 3p1b for solving Problem 1

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	80	87	79	76	73
	4	84	90	69	61	55
	6	77	87	62	50	45
18000	2	96	97	94	91	91
	4	95	99	93	86	83
	6	95	96	87	79	74
30000	2	99	100	99	99	98
	4	97	99	98	94	91
	6	96	98	94	89	84

Table 16: Efficiency (%) of 2p1b for solving Problem 2

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	79	79	75	71	70
	4	79	83	64	56	53
	6	74	79	52	45	41
18000	2	97	98	98	97	97
	4	93	98	91	85	83
	6	89	91	84	76	73
30000	2	99	99	99	97	96
	4	94	99	94	90	88
	6	92	95	89	83	80

Table 17: Efficiency (%) of 3p1b for solving Problem 2

N	P	TOL				
		10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
6000	2	83	84	79	75	72
	4	79	84	69	61	56
	6	72	73	58	49	44
18000	2	98	98	97	95	94
	4	96	98	91	86	83
	6	88	90	85	77	73
30000	2	99	99	99	98	98
	4	99	99	97	94	91
	6	93	98	92	88	84

In Table 14 – 17, the results show that the performance of the parallel system improved when solving large systems of ODEs.

We also could observe that as the number of processors increased for solving the same number of equations, the efficiency has slightly decreased. This is expected since the communication among the processors has increased and this has affected the parallel performance. The parallel performance improved as the workload increases.

5 Conclusion

In this paper we have proposed a parallel algorithm for solving large system of ODEs based on 2-point and 3-point block method. The code has shown the superiority of the parallelism across the system when using large-scale problems. The numerical results show that the speed-up improves as the problem size increases.

References:

[1] J.B. Rosser, A Runge-Kutta for all seasons,*SIAM Rev*. vol , 1967, pp. 417-452.

[2] K. Burrage, Efficient Block Predictor-Corrector Methods with a Small Number of Corrections, *J. of Comp. and App. Math*, vol 45, 1993, pp. 139-150.

[3] K. Burrage and H. Suhartanto, Parallel iterated methods based on multistep Runge-kutta methods of Radau type, *Adv. Comput. Math*, vol 7, 1997, pp. 37-57.

[4] K. Burrage and H. Suhartanto, Parallel iterated methods based on variable step-size multistep Runge-kutta methods of Radau type for stiff problems, *Adv. Comput. Math*, vol 13, 2000, pp. 257-270.

[5] N. H. Cong, K. Strehmel, R. Weiner and H. Podhaisky,Runge-Kutta Nystrom-type block predictor corrector methods, *Adv. In Comput. Math*, vol 10, 1999, pp. 115–133.

[6] P.J. Houwen, P.B. Sommeijer, Block Runge-Kutta methods on parallel computers, Report NM-R8906, Center for Mathematics and Computer Science, Amsterdam, 1989.

[7] Q. Feng. And B. Zheng, Parallel alternating group explicit iterative method for convection-diffusion equation, Proceeding of the 8th WSEAS International Conference on a Applied Computer and Applied Computational Sciences, 2009, pp 383-387.

[8] Q. Feng. And B. Zheng, A parallel finite difference for fourth order parabolic equation Proceeding of the 8th WSEAS International Conference on a Applied Computer and Applied Computational Sciences, 2009, pp 411-387.pp 411-416.

[9] Q. Feng. And B. Zheng, A class of parallel difference method for solving convection-diffusion equation with variable coefficient. Proceeding of the 8th WSEAS International Conference on a Applied Computer and Applied Computational Sciences, 2009, pp 379-382

[10] T.E. Hull, W.E. Enright, B.M. Fellel and A.E. Sedgwick, Comparing numerical methods for

ordinary differential equations, *SIAM J. Num. Anal,* vol 9, no. 4, 1972, pp. 603-637.

[11] U.K.S Din, F.Ismail,M.Suleiman, M. Othman and Z.A.Majid. The parallel three-processor fifth order diagonally implicit Runge-Kutta methods for solving ordinary differential equations. Proceeding of the *The 12th WSEAS International Conference on Applied Mathematics,* 2007, pp 184-188.

[12] W.L. Miranker and W.M. Liniger, Parallel methods for the numerical integration of ordinary differential equations, *Math. Comp*, vol 21, no. 99, 1967, pp. 303-320.

[13] Z.A. Majid and M.B. Suleiman, Implementation of parallel three-point block codes for solving large system of ordinary differential equations, *International Journal of Computer Mathematics*. Vol 87, no. 6, 2010, pp. 1-15.87:6, 1415 – 1429.

[14] Z. Omar, Developing parallel block methods for solving higher order ODEs directly, Ph.D. dissertation, University Putra Malaysia, Malaysia, 1999.