Menemui Matematik (Discovering Mathematics)
Vol. 33, No. 1: 23 – 36 (2011)

Parallel Two-Processor Fifth Order Diagonally Implicit Runge-Kutta Method

Ummul Khair Salma Din[1], Fudziah Ismail[2]
[1]*School of Mathematical Sciences,*
Faculty of Science and Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor
[2]*Department of Mathematics, Faculty of Science,*
Universiti Putra Malaysia, 43400 UPM Serdang, Selangor
e-mail: [1] *ummul@.ukm.my* , [2]*fudziah_i@yahoo.com.my*

ABSTRACT

The reduction of time on parallel computation has spurred interests on the development of numerical methods with parallel capability. This paper presents a derivation of a fifth order Runge-Kutta method which is suitable for parallel implementation. The sparsity structure in the development of fourth order parallel Runge-Kutta methods has inspired the construction of this method. Numerical results in terms of accuracy are compared to three established fourth order parallel Runge-Kutta method. The results based on sequential computations suggest that the new method is more accurate compared to the three methods. On the parallel performance of the method an algorithm was developed using C language and the parallel computation was done in Message Passing Interface (MPI) environment. The execution times of the method by sequential and parallel implementation on two large problems are presented. Results show that time taken for the parallel computation gave significant reduction compared to the sequential implementation.

Keywords: Fifth order Runge-Kutta method, initial value problem, parallel computation, two processors

INTRODUCTION

Runge-Kutta method is a famous one-step method for solving initial value problems (IVP). Developed from Euler's Rule, Runge-Kutta methods are able to achieve higher order without sacrificing the one-step form. The classical fourth order Runge-Kutta method often emerged as one of numerical methods introduced mainly for students studying applied mathematics and engineering. Traditionally, Runge-Kutta methods are all explicit however Butcher (1987) has listed six basic reasons for taking a serious interest in implicit Runge-Kutta methods. The main reason is because of the higher orders of accuracy can be obtained than for explicit methods. Due to the excessive cost in evaluating the stages in a fully implicit Runge-Kutta method, many researchers have opted for the diagonally implicit Runge-Kutta (DIRK) method, as called by Alexander (1977). For these methods, the coefficient matrix A has a lower triangular structure with equal elements on the diagonal. Sometimes these methods are referred to as singly diagonally implicit Runge-Kutta (SDIRK), with DIRK methods not necessarily having equal diagonals.

Latest development has shown that attentions are now given to parallel implementation in numerical computation. According to Jackson (1991), the desire for parallel IVP solvers arises from

the need to solve many important problems more rapidly than is currently possible. The reduction in cost particularly time, is undeniably give great motivation in developing this idea.

Iserles and Nørsett's (1990) idea in presenting parallel Runge-Kutta methods through sparsity structure is considered as one of the best parallel designs for Runge-Kutta methods. The structure allows the evaluation of the functions with independent arguments to be computed on different processors at the same time as one single evaluation on one processor. Previous methods using the sparsity structure are fourth order DIRK methods suitable to be implemented on two processors (Nørsett & Simonsen (1987), Jackson (1991), van der Houwen *et al.* (1992), Jackson & Nørsett (1995) and Burrage (1995)).

In this paper, we present a fifth order Runge-Kutta method (which later will be referred as P2DIRK5), where a pattern of DIRK is developed to permit the implementation of parallel computation. Comparison of numerical results based on sequential computations to established parallel fourth order Runge-Kutta methods will be discussed. The performance of P2DIRK5 particularly on the cost of computation time in solving two large systems of ordinary differential equations (ODEs) will be presented.

THE PARALLELISM OF RUNGE-KUTTA METHODS

The IVP for a system of first order ordinary differential equations (ODEs) is defined by

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

The general *s*-stage Runge-Kutta method for problem as in Eq. is defined by

where

$$\left.\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i \\ k_i &= f\left(x_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_j\right), \qquad i = 1,2,...,s, \end{aligned}\right\} \tag{2}$$

assuming the following holds:

$$c_i = \sum_{j=1}^{s} a_{ij}, \qquad i = 1,2,...,s. \tag{3}$$

The coefficients in Eqs. (2) can also be displayed by using *Butcher' array or Butcher's tableau* (Butcher, 1987) as shown in Figure 1

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

Figure 1. Butcher' array or Butcher's tableau

The *s*-dimensional vectors *c* and *b* and the *sxs* matrix *A* in Figure 1 are $[c_1,c_2,c_3,\ldots,c_s]^T$, $[b_1,b_2,b_3,\ldots,b_s]^T$ and $[a_{ij}]$, respectively.

Iserles and Nørsett (1990) have explored the parallelism in Runge-Kutta methods by analyzing directed graph for matrix of sparsity structure as shown in Figure 2 (also in Nørsett and Simonsen, 1987).

c_1	a_{11}			
c_2	0	a_{22}		
c_3	a_{31}	a_{32}	a_{33}	
c_4	a_{41}	a_{42}	0	a_{44}
	b_1	b_2	b_3	b_4

Figure 2. Sparsity structure for fourth order Runge-Kutta methods.

In Runge-Kutta formulation it is written as,

$$y_{n+1} = y_n + h\sum_{i=1}^{4} b_i k_i$$

where

$$\left.\begin{aligned} k_1 &= f(x_n + c_1 h, y_n + h(a_{11}k_1),\\ k_2 &= f(x_n + c_2 h, y_n + h(a_{22}k_2),\\ k_3 &= f(x_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2 + a_{33}k_3),\\ k_4 &= f(x_n + c_4 h, y_n + h(a_{41}k_1 + a_{42}k_2 + a_{44}k_4), \end{aligned}\right\} \quad (4)$$

which is a fourth order diagonally implicit Runge-Kutta method. Clearly from the method, does not depend on and does not depend on k_3. A digraph is used to model the sparsity pattern of the matrix as shown in Figure 3.

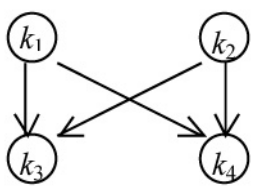


Figure 3. Digraph of sparsity structure for fourth order Runge-Kutta methods.

Methods of this kind are:

1	1	0	0	0
$\frac{3}{5}$	0	$\frac{3}{5}$	0	0
0	$\frac{171}{44}$	$-\frac{215}{44}$	1	0
$\frac{2}{5}$	$-\frac{43}{20}$	$\frac{39}{20}$	0	$\frac{3}{5}$
	$\frac{11}{72}$	$\frac{25}{72}$	$\frac{11}{72}$	$\frac{25}{72}$

Figure 4. Method by Jackson and Norsett (DIRK1)

$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{2}{3}$	0	$\frac{2}{3}$	0	0
$\frac{1}{2}$	$-\frac{5}{2}$	$\frac{5}{2}$	$\frac{1}{2}$	0
$\frac{1}{3}$	$-\frac{5}{3}$	$\frac{4}{3}$	0	$\frac{2}{3}$
	-1	$\frac{3}{2}$	-1	$\frac{3}{2}$

Figure 5. Method by Iserles and Norsett (DIRK2)

$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
1	0	1	0	0
$\frac{1}{2}$	$\frac{3}{2}$	$-\frac{3}{2}$	$\frac{1}{2}$	0
0	-3	2	0	1
	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$

Figure 6. Method by Iserles and Norsett (DIRK3).

Figure 4 is a method by Jackson and Norsett (1995) and through the discussion it will be referred as DIRK1. Meanwhile Figures 5 and 6 which are methods by Iserles and Norsett (1990) will be referred as DIRK2 and DIRK3. These three methods are suitable for parallel implementation using two processors.

We extend the method to a higher order Runge-Kutta method which is a fifth order with six stages having the same sparsity structure (see Figure 7).

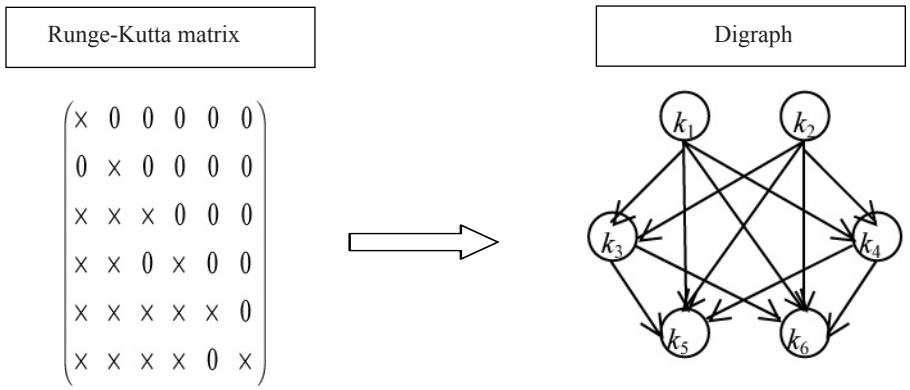


Figure 7. Sparsity structure and digraph for fifth order Runge-Kutta methods with six stages.

The derivation of the method can be found in Din *et al.* (2006). Following is the coefficients obtained for the method.

$\gamma = 0.5111360444707436,$ $\alpha_{65} = 0.0$

$\beta = 0.1200913622711499,$ $a_{66} = \beta,$

$a_{22} = \beta,$ $c_1 = \gamma,$

$a_{32} = 0.3304320730047836,$ $c_2 = \beta,$

$a_{33} = \gamma,$ $c_3 = 0.5083725832456476,$

$a_{42} = -0.001764317609504879,$ $c_4 = 0.1048607614397692,$

$a_{43} = 0.0,$ $c_5 = 1-\gamma,$

$a_{44} = \beta,$ $c_6 = 1-\beta,$

$a_{52} = 0.09184637904240046,$ $b_1 = 0.0,$

$a_{53} = -0.2402905535606616,$ $b_2 = 0.0,$

$a_{54} = 0.206131006694386,$ $b_3 = -0.179679897773612,$

$a_{55} = \gamma,$ $b_4 = 0.2614074344151084,$

$a_{62} = 0.008530162133924786,$ $b_5 = 0.6241305207277307,$

$a_{63} = 0.5217820185104087,$ $b_6 = 0.2941419426307729.$

$a_{64} = 0.251411354949371,$

The values of α_{i1} for $i = 1, 2,\ldots, 6$ are obtained by following the relationship,

$$a_{i1} = c_i - \sum_{j=2}^{i} a_{ij}$$

STABILITY OF THE METHOD

The stability function or stability polynomial $R(\hat{h})$ as given in Lambert (1991)

$$R(\hat{h}) = 1 + \hat{h}b^T(I - \hat{h}A)^{-1}e \tag{5}$$

is obtained when the method is applied to the linear test equation

$$y' = \lambda y,\ \lambda \in C,\ \mathrm{Re}(\lambda) < 0 \tag{6}$$

$\hat{h}$ in Eq. (5) represents $h\gamma$ and $e = [1, 1, ...1]^T$ An alternative expression for Eq. is given in the following lemma. The proof can be seen in Butcher (2003).

Lemma: *Let (A,b,c) denote a Runge-Kutta method. Then its stability function is given by*

$$R(\hat{h}) = \frac{\det(I + \hat{h}(eb^T - A)}{\det(I - \hat{h}A)}.$$

Using Mathematica, we obtained the stability polynomial for the method as

$$R(\hat{h}) = \frac{0.890628(\hat{h} - 19.1201)(\hat{h} - 8.52181)(\hat{h}^2 - 3.11507\hat{h} + 2.53164)(\hat{h}^2 + 5.96002\hat{h} + 11.7689)}{(\hat{h} - 8.32693)(\hat{h} - 1.95644)(\hat{h}^2 - 16.6541\hat{h} + 69.3394)(\hat{h}^2 - 3.91283\hat{h} + 3.82757)},$$

while the stability region where the region lies outside the closed boundary is shown in Figure 8. P2DIRK5 is said to has an A(α)-stability with $\alpha \approx 88.58^{\circ}$.

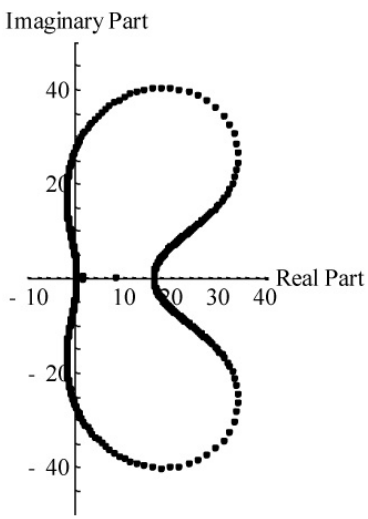


Figure 8. Stability region for PDIRK5

THE PERFORMANCE OF THE PARALLEL PROCESSING

In comparing the performance between a multiprocessor system and a single processor system we shall use the *speedup* factor and *efficiency* denoted by S_p and E_p respectively. S_p is defined as,

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

where t_p is the execution time of the best sequential algorithm running on a single processor while t_p is the execution time for solving the same problem on a multiprocessor. The maximum speedup possible is usually p with processors p, normally referred to as *linear speedup*. However several factors will appear as overhead in the parallel version and limit the speedup making it rare in achieving the linear result. Among common overheads are (Wilkinson & Allen, 2005):

1. Periods when not all processors can be performing useful work and are simply idle.
2. Extra computations in the parallel version not appearing in the sequential version.
3. Communication time between processes.

.

. Due to these factors the speedup is often less than p.

The efficiency E_p is defined as,

$$E_p = \frac{S_p}{p}$$

which is the ratio of speedup to the number of processors. It is used to measure the processor utilization. If given as a percentage, the efficiency of 100% happens when all the processors are being used on the computation at all times and S_p equal to p.

The algorithm for P2DIRK5 is executed in C language. For the parallel implementation it is supported by Message Passing Interface (MPI) which is a message passing library standard and is extensively used to write message passing programs on high performance computing (HPC) platforms. Both sequential and parallel algorithms were run on Sunfire V1280 with eight homogenous processors located at Institute of Mathematical Research (INSPEM), Universiti Putra Malaysia, Serdang.

NUMERICAL EXPERIMENTS

Accuracy of the Method

All problems that were tested are non-stiff problems. The codes for the algorithm are written and compiled in Microsoft Visual C++. Table 1 gives the performance comparison between P2DIRK5, DIRK1, DIRK2 and DIRK3 in term of maximum error. The step sizes used are 0.01, 0.001, 0.0001 and 0.00001. The maximum error is defined as

$$\max_{1 \le i \le steps} \left(\left| y_i - y(x_i) \right| \right)$$

where y_i is the computed value and $y(x_i)$ is the true solution of the problems.

A. Tested problems:

Problem S1:

$$y' = \frac{1}{2}(x - y)$$

$$y(0) = 1,\ 0 \le x \le 20$$

Exact solution: $y(x) = e^{-x}$
Source: Artificial problem

Problem S2:

$$y' = \frac{1}{2}(x - y)$$

$$y(0) = 1,\ 0 \le x \le 20.$$

Exact solution: $y(x) = -2 + x + 3e^{-\frac{1}{2}x}.$

Source: Mathews (1992)

Problem S3:

$$y_1' = -y_1 - \sqrt{3}\,y_2,\ y_2 = \sqrt{3}\,y_1 - y_2$$
$$y_1(0) = 1,\ y_2(0) = 0,\ 0 \le x \le 20.$$

Exact solutions: $y_1(x) = e^{-x}\cos\sqrt{3}\,x,\ y_2(x) = e^{-x}\sin\sqrt{3}\,x.$

Source: Tam (1992)

Problem S4:

$$y_1 = -y_1 + y_2,\ y_2 = y_1 - 2y_2 + y_3,\ y_3 = y_2 - y_3$$
$$y_1(0) = 2,\ y_2(0) = 0,\ y_3(0) = 1,\ 0 \le x \le 20.$$

Exact solutions: $y_1(x) = \frac{1}{2}e^{-3x} + 1 + \frac{1}{2}e^{-x},$

$$y_2(x) = 1 - e^{-3x},\ y_3(x) = 1 + \frac{1}{2}e^{-3x} - \frac{1}{2}e^{-x}$$

Source: Shampine (1980)

Problem S5:

$$y_1 = y_2,\ y_2 = -y_3,\ y_3 = y_4,\ y_4 = y_2 + 2e^x$$
$$y_1(0) = 0,\ y_2(0) = -2,\ y_3(0) = 0,\ y_4(0) = -2,\ 0 \le x \le 10.$$

Exact solutions: $y_1(x) = -e^x + e^{-x},\ y_2(x) = -e^x - e^{-x},$

$$y_3(x) = e^x - e^{-x},\ y_4(x) = e^x + e^{-x}.$$

Source: Bronson (1973)

B. Results

Table 1. Numerical Results for Test Problems S1 - S5 using DIRK1, DIRK2, DIRK3 and P2DIRK5

Problem	Method	h=0.01	h=0.001	h=0.0001	h=0.00001
1	DIRK1	7.3605888(-08)	7.3612702(-08)	7.3579575(-08)	7.3576266(-08)
	DIRK2	6.4177323(-09)	1.1036538(-10)	1.1032453(-11)	1.1093904(-12)
	DIRK3	7.3685400(-08)	7.3612690(-08)	7.3579575(-08)	7.3576266(-08)
	P2DIRK5	6.0281913(-10)	6.0285110(-14)	4.2743586(-15)	7.3274720(-15)
2	DIRK1	4.0070930(-07)	3.9980949(-07)	3.9971949(-07)	3.9970999(-07)
	DIRK2	1.4971420(-09)	1.4979307(-10)	1.4992452(-11)	1.8207658(-12)
	DIRK3	4.0070932(-07))	3.9980949(-07)	3.9971948 (-07)	3.9970999(-07)
	P2DIRK5	3.7505332(-11)	7.1054274(-14)	1.1368684(-13)	6.2527761(-13)
3	DIRK1	1.5551270(-07)	1.3570843(-07)	1.3742278(-07)	1.3741428(-07)
	DIRK2	8.8400030(-08)	4.4003146(-10)	4.3480053(-11)	4.3480497(-12)
	DIRK3	1.5264842(-07)	1.3750830(-07)	1.3742278(-07)	1.3741428(-07)
	P2DIRK5	8.3484779(-09)	8.3568894(-13)	2.1371793(-15)	5.7731597(-15)
4	DIRK1	3.6288467(-07)	7.3683759(-08)	7.3586936(-08)	7.3577007(-08)
	DIRK2	5.1755977(-07)	3.3248782(-10)	3.3109404(-11)	3.3071323(-12)
	DIRK3	3.3138603(-07)	7.3684274(-08)	7.3586934(-08)	7.3577007(-08)
	P2DIRK5	4.8777528(-08)	4.8845324(-12)	2.0250468(-13)	1.7443824(-12)
5	DIRK1	1.0926560(-02)	1.1000071(-02)	1.1003057(-02)	1.1003246(-02)
	DIRK2	1.3619845(-04)	1.3223398(-05)	1.3228455(-06)	1.3214958(-07)
	DIRK3	1.0929159(-02)	1.1000069(-02)	1.1003057(-02)	1.1003246(-02)
	P2DIRK5	1.9870018(-05)	1.7280399(-09)	1.7826096(-10)	4.1836756(-10)

Execution time

Two large problems were tested using P2DIRK5 to compare the sequential and parallel time taken to solve the problems for N=5000, 10000, 15000, 20000 and 30000 for step sizes 0.01, 0.001, 0.0001 and 0.00001. Table 2 shows the total steps for every step size considered.

Table 2. Step size and its total steps

Step size	Total steps
0.01	100
0.001	1000
0.0001	10000
0.00001	100000

A. Tested problems

Problem L1 (A Parabolic Partial Differential Equations):

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

N = Number of equations

$0 \leq x \leq 1$

Source: Hull *et al*. (1972)

Problem L2 (Lagrange Equation for the hanging string):

$$\begin{aligned} y_1 &= y_2 \\ y_2 &= -y_1 + 2y_3 \\ y_3 &= y_4 \\ y_4 &= y_1 - 5y_3 + 4y_5 \\ &\vdots \\ y_{N-1} &= y_N \\ y_N &= (N-3)y_{N-3} - (2N-3)y_{N-1} \end{aligned}$$

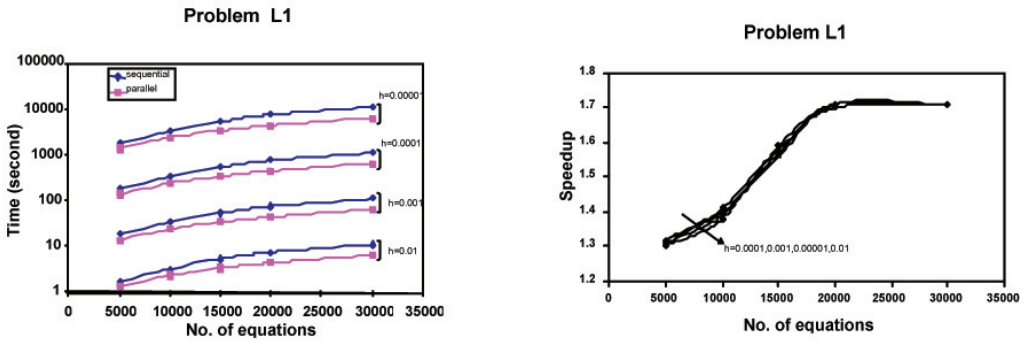
N = Number of equations

$0 \leq x \leq 1$

Initial values: $y_i(0) = 0$ except $y_{N-2}(0) = 1$
Source: Majid (2004)

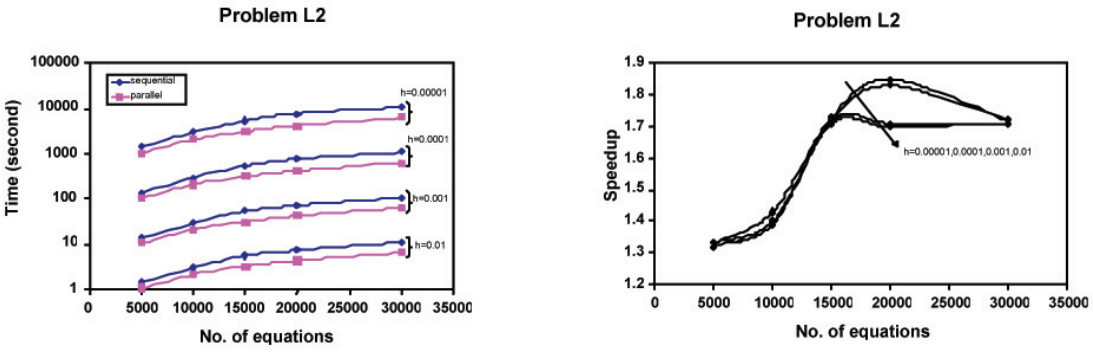
B. Results

The performance of the sequential and parallel execution times for every problem is shown in Figure 9(a) and 10(a) respectively while Figure 9(b) and 10(b) show the speedup performance for the problems.



(a) Time taken for sequential and parallel executions (b) Speedup

Figure 9. Results for Problem L1



(a) Time taken for sequential and parallel executions (b) Speedup

Figure 10. Results for Problem L2

The efficiency of the method is given in Table 3.

Table 3. The efficiency of P2DIRK5

No. of equations	Problem	Efficiency (%)			
		h=0.01	h=0.001	h=0.0001	h=0.00001
5000	L1	65.13	65.77	65.81	65.71
	L2	65.81	65.99	66.29	66.29
10000	L1	69.22	69.81	70.4	69.66
	L2	69.35	69.61	70.24	71.5
15000	L1	79.53	78.11	78.96	78.27
	L2	85.49	85.79	85.96	86.29
20000	L1	85.51	85.32	85.06	85.37
	L2	85.07	85.44	**91.53**	**92.42**
30000	L1	85.62	85.55	85.27	85.73
	L2	85.32	85.51	85.98	85.83

DISCUSSION AND CONCLUSION

From the results in Table 1,it is found that among the three established methods, DIRK2 shows the most accurate result. An interesting finding can be seen for DIRK1 and DIRK3 where they reach to a same performance when we used the step sizes 0.0001 and 0.00001. For all methods, the smaller the step size used, the results became more accurate. Finally, it is clearly shown that PDIRK5 is the best method among the four methods in term of accuracy.

As for the execution time, Figure 9(a) and 10(a), showed that for every step sizes the parallel execution times are better than the sequential execution times. The time reduction is more apparent as the number of equations increased and the results are consistent for every step size. In Figure 9(b) and 10(b), the results show that as the number of equations gets larger, the speedup increased. In Problem L2, for step sizes 0.0001 and 0.00001 the speedup have reached up to 1.83 and 1.85 respectively, approaching to the ideal speedup for two processors execution. The processors utilization for that particular execution are 91.53% and 92.42% respectively, as shown in Table 3. We can say that during the execution, the processors are almost fully utilized. We can also observed in Table 3 that as the problems get larger, the processors utilization improved.

P2DIRK5 is a newly derived fifth order DIRK method with parallel capability. The parallel structure is based on the sparsity structure introduced by Iserles and Nørsett (1990) which has been used in deriving efficient fourth order DIRK methods. Din et. al (2006), have shown that P2DIRK5. In this paper, we showed that P2DIRK has better accuracy than the fourth order methods of its type and the ability to reduce the computation time by the parallel implementation on two processors. This contribution is significant when large problems come in hands because time reduction means cheaper cost in the computation processes. This makes P2DIRK5 as an efficient method for solving ODEs.

REFERENCES

Alexander, R. (1977). Diagonally implicit Runge-Kutta methods for stiff ODE's. *SIAM Journal on Numerical Analysis,* **14**(6):1006-1021.

Bronson, R. (1973). *Schaum's Outline of Modern Introductory Differential Equations*: Mc Graw-Hill.

Burrage,K. (1995). *Parallel and sequential methods for ordinary differential equations*: Oxford University Press Inc.,New York.

Butcher, J. C. (1987). *The numerical analysis of ordinary differential equations: Runge-Kutta and General Linear methods*: John Wiley & Sons.

Butcher, J. C. (2003). *Numerical Methods for Ordinary Differential Equations*: John Wiley & Sons.

Din, U.K.S., Ismail, F., Suleiman, M. and Ahmad, S. (2006). Fifth order Runge-Kutta method for parallel execution. *Prosiding Seminar Kebangsaan Sains Kuantitatif 2006*, UUM: 1-10.

Hull, T.E., Enright, W.H., Fellen, B.M. and Sedgwick, A.E. (1972). Comparing numerical methods for ordinary differential equations. *SIAM J. Num. Anal* 9(4): 603-637.

Iserles, A., and Nørsett, S.P. (1990). On the theory of parallel Runge-Kutta methods. *IMA Journal of Numerical Analysis, 10*, 463-488.

Jackson, K. R. (1991). A survey of parallel numerical methods for initial value problems for ordinary differential equations. *IEEE Transactions on Magnetic, 27*(5), 3792-3797.

Jackson, K. R., and Nørsett, S.P. (1995). The potential for parallelism in Runge-Kutta methods. Part 1: RK formulas in standard form. *Siam J. Numer. Anal., 32*(1), 49-82.

Lambert,J.D.(1991). *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*: John Wiley & Sons.

Majid, Z.A. (2004). Parallel block methods for solving ordinary differential equations. Ph.D. Thesis. Universiti Putra Malaysia.

Mathews, J. H. (1992). *Numerical Methods for Mathematics, Science and Engineering*: Prentice Hall.

Nørsett, S. P., and Simonsen,H.H. (1987). Aspects of parallel Runge-Kutta methods, in *Numerical methods for ordinary differential equations*, Bellen,A.,Gear, C.W. and Russo, E..eds.,Lecture Notes in Mathematics:103-107. Springer-Verlag,Berlin.

Shampine L.F. (1980). What everyone solving differential equations numerically should know. *Computational Techniques for Ordinary Differential Equations*, ed. Gladwell, I. and Sayers,D.K.: Academic Press.

Tam, H. W. (1992). Two-stage parallel methods for the numerical solution of ordinary differential equations. *Siam J. Sci. Stat. Comput.,* **13**(5):1062-1084.

van der Houwen,P.J.,Sommeijer, B.P. and Couzy, W. (1992). Embedded diagonally implicit Runge-Kutta algorithms on parallel computer. *Maths. Of Comp*.,58: 135-59.

Wilkinson, B. and Allen, M. (2005). *Parallel programming: Techniques and applications using networked workstations and parallel computers*, 2nd. Edition. Pearson Education, Inc. USA.

.