Parallel Solution in Space of Large ODEs Using Block Multistep Method with Step Size Controller

Siamak Mehrkanoon  
Mathematics Department, Faculty Science, University Putra Malaysia  
43400 UPM Serdang, Selangor Darul Ehsan, Malaysia  
E-mail: mehrkanoon@yahoo.com  
Tel: +60172063462

Mohamed Suleiman  
Mathematics Department, Faculty Science, University Putra Malaysia  
43400 UPM Serdang, Selangor Darul Ehsan, Malaysia

Zanariah Abdul Majid  
Mathematics Department, Faculty Science, University Putra Malaysia  
43400 UPM Serdang, Selangor Darul Ehsan, Malaysia

Khairil Iskandar Othman  
Mathematics Department, Faculty of Information Technology and Science Quantitative  
University Technology MARA, 40450 Shah Alam, Selangor, Malaysia

Abstract

The parallel 4-point implicit multistep block method is developed for solving large system of first order ODEs using variable step size. The proposed method computes the numerical solution at four points simultaneously. The Gauss Seidel style is used for the implementation of the method. The parallelism across the system is considered for the parallelization of the method and parallel version has been implemented using MPI communication environment on a High Performance Computer (HPC). The results indicate the advantage of utilizing the parallel implementation of the proposed method for solving large-scale system of ODEs.

Keywords: Block method, parallel algorithms, large system of ODEs.

1. Introduction

The parallel solution of ODE 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

\[ Y' = F(x, Y) \quad Y(a) = Y_0, \quad a \leq x \leq b \]  

Where \( a \) and \( b \) are finite and \( Y' = [y'_1, y'_2, \ldots, y'_n]^T \), \( Y = [y_1, y_2, \ldots, y_n]^T \) and \( F = [f_1, f_2, \ldots, f_n]^T \). Block methods for numerical solution of first order ODEs have been proposed by several authors such as [1, 5, 11, 12]. There exist many parallel differential equation solvers such as in
Miranker and Liniger [9], Cong et al. [4] and Burrage and Suhartanto [2, 3]. The parallel block codes for solving first and higher order ODEs using variable step size and order have been developed by Omar [10]. Majid and Suleiman [7, 8] have proposed three and four point block methods for solving large system of ODEs and Jacobi iteration was used for the implementation of the methods. These codes utilized the same approach for parallelization i.e. parallelism across the method.

The aim of this paper is to introduce the parallel four point block method for solving large system of ODEs. The Gauss Seidel approach is considered for the implementation of the method. The parallelism across the system is applied to the proposed code. The advantage of parallelism across the system over parallelism across the method for proposed block method is that we are not restricted for using arbitrary number of processors while we are limited when using the parallelism across the method (see [8]).

The proposed block method will approximate the solutions at four points simultaneously at each step using variable step size. The method is derived by using Lagrange interpolation polynomial and the closest point in the interval will be considered for obtaining the corrector and predictor formula. Therefore, the approximated values of \( y_{n+1}, y_{n+2}, y_{n+3} \) and \( y_{n+4} \) are obtained by integrating (1) over the interval \([x_n, x_{n+1}], [x_{n+1}, x_{n+2}], [x_{n+2}, x_{n+3}] \) and \([x_{n+3}, x_{n+4}] \) respectively.

### 2. The 4-point implicit block multistep method (4p1b)

In Figure 1, the solution of \( y_{n+1}, y_{n+2}, y_{n+3} \) and \( y_{n+4} \) with step size \( h \) at the points \( x_{n+1}, x_{n+2}, x_{n+3} \) and \( x_{n+4} \) respectively were approximated simultaneously using five back values at the points \( \{x_{n-j}\}^4_{j=0} \), of the previous four steps with step size \( rh \). The set of points \( \{x_{n-7}, \ldots, x_n\} \) are used for deriving the predictor formula and the order is one less than the order of the corrector.

**Figure 1:** 4-point implicit block multistep method

The interpolation points involved for obtaining the corrector formula for \( \{y_{n+j}\}^4_{j=1} \) are \( \{(x_{n-4}, f_{n-4}), \ldots, (x_{n+4}, f_{n+4})\} \). The four values of \( \{y_{n+j}\}^4_{j=1} \) can be derived by integrating (1) over the mentioned intervals respectively. After simplifying using MATHEMATICA one can obtain the following corrector formula in terms of \( r \):
The 1st point
\[ y_{n+1} = y_n + h \left[ \frac{17010r^3 + 13530r^2 + 4005r + 413}{241920r^4(\text{r}+1)(2\text{r}+1)(4\text{r}+1)(4\text{r}+3)} f_{n-4} + \frac{6480r^3 + 4920r^2 + 1335r + 118}{6480r^4(\text{r}+1)(3\text{r}+1)(3\text{r}+2)(3\text{r}+4)} f_{n-3} + \right. \]
\[ \left. \frac{34020r^3 + 23370r^2 + 5340r + 413}{20160r^4(\text{r}+1)(2\text{r}+1)(2\text{r}+3)} f_{n-2} + \frac{136080r^3 + 63960r^2 + 12015r + 826}{15120r^4(\text{r}+1)(r+2)(r+3)(r+4)} f_{n-1} + \right. \]
\[ \left. \frac{253008r^4 + 141750r^3 + 43050r^2 + 6675r + 413}{725760r^4} f_n + \frac{325784r^4 - 394800r^3 + 190260r^2 + 41010r + 3275}{6480(r+1)(2\text{r}+1)(3\text{r}+1)(4\text{r}+1)} f_{n+1} - \right. \]
\[ \left. \frac{44352r^4 + 43050r^3 + 17220r^2 + 3165r + 220}{20160(r+1)(r+2)(2\text{r}+1)(3\text{r}+2)} f_{n+2} + \frac{7632r^4 + 7200r^3 + 2820r^2 + 510r + 35}{6480(r+1)(r+3)(2r+3)(4r+3)} f_{n+3} - \right. \]
\[ \left. \frac{19152r^4 + 17850r^3 + 6930r^2 + 1245r + 85}{241920(r+1)(r+2)(r+4)(3r+4)} f_{n+4}. \right] \]

The 2nd point
\[ y_{n+1} = y_n + h \left[ \frac{6930r^3 + 18810r^2 + 15525r + 3997}{241920r^4(\text{r}+1)(2\text{r}+1)(4\text{r}+1)(4\text{r}+3)} f_{n-4} + \right. \]
\[ \left. \frac{2640r^3 + 6840r^2 + 5175r + 1142}{6480r^4(\text{r}+1)(3\text{r}+1)(3\text{r}+2)(3\text{r}+4)} f_{n-3} - \right. \]
\[ \frac{13860r^3 + 32490r^2 + 20700r + 3997}{20160r^4(\text{r}+1)(2\text{r}+1)(2\text{r}+3)} f_{n-2} + \frac{55440r^3 + 88920r^2 + 46575r + 7994}{15120r^4(\text{r}+1)(r+2)(r+3)(r+4)} f_{n-1} - \right. \]
\[ \left. \frac{19152r^4 + 57750r^3 + 59850r^2 + 25875r + 3997}{725760r^4} f_n + \frac{174384r^4 + 478800r^3 + 454860r^2 + 181710r + 26165}{15120(r+1)(2\text{r}+1)(3\text{r}+1)(4r+1)} f_{n+1} + \right. \]
\[ \left. \frac{76608r^4 + 261450r^3 + 306180r^2 + 149085r + 25820}{20160(r+1)(r+2)(2\text{r}+1)(3r+2)} f_{n+2} - \frac{5328r^4 + 16800r^3 + 18180r^2 + 8190r + 1315}{6480(r+1)(r+3)(2r+3)(4r+3)} f_{n+3} + \right. \]
\[ \left. \frac{11088r^4 + 34650r^3 + 37170r^2 + 16605r + 2645}{241920(r+1)(r+2)(r+4)(3r+4)} f_{n+4}. \right] \]

The 3rd point,
\[ y_{n+1} = y_n + h \left[ \frac{6930r^3 + 32010r^2 + 44325r + 18983}{241920r^4(\text{r}+1)(2\text{r}+1)(4\text{r}+1)(4\text{r}+3)} f_{n-4} + \right. \]
\[ \left. \frac{2640r^3 + 11640r^2 + 14775r + 5398}{6480r^4(\text{r}+1)(3\text{r}+1)(3\text{r}+2)(3\text{r}+4)} f_{n-3} + \right. \]
\[ \frac{13860r^3 + 55290r^2 + 59100r + 18893}{20160r^4(\text{r}+1)(2\text{r}+1)(2\text{r}+3)} f_{n-2} + \frac{55440r^3 + 151320r^2 + 132975r + 37786}{15120r^4(\text{r}+1)(r+2)(r+3)(r+4)} f_{n-1} + \right. \]
\[ \left. \frac{11088r^4 + 57750r^3 + 101850r^2 + 73875r + 18893}{725760r^4} f_n + \frac{37296r^4 + 193200r^3 + 338940r^2 + 244590r + 62245}{15120(r+1)(2\text{r}+1)(3\text{r}+1)(4r+1)} f_{n+1} + \right. \]
\[ \left. \frac{76608r^4 + 376950r^3 + 629580r^2 + 433635r + 105620}{20160(r+1)(r+2)(2\text{r}+1)(3r+2)} f_{n+2} + \frac{24912r^4 + 139200r^3 + 263220r^2 + 204510r + 55955}{6480(r+1)(r+3)(2r+3)(4r+3)} f_{n+3} - \right. \]
\[ \left. \frac{19152r^4 + 101850r^3 + 183330r^2 + 135645r + 35365}{241920(r+1)(r+2)(r+4)(3r+4)} f_{n+4}. \right] \]
The 4th point

\[ y_{n+4} = y_{n+3} + \frac{17010r^3 + 111210r^2 + 217125r + 129997}{241920r^2(r+1)(2r+1)(4r+1)(4r+3)} + \frac{f_{n+4}}{241920r^2(r+1)(2r+1)(4r+1)(4r+3)} f_{n-3} - \]

\[ - \frac{34020r^2 + 192090r^2 + 289500r + 129997}{20160r^2(r+1)(2r+1)(4r+1)(4r+3)} f_{n+2} + \frac{136080r^3 + 525720r^2 + 651375r + 259994}{15120r^2(r+1)(2r+1)(4r+1)(4r+3)} f_{n-1} - \]

\[ - \frac{19152r^4 + 141750r^3 + 353850r^2 + 361875r + 129997}{725760r^4} f_{n+4} + \frac{53424r^4 + 394800r^3 + 984060r^2 + 1004910r + 360485}{15120r^4(r+1)(2r+1)(4r+1)(4r+3)} f_{n+1} - \]

\[ - \frac{44352r^4 + 326550r^3 + 811020r^2 + 825315r + 295060}{20160(r+1)(2r+1)(3r+2)(4r+2)} f_{n+2} + \frac{46512r^4 + 331200r^3 + 796620r^2 + 786210r + 273005}{6480(r+1)(2r+1)(3r+2)(4r+3)} f_{n+3} + \]

\[ - \frac{253008r^4 + 1966650r^3 + 5152770r^2 + 5527005r + 2080805}{241920(r+1)(2r+1)(4r+3)} f_{n+4}. \]

During the implementation of the method, the choices for the next step size will be limited to half, double or the same as the current step size. In the developed codes, 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 size 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. After a successful step, the step size increment is given by:

\[ h_{\text{new}} = \tau \times h_{\text{old}} \times \left( \frac{TOL}{LTE_k} \right)^{\frac{1}{k}}, \]

if \( h_{\text{new}} \geq 2 \times h_{\text{old}} \), then \( h_{\text{new}} = 2 \times h_{\text{old}} \).

In our code, an estimation of local truncation error is obtained by comparing the derived corrector formula of order \( p \) to the same corrector formula of order \( p - 1 \) at the fourth point.

3. Implementation of 4p1b

The method is implemented in \( PE \) (CE) \( m \) mode where \( P \) stands for an application of a predictor, \( E \) stands for an evaluation of a function \( f \), and \( C \) stands for an application of a corrector. During the implementation, the iteration will involve the Gauss Seidel style. The \( m \)th iteration process for the corrector formula is as follows:

\[ C : \quad y_{n+1}^m = y_{n} + \sum_{j=n-4}^{n+4} \beta_{j,k} f_j \]

\[ E : \quad f(x_{n+1}, y_{n+1}^m) \]

\[ C : \quad y_{n+k}^m = y_{n+k-1} + \sum_{j=n-4}^{n+4} \beta_{j,k} f_j \]

\[ E : \quad f(x_{n+k}, y_{n+k}^m) \quad \text{for } k = 2, 3, 4 ; \quad m = 1, 2, \ldots \text{ until convergence} \]

The estimated values of \( y_{n+1}^m \) used the approach of Jacobi iteration. While for obtaining the approximate value of \( \{ y_{n+k}^m \} \) at the points \( \{ x_{n+k} \} \) respectively at the \( m \)th iteration the Gauss Seidel iteration is utilized.

The error calculated are defined as

\[ (e_i) = \frac{(y_i) - (y(x_i))}{A + B(y(x_i))} \]
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 utilized for all the above problems. The maximum error is defined as follow,

\[
\text{MAXE} = \max \left( \max_{1 \leq i \leq N} (e_i)_{t} \right)
\]

where \( N \) is the number of equation in the system and \( p \) is the number of points which at each step their solutions will be estimated. In the code, we iterate the corrector to convergence using the convergence criteria:

\[
|y_{n+4}^{r+1} - y_{n+4}^{r}| < 0.1 \times \text{TOL}
\]

and \( r \) is the number of iteration.

4. **Parallelizing Across the System using MPI**

There are several approaches for parallelizing the proposed sequential program. 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 \( N \) is the number of equations in (1), and \( P \) is the number of processors. In our case \( N \) is divisible by \( P \). Therefore, the share of equations for the \( i^{th} \) processor is from \((i\times L) + 1\) to \((i+1)\times L\), where \( L = \frac{N}{P} \). The parallel algorithm of the proposed 4p1b method is discussed in Algorithm 1.

According to Algorithm 1, all the processors have to exchange their required computed values at Step 1 before calling the evaluation function and it is done by using the \texttt{MPI_Send()} and \texttt{MPI_Recv()} commands. In the code developed we avoid to exchange unnecessary computed values in order to reach a peak speed-up of the parallel codes.

At Step 2 the procedure is executed until convergence. Each processor does the convergent test on its computed values. In case of having satisfactory results each processor calculates its own LTE. All stages at Step 1 and 2 are done simultaneously. At step 3, the master will calculate the next step size and the ratio \( r \) and \( q \). These computed values will be sent to all processors using \texttt{MPI_Bcast()}. Now all the processors in the communicator are 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 MPI.
Algorithm 1: The Parallel iterative algorithm based on proposed block method

<table>
<thead>
<tr>
<th>Step 1</th>
<th>for k=1 to 4 do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor ( i ): Prediction ( { y_{n+k,j}^P }_{j=r+L}^{(i+1)r+L} )</td>
<td></td>
</tr>
<tr>
<td>Exchange of required computed values between processors</td>
<td></td>
</tr>
<tr>
<td>Processor ( i ): Evaluation ( { f_{n+k,j} }_{j=r+L}^{(i+1)r+L} )</td>
<td></td>
</tr>
<tr>
<td>End for.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2</th>
<th>for k=1 to 4 do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor ( i ): Correction ( { y_{n+k,j}^E }_{j=r+L}^{(i+1)r+L} )</td>
<td></td>
</tr>
<tr>
<td>Exchange of required computed values between processors</td>
<td></td>
</tr>
<tr>
<td>Processor ( i ): Evaluation ( { f_{n+k,j} }_{j=r+L}^{(i+1)r+L} )</td>
<td></td>
</tr>
<tr>
<td>End for.</td>
<td></td>
</tr>
</tbody>
</table>

Convergent test: if yes then
- Each processor calculates its own LTE.
- The maximum LTE will be sent to master.
- Go to Step 3.
else go to Step 2.
End if.

| Step 3 | Master calculates the next step size and the value of \( r \) and \( q \). |

5. 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:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOL</td>
<td>Tolerance</td>
</tr>
<tr>
<td>TS</td>
<td>Total Successful Steps</td>
</tr>
<tr>
<td>FS</td>
<td>Total Failure Steps</td>
</tr>
<tr>
<td>FN</td>
<td>Total function calls</td>
</tr>
<tr>
<td>MAXE</td>
<td>Absolute value of the maximum error of the computed solution</td>
</tr>
<tr>
<td>TIME</td>
<td>The execution time taken in microsecond</td>
</tr>
<tr>
<td>S4p1b</td>
<td>Sequential implementation of 4-point implicit block multistep method</td>
</tr>
<tr>
<td>P4p1b</td>
<td>Parallel implementation of 4-point implicit block multistep method</td>
</tr>
<tr>
<td>#p</td>
<td>Number of Processor used</td>
</tr>
</tbody>
</table>

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:

\[
S_p(n) = \frac{T_S(n)}{T_p(n)}, \quad E_p(n) = \frac{S_p(n)}{P}
\]

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 which has 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% percentages.
Problem 1: A radioactive decay chain.
\[
\begin{bmatrix}
    y'_1 \\
    y'_2 \\
    \vdots \\
    y'_N \\
\end{bmatrix}
= \begin{bmatrix}
    -1 & 0 & \cdots & 0 \\
    1 & -1 & \cdots & \vdots \\
    \vdots & \vdots & \vdots & \vdots \\
    0 & \cdots & 1 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_N \\
\end{bmatrix},
\quad y(0) = \begin{bmatrix}
    \vdots \\
\end{bmatrix}
\]

\(N = \text{Number of equations, } 0 \leq x \leq b\). Source: Hull et al. [6].

Problem 2: A parabolic partial differential equations.
\[
\begin{bmatrix}
    y'_1 \\
    y'_2 \\
    \vdots \\
    y'_N \\
\end{bmatrix}
= \begin{bmatrix}
    -2 & 1 & 0 & \cdots & 0 \\
    1 & -2 & \cdots & \vdots \\
    \vdots & \vdots & \vdots & \vdots \\
    0 & \cdots & 1 & -2 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_N \\
\end{bmatrix},
\quad y(0) = \begin{bmatrix}
    \vdots \\
\end{bmatrix}
\]

\(N = \text{Number of equations, } 0 \leq x \leq b\). Source: Hull et al. [6].

Problem 1 and 2 are solved without exact reference solution. To measure the accuracy of the code, the solutions obtained by the method 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. Therefore the presented execution times do not include the elapsed time for calculating the errors. The performance of the code were 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.

Table 1 shows the obtained total steps, function calls and maximum errors of 4p1b method when solving Problem 1 and 2. It should be noted that the sequential and parallel implementation of the method has resulted in the same number of total step, function calls and maximum errors. The reported results in Table 1 were obtained when \(N=6000\) and for each different \(N\) value we will get the same results.

<table>
<thead>
<tr>
<th>TOL</th>
<th>Problem 1, [0,10]</th>
<th>Problem 2, [0,5]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TS</td>
<td>FS</td>
</tr>
<tr>
<td>10^{-2}</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>10^{-4}</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>10^{-6}</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>10^{-8}</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>10^{-10}</td>
<td>51</td>
<td>0</td>
</tr>
</tbody>
</table>

The observed execution times and speed-ups of 4p1b method when solving problem 1 and 2 are reported in Table 2 and 3. Note that the values in the square brackets give the speed-up of the parallel code. It is apparent that the parallel execution times are faster compared to sequential timing, particularly for large number of processors. In both tables the expected growth of the speed-up with number of processors may be observed. Concerning Table 2 and 3, it is worthwhile to mention that
there exists a moderate growth of speed-up with increasing number of equations. This is the feature of the proposed parallel block method. Namely, in solving problem 1 the speed-ups gained by the parallel performance of 4p1b code using six processors when N=6000, 18000 and 30000 are [2.52], [4.43] and [4.81] respectively at TOL=$10^{-10}$. In problem 2 the obtained speed-ups also improved when dimension N increased. Thus, the performance of the parallel code has improved for large system of ODEs.

Table 2: Measured execution times and speed-ups of 4p1b method for solving Problem 1, [0,10]

<table>
<thead>
<tr>
<th>TOL</th>
<th>N</th>
<th>MTD</th>
<th>#P</th>
<th>10^{-2}</th>
<th>10^{-4}</th>
<th>10^{-6}</th>
<th>10^{-8}</th>
<th>10^{-10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>4p1b</td>
<td>1</td>
<td>0.77 [1.00]</td>
<td>0.68 [1.00]</td>
<td>0.97 [1.00]</td>
<td>1.27 [1.00]</td>
<td>1.81 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.46 [1.66]</td>
<td>0.39 [1.60]</td>
<td>0.59 [1.53]</td>
<td>0.81 [1.44]</td>
<td>1.17 [1.40]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.32 [2.50]</td>
<td>0.27 [2.46]</td>
<td>0.43 [2.12]</td>
<td>0.61 [1.91]</td>
<td>0.89 [1.81]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.25 [3.57]</td>
<td>0.21 [3.36]</td>
<td>0.35 [2.57]</td>
<td>0.50 [2.26]</td>
<td>0.75 [2.12]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.21 [4.16]</td>
<td>0.18 [4.11]</td>
<td>0.30 [3.07]</td>
<td>0.44 [2.53]</td>
<td>0.67 [2.36]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.19 [5.00]</td>
<td>0.16 [4.62]</td>
<td>0.27 [3.32]</td>
<td>0.40 [2.74]</td>
<td>0.62 [2.52]</td>
<td></td>
</tr>
<tr>
<td>18000</td>
<td>4p1b</td>
<td>1</td>
<td>2.59 [1.00]</td>
<td>2.32 [1.00]</td>
<td>3.16 [1.00]</td>
<td>4.03 [1.00]</td>
<td>5.61 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1.05 [2.84]</td>
<td>0.92 [2.93]</td>
<td>1.25 [2.80]</td>
<td>1.58 [2.68]</td>
<td>2.20 [2.62]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.69 [3.79]</td>
<td>0.60 [3.91]</td>
<td>0.86 [3.61]</td>
<td>1.15 [3.42]</td>
<td>1.64 [3.32]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.56 [4.78]</td>
<td>0.48 [4.86]</td>
<td>0.72 [4.44]</td>
<td>0.96 [4.05]</td>
<td>1.38 [3.91]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.52 [5.35]</td>
<td>0.42 [5.42]</td>
<td>0.62 [5.16]</td>
<td>0.88 [4.63]</td>
<td>1.24 [4.43]</td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>4p1b</td>
<td>1</td>
<td>4.48 [1.00]</td>
<td>4.00 [1.00]</td>
<td>5.41 [1.00]</td>
<td>6.85 [1.00]</td>
<td>9.50 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.79 [5.44]</td>
<td>0.67 [5.58]</td>
<td>0.95 [5.31]</td>
<td>1.26 [4.98]</td>
<td>1.79 [4.81]</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Measured execution times and speed-ups of 4p1b method for solving Problem 2, [0,5]

<table>
<thead>
<tr>
<th>TOL</th>
<th>N</th>
<th>MTD</th>
<th>#P</th>
<th>10^{-2}</th>
<th>10^{-4}</th>
<th>10^{-6}</th>
<th>10^{-8}</th>
<th>10^{-10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>4p1b</td>
<td>1</td>
<td>0.71 [1.00]</td>
<td>0.69 [1.00]</td>
<td>0.88 [1.00]</td>
<td>1.20 [1.00]</td>
<td>1.65 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.38 [1.57]</td>
<td>0.35 [1.57]</td>
<td>0.44 [1.50]</td>
<td>0.63 [1.41]</td>
<td>0.88 [1.38]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.35 [2.20]</td>
<td>0.30 [2.53]</td>
<td>0.40 [2.09]</td>
<td>0.57 [1.88]</td>
<td>0.81 [1.79]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.27 [3.14]</td>
<td>0.25 [3.30]</td>
<td>0.32 [2.50]</td>
<td>0.48 [2.22]</td>
<td>0.69 [2.10]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.23 [3.66]</td>
<td>0.21 [4.12]</td>
<td>0.29 [2.87]</td>
<td>0.43 [2.46]</td>
<td>0.62 [2.30]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.21 [4.40]</td>
<td>0.20 [4.71]</td>
<td>0.26 [3.13]</td>
<td>0.40 [2.66]</td>
<td>0.57 [2.47]</td>
<td></td>
</tr>
<tr>
<td>18000</td>
<td>4p1b</td>
<td>1</td>
<td>2.51 [1.00]</td>
<td>2.41 [1.00]</td>
<td>2.86 [1.00]</td>
<td>3.75 [1.00]</td>
<td>5.07 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.70 [3.69]</td>
<td>0.64 [3.90]</td>
<td>0.79 [3.63]</td>
<td>1.08 [3.40]</td>
<td>1.50 [3.29]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.58 [4.47]</td>
<td>0.53 [4.80]</td>
<td>0.66 [4.33]</td>
<td>0.91 [4.01]</td>
<td>1.27 [3.86]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.51 [5.31]</td>
<td>0.46 [5.43]</td>
<td>0.57 [5.04]</td>
<td>0.81 [4.55]</td>
<td>1.12 [4.37]</td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>4p1b</td>
<td>1</td>
<td>4.33 [1.00]</td>
<td>4.17 [1.00]</td>
<td>4.90 [1.00]</td>
<td>6.41 [1.00]</td>
<td>8.64 [1.00]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>0.81 [5.52]</td>
<td>0.72 [5.69]</td>
<td>0.88 [5.33]</td>
<td>1.20 [4.98]</td>
<td>1.65 [4.81]</td>
<td></td>
</tr>
</tbody>
</table>
The measured execution times of the proposed 4p1b method when solving problem 1 and 2 (using fixed tolerance $10^{-10}$) are depicted in Figure 2-4. In these figures it is apparent that increasing number of processors leads to the decline of execution time. On the other hand as the problem dimension $N$ increases, the influence of the communication time is reduced and an improvement of speed-up is achieved which can be seen exactly in tables 2-3.

**Figure 2:** Sequential and parallel timing comparison when $N=6000$, $TOL=10^{-10}$

**Figure 3:** Sequential and parallel timing comparison when $N=18000$, $TOL=10^{-10}$
6. Conclusion

The parallel 4-point implicit block multistep method has been developed for solving large system of ODEs using variable step size on a High Performance Computer. By using large-scale problems and by parallelism across the system, the method has shown the superiority of the parallel code over the sequential one. The numerical results indicate that the parallel implementation of proposed 4p1b method for solving large-scale problem leads to a considerable decline of computation cost without losing the desired accuracy. Also the speed-up improves as the problem size increases. In fact, the speed-up is approaching the linear speed-up as the number of equations increased.
References


