Parallel Block Backward Differentiation Formulas For Solving Large Systems of Ordinary Differential Equations

Zarina Bibi, I., Khairil Iskandar, O.

Abstract—In this paper, parallelism in the solution of Ordinary Differential Equations (ODEs) to increase the computational speed is studied. The focus is the development of parallel algorithm of the two point Block Backward Differentiation Formulas (PBBDF) that can take advantage of the parallel architecture in computer technology. Parallelism is obtained by using Message Passing Interface (MPI). Numerical results are given to validate the efficiency of the PBBDF implementation as compared to the sequential implementation.

Keywords—Ordinary differential equations, parallel.

I. INTRODUCTION

We shall consider parallel Block Backward Differentiation Formulas (PBBDF) for the numerical solution of initial value problems (IVPs) for the first order Ordinary Differential Equations (ODEs) of the form

\[
\begin{align*}
\frac{dy}{dx} &= f(x,y), \quad x \in [0,X] \\
y(0) &= y_0
\end{align*}
\]

where \( f : \mathbb{R} \times \mathbb{R}^m \rightarrow \mathbb{R}^m \). For all \( x \in [0,X] \),
\[
\|f(x,y) - f(x,z)\| \leq L\|y - z\|, \quad L \text{ is a Lipschitz constant.}
\]

Most of the existing numerical methods for solving (1) are sequential in nature. Various approaches to solve (1) using multiple processor computer system with emphasis on reduction the computation time is due to the recent advances in computer technology. Many researchers develop or modified existing numerical methods to fully utilize the parallel architecture so that some of the computations can be executed simultaneously on multiple processor computer system. Generally, parallelism can be achieved by partitioning the tasks across the methods or across the system of equations. Parallel block methods have been proposed by many researchers to speed up the integration of (1). Some of the earlier works on parallelism on ODEs are found in Gear [3], Bellen and Zennaro [1], Franklin [4], and Chu and Hamilton [2] to name a few. In [3] parallelism is classified as parallelism by partitioning the tasks either “across the method” or “across the systems of equations”. [1] introduced parallelism across the time which means that each processor evaluates \( f \) for different values of \( x \).

The paper is organized as follows. The PBBDF method is presented in Section II. In Section III, a detailed implementation of the PBBDF method using Message Passing Interface (MPI) is given. Section IV provides numerical result to validate the efficiency of the parallel algorithm of the PBBDF method. The conclusions are given in Section V.

II. THE BLOCK BACKWARD DIFFERENTIATION FORMULAS

In this section, we reviewed a class of block multistep methods proposed by Ibrahim et. al in [6] which is called Block Backward Differentiation Formulas for solving stiff ODEs. The method given in [6] will compute the solutions of Initial Value Problems (IVPs) at two points simultaneously on the x-axis, i.e. \( y_{n+1} \) and \( y_{n+2} \). The solver start with constant step size which is formulated as

\[
\begin{align*}
\frac{1}{10}y_{n-2} + \frac{3}{5}y_{n-1} - \frac{9}{5}y_n + y_{n+1} + \frac{3}{10}y_{n+2} = \frac{6}{5}hf_{n+1} \\
\frac{3}{25}y_{n-2} - \frac{16}{25}y_{n-1} + \frac{36}{25}y_n - \frac{48}{25}y_{n+1} + y_{n+2} = \frac{12}{25}hf_{n+2}
\end{align*}
\]

The step size choosing strategy is based on the estimate of the local truncation error (LTE). The step is accepted if the LTE compared with the prescribed tolerance limit, TOL satisfy LTE < TOL and rejected otherwise. Denoting tolerance by \( \varepsilon \),

the next step size \( h_{new} \) is computed by

\[
h_{new} = c \times h_{bad} \times \left( \frac{\|y_{n+1} - y_{n+2}\|^p}{\|y_{n+1} - y_{n+2}\|^p} \right)^{1/p}
\]

where \( c \) is the safety factor and \( p \) is the order of the PBBDF method. For our code, we take the safety factor as 0.8.

The next step size is increased by a factor of 1.6 to speed up the computation. The PBBDF solver with the increased step size 1.6\( h \) is formulated as
Applying Newton iteration to the matrix above by letting

\[
P \frac{\partial F}{\partial y^{n+1}} y^{n+1}= \frac{\partial F}{\partial y^{n+1}} y^{n+2} = \frac{546}{1195} h f_{n+2}
\]

where \((I-A)\frac{\partial F}{\partial y^{n+1}} y^{n+2}\) is the Jacobian matrix of \(\hat{F}\) with respect to \(y\). To reduce the amount of computations, the Jacobian matrix is updated when there is a consecutive step failure in the integration i.e. LTE > ε. The starting values were computed from the exact solution if available or by using the Euler method.

III. PARALLEL IMPLEMENTATION OF BBDF

In this section, we discuss the parallel implementation of the BBDF method which allows the distribution of tasks amongst the available processors in order to reduce the execution time. Simultaneous approximations for several steps are obtained using the Message Passing Interface (MPI) library which runs on a High Performance Computer (HPC). These parallel implementations are based on the master – slave approach. The computation occurs only in the slaves while the master broadcast all the data needed by the slaves. The subprogram, JAC1, calculates the Newton-iteration for the PBBDF method. The matrix multiplication is given as

\[
[ h B \frac{\partial F}{\partial y^{n+1}} y^{n+2} ]
\]

One way of performing the multiplication in parallel is to have each processor compute different parts of the product. Consider matrix \(A_{ik}, B_{kj}\) and \(C_{ij} = \sum_{k=1}^{n_P} A_{ik} B_{kj}\).

i) First, the matrix \(A\) is partition by rows and \(B\) by columns:

\[
\begin{bmatrix}
  a_{11} & a_{12} & ... & a_{1i} \\
  a_{21} & a_{22} & ... & a_{2i} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{ni} & a_{ni} & ... & a_{ni}
\end{bmatrix}
\times
\begin{bmatrix}
  b_{11} & b_{12} & ... & b_{1j} \\
  b_{21} & b_{22} & ... & b_{2j} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{nj} & b_{nj} & ... & b_{nj}
\end{bmatrix}
= \begin{bmatrix}
  c_{11} & c_{12} & ... & c_{1j} \\
  c_{21} & c_{22} & ... & c_{2j} \\
  \vdots & \vdots & \ddots & \vdots \\
  c_{nj} & c_{nj} & ... & c_{nj}
\end{bmatrix}
\]

\(a_{ij}\) is done by slaves, \(b_{ij}\) (master do the broadcast)

ii) The matrix \(A\) is striped row-wise among the slaves \(P_1, P_2, ..., P_n\) so that each processor is assigned to one row. In order, to avoid any processor been idle, the processor that finish the computation early, will automatically take the next row. Each process in row \(i\) will need all the values in column \(j\). Therefore, processor \(P_0\) referred as master will broadcast the entire matrix \(B\) of size \(k \times j\) to all the slaves \(P_1, P_2, ..., P_n\) as needed prior to the start of the multiplication. Take note that the computation occurs only in the slaves while the master broadcast all the data needed by the slaves.

IV. NUMERICAL RESULTS

Problem 1: Brusselator systems
Brusselator systems is a nonlinear partial differential equation which arise in the modeling of chemical reaction-diffusion which is of the form

\[
\frac{\partial u}{\partial t} = A + u^2 v - (B + 1)u + \alpha \frac{\partial^2 u}{\partial x^2}
\]

\[
\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2}
\]

with \( x \in [0,1] \), \( \alpha \geq 0 \), \( A \) and \( B \) are the constant parameters. In this paper, we consider \( A = 1, B = 3, \alpha = 1/50 \) and boundary conditions for \( u \) and \( v \) which are given by

\[
u(0,t) = 1 = u(1,t) , \quad v(0,t) = 3 = v(1,t)
\]

\[
u(x,0) = 1 + \sin(2\pi x) , \quad v(x,0) = 3 .
\]

By applying the method of lines, we obtain a system of differential equations to be solved on the interval \( 0 \leq x \leq 10 \).

\[
\frac{du}{dt} = 1 + u^2 v_i - 4u_i + \alpha (N+1)^2 (u_{i-1} - 2u_i + u_{i+1})
\]

\[
\frac{dv_i}{dt} = 3u_i - u^2 v_i + \alpha (N+1)^2 (v_{i-1} - 2v_i + v_{i+1})
\]

with \( u_i(t) = 1 = u_{N+1}(t) \), \( v_i(t) = 3 = v_{N+1}(t) \), \( u_i(0) = 1 + \sin(2\pi x_i) \), \( v_i(0) = 3 \), \( x_i = \frac{i}{N+1} \), \( i = 1,..,N \).

Equation (7) is used to illustrate the performance of the PBBDF method. For more details of this example see Nicolis and Prigogine [7] and Prigogine and Lefever [8].

Problem 2:

\[
\begin{bmatrix}
\dot{y}_1 \\
\dot{y}_2 \\
\vdots \\
\dot{y}_N
\end{bmatrix}
= \begin{bmatrix}
1 & -2 & \cdots & 0 \\
2 & -3 & \cdots & \cdots \\
& & \ddots & \cdots \\
& & & 0 & -N+1 \\
& & & & N-1 & 0
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_N
\end{bmatrix}
\]

\( N \) = number of equations, Initial values: \( y(0) = (1,0,\ldots,0)^T \)

Interval: \( 0 \leq x \leq 20 \)

Source: Hull, T.E. et al. [5].

The numerical results are performed on Sunfire V1280 HPC. Parameters evaluated are the execution time, speedup and efficiency. The notations used in the tables take the following meaning,

\( S_p \) : Speedup

\( E_p \) : Efficiency

\( \text{EQN} \) : Number of equations

\( \text{TIM} \) : The execution time in seconds

\( E \) : The speedup of a parallel algorithm is defined as

\[
S_p = \frac{T_s}{T_p}
\]

where \( T_s \) is the execution time of sequential algorithm using one processor and \( T_p \) is the execution time by a parallel algorithm using \( p \) processors. The efficiency of the parallel algorithm denoted by \( E_p \) is defined as the ratio of speedup to the number of processors

\[
E_p = \frac{S_p}{p}
\]

Theoretically, the vailue of efficiency is \( 0 < E_p \leq 1 \). Table 1 shows the speedup and efficiency for Problem 1 when run with different number of processors.

<table>
<thead>
<tr>
<th>( S_p )</th>
<th>( 2p )</th>
<th>( 4p )</th>
<th>( 6p )</th>
<th>( 8p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.876</td>
<td>2.151</td>
<td>2.987</td>
<td>3.245</td>
</tr>
<tr>
<td>60</td>
<td>1.146</td>
<td>3.381</td>
<td>5.556</td>
<td>7.434</td>
</tr>
<tr>
<td>100</td>
<td>1.158</td>
<td>3.454</td>
<td>5.712</td>
<td>7.259</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( E_p )</th>
<th>( 2p )</th>
<th>( 4p )</th>
<th>( 6p )</th>
<th>( 8p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.438</td>
<td>0.538</td>
<td>0.498</td>
<td>0.406</td>
</tr>
<tr>
<td>60</td>
<td>0.573</td>
<td>0.845</td>
<td>0.926</td>
<td>0.929</td>
</tr>
<tr>
<td>100</td>
<td>0.579</td>
<td>0.863</td>
<td>0.952</td>
<td>0.907</td>
</tr>
</tbody>
</table>

Note that the speed up is approaching the linear speedup as the number of equations increased.

Table 2a and 2b shows the speedup and efficiency for Problem 2 when run with different number of processors at difference tolerance.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \text{NP} )</th>
<th>( 2 )</th>
<th>( 4 )</th>
<th>( 6 )</th>
<th>( 8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.976</td>
<td>2.275</td>
<td>3.045</td>
<td>3.262</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1.094</td>
<td>3.149</td>
<td>5.052</td>
<td>6.621</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.116</td>
<td>3.303</td>
<td>5.409</td>
<td>6.868</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1.124</td>
<td>3.342</td>
<td>5.546</td>
<td>7.460</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>1.130</td>
<td>3.384</td>
<td>5.604</td>
<td>7.295</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>0.976</td>
<td>2.275</td>
<td>3.045</td>
<td>3.262</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( E_p )</th>
<th>( 2p )</th>
<th>( 4p )</th>
<th>( 6p )</th>
<th>( 8p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.488</td>
<td>0.569</td>
<td>0.508</td>
<td>0.408</td>
</tr>
<tr>
<td>50</td>
<td>0.547</td>
<td>0.787</td>
<td>0.842</td>
<td>0.828</td>
</tr>
<tr>
<td>100</td>
<td>0.558</td>
<td>0.826</td>
<td>0.902</td>
<td>0.839</td>
</tr>
<tr>
<td>150</td>
<td>0.562</td>
<td>0.836</td>
<td>0.924</td>
<td>0.933</td>
</tr>
<tr>
<td>200</td>
<td>0.565</td>
<td>0.846</td>
<td>0.934</td>
<td>0.912</td>
</tr>
<tr>
<td>300</td>
<td>0.488</td>
<td>0.569</td>
<td>0.508</td>
<td>0.408</td>
</tr>
</tbody>
</table>

\( N \) : Number of processors

\( \text{NP} \) : Number of processors
TABLE IIb: $\varepsilon = 10^{-6}$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$NP$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>$Sp$</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0.985</td>
</tr>
<tr>
<td>50</td>
<td>1.095</td>
</tr>
<tr>
<td>100</td>
<td>1.116</td>
</tr>
<tr>
<td>150</td>
<td>1.120</td>
</tr>
<tr>
<td>200</td>
<td>1.130</td>
</tr>
<tr>
<td>300</td>
<td>0.985</td>
</tr>
<tr>
<td>$Ep$</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0.492</td>
</tr>
<tr>
<td>50</td>
<td>0.547</td>
</tr>
<tr>
<td>100</td>
<td>0.558</td>
</tr>
<tr>
<td>150</td>
<td>0.560</td>
</tr>
<tr>
<td>200</td>
<td>0.565</td>
</tr>
<tr>
<td>300</td>
<td>0.492</td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper, we have presented the parallel implementation of the PBBDF method for solving large systems of ordinary differential equations. The parallel implementation of the PBBDF method shows significant gains over the sequential implementation. The resulting speed-up validates the efficiency of the PBBDF method as the number of equations increased.

ACKNOWLEDGMENT

This research is supported by the Institute of Mathematical Research (INSPEM, University Putra Malaysia (UPM) under Fundamental Research Grant Scheme (FRGS).}

REFERENCES