Abstract—A parallel block method based on Backward Differentiation Formulas (BDF) is developed for the parallel solution of stiff Ordinary Differential Equations (ODEs). Most common methods for solving stiff systems of ODEs are based on implicit formulae and solved using Newton iteration which requires repeated solution of systems of linear equations with coefficient matrix, \( I - hB \). Here, \( J \) is the Jacobian matrix of the problem. In this paper, the matrix operations is paralleled in order to reduce the cost of the iterations. Numerical results are given to compare the speedup and efficiency of parallel algorithm and that of sequential algorithm.

Keywords—Backward Differentiation Formula, block, ordinary differential equations.

I. INTRODUCTION

We consider block method for the parallel solution of Ordinary Differential Equations (ODEs)

\[ y' = f(x,y) \]  

with initial values \( y(a) = y_0 \) in the interval \( x \in [a,b] \).

Various parallel block methods have been proposed for the parallel solution of (1). Watts and Shampine [5], Worland [6], Birta and Abou-Rabia [1], Chu and Hamilton [2] to name a few. Earlier work on parallelism in ODE are found in Rosser [4], Watts [5] and Gear [3]. Gear classifies parallelism into two categories: (i) parallelism across time which is also referred as “parallelism across the method” and (ii) parallelism across the systems. In parallelism across time, each processor executes a different part of the method. Parallelism across the systems where the systems are divided into a set of sub-systems and each sub-system is assigned to a different processor.

In the next section, we reviewed a class of block methods proposed by Zarina et. al in [7] which are based on Backward Differentiation Formulas (BDF) for solving stiff ODEs. Such methods are called Block Backward Differentiation Formulas (BBDF).

II. THE BBDF METHOD

Traditionally, the BDF computation proceeds to an approximation \( y_{n+1} \) of \( y(x_{n+1}) \) one step at a time while in 2-point BBDF, the approximation solutions \( y_{n+1} \) and \( y_{n+2} \) are obtained simultaneously in every step. The simultaneous sequence of computation symbolized as PECE.

The first point approximation is

\[ \Rightarrow y_{n+1}^{PECE} (\text{Predict} : P) \rightarrow f_{n+1}^{PECE} (\text{Evaluate} : E) \rightarrow y_{n+1}^{PECE} (\text{Correct} : C) \rightarrow f_{n+1}^{PECE} (\text{Evaluate} : E) \]

and the second point approximation is

\[ \Rightarrow y_{n+2}^{PECE} (\text{Predict} : P) \rightarrow f_{n+2}^{PECE} (\text{Evaluate} : E) \rightarrow y_{n+2}^{PECE} (\text{Correct} : C) \rightarrow f_{n+2}^{PECE} (\text{Evaluate} : E) \]

In 2 point BBDF method, the interval \([a,b]\) is divided into series of blocks with each block containing two equally spaced points (see Fig. 1).

\[ rh \quad rh \quad h \quad h \]

Fig. 1 2-point block method

Let the step size, \( h \) of the computed block be \( 2h \) and the step size of the previous block be \( 2rh \) where \( r \) is the step size ratio. In the step size selection, the decrease of the step size when there is a step failure is limited to halving while the increment of the step size is increased by a factor of \( 1.6h \) to ensure zero stability. The BBDF method expressed in the general form is giveb by

\[
\begin{align*}
\begin{bmatrix}
y_{n+1} \\
y_{n+2}
\end{bmatrix} &= \begin{bmatrix}
\theta_1 & \alpha_1 h f_{n+1} + \psi_1 \\
\theta_2 & \alpha_2 h f_{n+2} + \psi_2
\end{bmatrix} \\
\end{align*}
\]

with \( \psi_1 \) and \( \psi_2 \) are the backvalues.

Equation (2) written in matrix-vector form is equivalent to

\[
(I - A) \begin{bmatrix}
y_{n+1} \\
y_{n+2}
\end{bmatrix} = B \begin{bmatrix}
\alpha_1 & 0 \\
0 & \alpha_2
\end{bmatrix} \begin{bmatrix}
y_{n+1} \\
y_{n+2}
\end{bmatrix} + hBF_{n+1,n+2} + \tilde{\varepsilon}_{n+1,n+2}
\]

with

\[
I = \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}, \quad Y_{n+1,n+2} = \begin{bmatrix}
y_{n+1} \\
y_{n+2}
\end{bmatrix}, \quad A = \begin{bmatrix}
\theta_1 & 0 \\
0 & \theta_2
\end{bmatrix}, \quad B = \begin{bmatrix}
\alpha_1 & 0 \\
0 & \alpha_2
\end{bmatrix},
\]

\[
F_{n+1,n+2} = \begin{bmatrix}
f_{n+1} \\
f_{n+2}
\end{bmatrix} \quad \text{and} \quad \varepsilon_{n+1,n+2} = \begin{bmatrix}
\psi_1 \\
\psi_2
\end{bmatrix}.
\]

Let

\[
\tilde{F}_{n+1,n+2} = (I - A) Y_{n+1,n+2} - hBF_{n+1,n+2} - \tilde{\varepsilon}_{n+1,n+2} = 0.
\]

To approximate this solution, select \( y_{n+1}^{(i)} \) and generate \( y_{n+1,n+2}^{(i)} \) by applying Newton’s Iteration to the system (2) to obtain,
\[ y(i)^{n+1} = -y_1 y_2 - 19 y_2 \]
\[ y_2(i)^{n+1} = -19 y_1 - 20 y_2 \]

Initial values:
\[ y_1(0) = 2, \quad y_2(0) = 0 \]

Interval:
\[ 0 \leq x \leq 20 \]

Solution:
\[ y_1(x) = e^{-39x + e^{-x}} \]
\[ y_2(x) = e^{-39x - e^{-x}} \]

### III. PARALLEL IMPLEMENTATION OF BBDF

This section describes the parallel implementation on the matrix multiplication. Parallelism is obtained using the Message Passing Interface (MPI) library which runs on a High Performance Computer (HPC).

In BBDF code, the matrix multiplication is
\[
\left[ h B \frac{\partial F}{\partial Y} (y(i)^{n+1}) \right]
\]

In order to parallel the matrix multiplication, the matrices involved must be assigned to different processors. This is done by first distributing the matrix JACBN (calculates the Jacobian) to all processors using the command MPI_BCast(JACBN). Next, rows of the matrix can be formed simultaneously. The setting up of the matrix multiplications is done by multiplying the processors to each column of matrix JACBN, and the results are transferred back master into the matrix JACBN1. This process is done simultaneously. The up setting of the matrix multiplications is done as follows:

<table>
<thead>
<tr>
<th>Procs</th>
<th>NEWB</th>
<th>JACBN</th>
<th>JACBN1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>x_2</td>
<td>a_2</td>
<td>e_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P_n</td>
<td>x_2</td>
<td>a_2</td>
<td>e_2</td>
</tr>
</tbody>
</table>

### IV. NUMERICAL RESULTS

Two test problems are considered. Problem 1 is given to validate the method BBDF works. Problem 2 is a reaction-diffusion problem, the Brusselator system.

### TABLE I

<table>
<thead>
<tr>
<th>TOL</th>
<th>MTD</th>
<th>FA</th>
<th>IST</th>
<th>TS</th>
<th>MAXE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^-2</td>
<td>BDF</td>
<td>11</td>
<td>60</td>
<td>71</td>
<td>1.9588e-01</td>
<td>19064</td>
</tr>
<tr>
<td></td>
<td>BDF</td>
<td>0</td>
<td>32</td>
<td>32</td>
<td>3.7078e-04</td>
<td>6083</td>
</tr>
<tr>
<td>10^-4</td>
<td>BDF</td>
<td>21</td>
<td>120</td>
<td>141</td>
<td>3.598e-03</td>
<td>23568</td>
</tr>
<tr>
<td></td>
<td>BDF</td>
<td>2</td>
<td>63</td>
<td>65</td>
<td>3.5710e-06</td>
<td>9866</td>
</tr>
<tr>
<td>10^-6</td>
<td>BDF</td>
<td>26</td>
<td>197</td>
<td>223</td>
<td>3.2120e-05</td>
<td>34418</td>
</tr>
<tr>
<td></td>
<td>BDF</td>
<td>0</td>
<td>170</td>
<td>170</td>
<td>2.6545e-08</td>
<td>19231</td>
</tr>
</tbody>
</table>
Tables II shows the speedup and efficiency for the Brussellator problem when run with different number of processors.

<table>
<thead>
<tr>
<th></th>
<th>Eqn 20</th>
<th>Eqn 60</th>
<th>Eqn 100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2p</td>
<td>4p</td>
<td>6p</td>
</tr>
<tr>
<td>Speedup</td>
<td>0.876</td>
<td>1.146</td>
<td>1.158</td>
</tr>
<tr>
<td>Efficiency</td>
<td>0.438</td>
<td>0.573</td>
<td>0.579</td>
</tr>
</tbody>
</table>

1P= 1 processor, 2P= 2 processor, 4P=4 processor, 6P= 6 processor, 8P= 8 processor.

V. Conclusion

The numerical results showed that the speedup improves as the problem size increases. In fact, the speed up is approaching the linear speedup as the number of equations increased. Therefore, the parallel implementation of the BBDF methods shows significance gains over the sequential BDF.

ACKNOWLEDGEMENT

This research was supported by Universiti Technology MARA under Fundamental Research Grant Scheme (FRGS).

REFERENCES