Chiang Mai J. Sci. 2006; 33(1) : 9-21
www.science.cmu.ac.th/journal-science/josci.html
Contributed Paper

Designing Parallel Algorithms for Solving Higher Order Ordinary Differential Equations Directly on a Shared Memory Parallel Computer Architecture

Zurni B. Omar* [a] and Mohamed B. Suleiman [b]
[a] Faculty of Quantitative Sciences, Universiti Utara Malaysia, 06010 UUM, Sintok, Kedah, Malaysia.
[b] National Accreditation Board, Menara PKNS-PJ No.17, 46050 Petaling Jaya, Selangor, Malaysia.
*Author for correspondence, e-mail: zurni@uum.edu.my

Received : 16 December 2004
Accepted : 23 September 2005

ABSTRACT

Parallelism is the long-term answer to powerful computation. However, the design of parallel algorithms highly depends on the parallel computer architectures. Therefore, understanding the computer architecture is very crucial before any algorithm can be developed. In this paper, new parallel algorithms namely 2-point and 3-point explicit block methods for solving higher order ordinary differential equations (ODEs) directly are introduced. These methods compute the numerical solution at two and three points simultaneously and therefore exploit the advantages of a shared memory parallel computer architecture. Computational advantages are presented comparing the results obtained by the new methods with the conventional 1-point explicit method.

Keywords: block method, directly, ordinary differential equations, Parallel.

1. INTRODUCTION

Over the past sixty years we witnessed great improvements in computing speed. These were due to the use of better and faster electronic components by computer manufacturers. However, this trend will soon come to an end [2]. This is because the current serial computer hardware technology is reaching to the point where further performance improvements are very hard to achieve. Users then started to realise that additional performance can be achieved by taking multiprocessor route. Due to the declining cost for computer hardware, it is possible to assemble parallel machines with millions of processors. Furthermore, there is no inherent limit to the expansion of parallel architectures [1].

At the early stage of parallel computer development, users were not expected to be aware of the architectural aspects. Programs seemed to work on most of the computer systems. The only hardware feature they had to be aware was the available memory. However, currently with the present of more complicated parallel computers especially in the distributed memory machines, users must also consider the architectural aspect of those computers while programming. The choice of algorithm and the way that an algorithm is expressed depend on the parallel computer architectures. Flynn attempted to classify the various types of computer based on how a machine relates its instruction stream to the data stream [7]. Flynn's taxonomy categorises the computer into four classes as follows

i) Single instruction stream, single data stream (SISD)
ii) Single instruction stream, multiple data stream (SIMD)

iii) Multiple instruction stream, single data stream (MISD)
iv) Multiple instruction stream, multiple data stream (MIMD)

The term SISD refers to the classical sequential design in which a single stream of instructions operates on a single data stream. A computer of this class consists of a single processing, control and memory unit. A MISD computer has several processors and each one of them has its own control unit that shares a common memory unit. Several streams of instructions are operated on a single stream of data simultaneously. No systems have been built that fit this classification. In a SIMD system, there is a master control unit and a number of identical processors. The control unit transmits the same instruction stream to each of the processors. Therefore, each processor performs the same instruction simultaneously on different sets of data. On the other hand, an MIMD system consists of a number of control units and identical processors. Each of the processors may be instructed to perform a different operation, i.e one processor does addition and the other one does multiplication, etc. The processors can communicate among themselves during computation in two ways: through a shared memory space or communication network. MIMD is the most general model of parallelism and best suited to large problems [12].

Besides the four classes of computer as described before, there are additional classes such as SPMD and ASIMD developed to cater with more complicated architectures. SPMD stands for single program, multiple data stream. The same program is duplicated on all the processors and the data is divided into independent blocks which are computed at their own pace. This model can be considered as a subset of the MIMD model [4]. ASIMP, on the other hand, stands for asynchronous single instruction multiple data which offers more flexibility when dealing with non-uniform algorithms. Although there are arguments over classification of certain modern systems such as hybrids, Flynn's taxonomy provides a useful starting point.

A shared memory multiprocessor computer falls under MIMD category. It comprises of several processors connected to a large shared memory by a single high

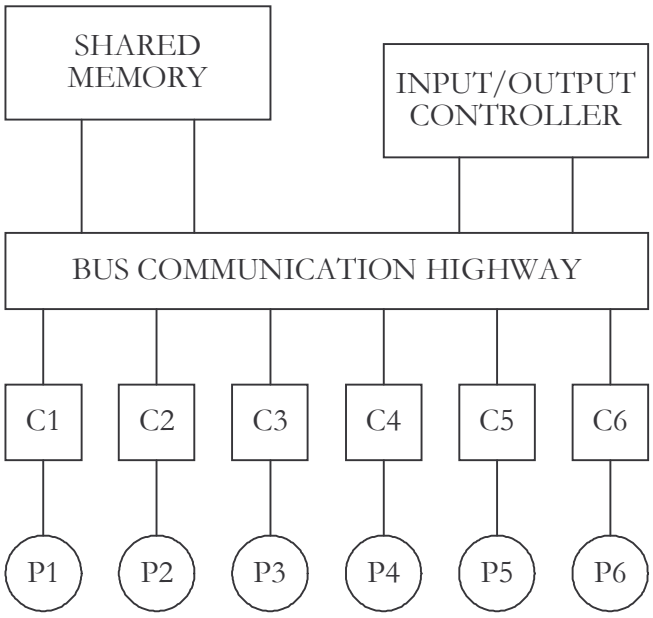

Figure 1. Shared Memory Parallel Computer.

speed bus. Figure 1 depicts a shared memory parallel computer with six processors.

To avoid congestion on the bus, each processor has a cache memory (C1,C2,…,C6) so that the processor can get frequently used data (private data) without communicating with the shared memory through the bus highway.

Normally, a shared memory parallel computer systems support two kinds of parallel programming – multiprogramming and multitasking. Multiprogramming is an operating system that allows a computer to execute multiple unrelated programs at the same time. An example of multiprogramming is a multi-user operating system where many users are allowed to run their programs concurrently. On the other hand, multitasking is performed at the software level (programming technique). Multitasking is a decomposition of a program into two or more tasks that can perform concurrently. The tasks must have no data or control dependence. So, we as programmers should focus on multitasking since we have some control over it where as multiprogramming is performed by the operating system. There are two multitasking programming methods:

a) data partitioning
b) function partitioning

Data partitioning involves identical tasks executed in parallel (homogeneous multitasking). It is appropriate for applications that perform the same operation repeatedly on a large set of data. In programming, data partitioning suits for applications that requires loops to perform on arrays or matrices where it is done by executing the loop iterations in parallel. The matrix multiplication algorithm adapts well to data partitioning as shown in [6].

Meanwhile, function partitioning involves different tasks executed in parallel (heterogeneous multitasking). It creates multiple unique processes and has them simultaneously perform different operations on a shared data (same data). Therefore, in programming, it is suitable for applications that have many unique subroutines of functions. The example of function partitioning can be found in [20]. According to Osterhaug, the data partitioning method fits more applications if compared to the function partitioning [14].

2. Designing Parallel Algorithms

Numerous problems encountered in various branches of science and engineering involve ODEs. In this paper we consider the following *d*th order ODE

$$\begin{aligned} y^{d} &= f(x, y, y', y'', \ldots, y^{d-1}), \\ y^{(i)}(a) &= \eta_i, \quad a \le x \le b \end{aligned} \tag{1}$$

Equation (1) can be solved by reducing it to the equivalent first order system and then solve it using first order ODEs methods. The disadvantage of these methods is that the system in (1) has been enlarged. The other approach is to solve (1) directly as discussed in [8], [9], [10], [11], [17] and [18]. There have been quite a number of parallel methods for solving first order ODEs as discussed in [3], [5], [16] and [19]. However, the available methods for solving (1) directly are mostly sequential.

There are two ways to parallelise the existing methods for solving higher ODES directly:

i) Parallelisation across the time
ii) Parallelisation across the method

Parallelisation Across the Time

In this method, the existing sequential codes need not be modified and redesigned. Basically we just restructure the loop statements and run the codes using a parallel compiler. The reason for restructuring the loops is because a loop's body often executes many times and it often comprises a large portion of the program's parallelism [13]. We can exploit coarse-grained parallelism by distributing entire loop iterations to different processors.

Each loop of the existing methods is carefully studied in order to identify whether or not it is independent. Parallelisation will only

be performed on independent loops that have considerably large set of data. Independent loops that contain small set of data are not parallelised because the parallel overheads will dominate the execution time. Therefore, there is no advantage of parallelisation. The task division in the loops is statically determined and therefore it is appropriate to adopt static scheduling. Data partitioning is employed because the tasks to be executed in parallel are identical.

Parallelisation Across the Method

This method is more realistic approach if compared to the previous method [4]. It requires the modification and redesign of sequential algorithms so that the new algorithms are parallel in nature and suit parallel computers well. This approach is quite challenging because not only new algorithms have to be developed, a parallel execution need also be considered. We opt to use this approach in this paper.

It is worth mentioning that the architectural aspect of a shared memory parallel computer needs to take into account while deriving the following methods. The most crucial part is to develop the methods which have no data dependencies in each equation.

3. Derivation of the Parallel 2-Point Explicit Block Method

In a 2-point block method (refer to Figure 2), the interval $[a,b]$ is divided into series of blocks with each block containing two points, i.e x_{n-1} and x_n in the first block while x_{n+1} and x_{n+2} in the second block where solutions to (1) are to be computed.

The computation which proceeds in blocks is based on the computed values at the earlier blocks. If the computed values at the previous k blocks are used to compute the current block containing r points, then the method is called *r*-point *k*-block method. For example, in Figure 3 the values used at the previous three blocks are x_{n-5}, x_{n-4}, x_{n-3}, x_{n-2}, x_{n-1}, x_n to compute the solutions of (1) at x_{n+1} and x_{n+2}. This method is known as a 2-point 3-block method.

Since the computational tasks at each point within a block are independent, it is possible to assign the tasks to different processors so that the computations can be performed simultaneously.

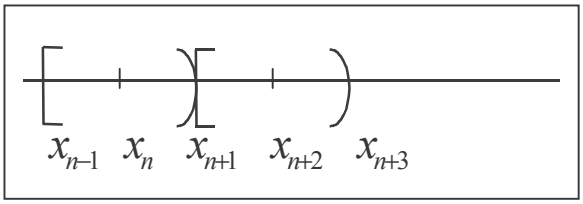

Figure 2. 2-Point Method.

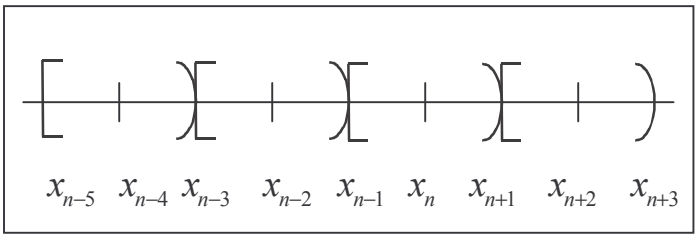

Figure 3. 2-Point 3-Block Method.

Let $x_{n+t} = x_n + th$, $t = 1,2.$ Integrating Equation (1) p times gives

$$\int_{x_n}^{x_{n+t}} \int_{x_n}^{x} \int_{x_n}^{x} \int_{x_n}^{x} \ldots \int_{x_n}^{x} y^d(x,y,y',...,y^{d-1})dx...dx = \int_{x_n}^{x_{n+t}} \int_{x_n}^{x} \int_{x_n}^{x} \int_{x_n}^{x} \ldots \int_{x_n}^{x} f(x,y,y',...,y^{d-1})\,dx...dx$$

$\leftarrow p$ times $\rightarrow$

Define $P_{k-1,n}(x)$ as the interpolation polynomial which interpolates $f(x,y,y',...,y^{d-1})$ at the k back values namely $\{x_{n-i} \mid i = 0,1,2,...,k-1\}$ as follows

$$P_{k,n}(x) = \sum_{m=0}^{k-1} (-1)^m \binom{-s}{m} \nabla^m f_n$$

where

$$s = \frac{x - x_n}{h}$$

Approximating $f(x,y,y')$ with $P_{k-1,n}(x)$, we then have

$$\begin{bmatrix} y_{n+1}^{(n-p)} \\ y_{n+2}^{(n-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(n-p)} \\ y_n^{(n-p)} \end{bmatrix} + h\begin{bmatrix} y_n^{(n-p+1)} \\ 2y_n^{(n-p+1)} \end{bmatrix} + \frac{h^2}{2!}\begin{bmatrix} y_n^{(n-p+2)} \\ 2^2 y_n^{(n-p+2)} \end{bmatrix} + \ldots \frac{h^{p-1}}{(p-1)!}\begin{bmatrix} y_n^{(n-1)} \\ 2^{p-1} y_n^{(n-1)} \end{bmatrix} + \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \tag{2}$$

where

$$A_1 = \int_{x_n}^{x_{n+1}} \int_{x_n}^{x} \int_{x_n}^{x} \int_{x_n}^{x} \ldots \int_{x_n}^{x} \sum_{m=0}^{k-1} (-1)^m \binom{-s}{m} \nabla^m f_n \, dx...dx \quad \text{and}$$

$$A_2 = \int_{x_n}^{x_{n+2}} \int_{x_n}^{x} \int_{x_n}^{x} \int_{x_n}^{x} \ldots \int_{x_n}^{x} \sum_{m=0}^{k-1} (-1)^m \binom{-s}{m} \nabla^m f_n \, dx...dx$$

Replacing $dx = hds$ and changing the limit of integration in (2) leads to

$$\begin{bmatrix} y_{n+1}^{(n-p)} \\ y_{n+2}^{(n-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(n-p)} \\ y_n^{(n-p)} \end{bmatrix} + h\begin{bmatrix} y_n^{(n-p+1)} \\ 2y_n^{(n-p+1)} \end{bmatrix} + \frac{h^2}{2!}\begin{bmatrix} y_n^{(n-p+2)} \\ 2^2 y_n^{(n-p+2)} \end{bmatrix} + \ldots \frac{h^{p-1}}{(p-1)!}\begin{bmatrix} y_n^{(n-1)} \\ 2^{p-1} y_n^{(n-1)} \end{bmatrix} + h^p \begin{bmatrix} \sum_{m=0}^{k-1} \alpha_{1,m}^{(p)} \nabla^m f_n \\ \sum_{m=0}^{k-1} \alpha_{2,m}^{(p)} \nabla^m f_n \end{bmatrix} \tag{3}$$

where

$$\alpha_{j,m}^{(p)} = (-1)^m \int_0^j \frac{(1-s)^{p-1}}{(p-1)!} \binom{-s}{m} ds \quad \text{for} \quad j = 1,2.$$

Define the generating functions $G_j^{(p)}(t)$ as follows

$$G_j^{(p)}(t) = \sum_{m=0}^{\infty} \alpha_{j,m}^{(p)} t^m \tag{4}$$

Solving (4) leads to the following relationships

$$G_j^{(p)}(t) = \frac{j^{(p-1)} - (p-1)!G_j^{(p-1)}(t)}{(p-1)!\log(1-t)} \text{ for } p = 2,3,...,d \tag{5}$$

The above relationships can easily be verified using mathematical induction. Solving (5) gives the following solution

$$\begin{aligned}
&\alpha_{1,0}^{(1)} = 1,\ \alpha_{1,m+1}^{(1)} = 1 - \sum_{r=0}^{m} \frac{\alpha_{1,r}^{(1)}}{m+2-r}\ , \\
&\alpha_{2,0}^{(1)} = 2,\ \alpha_{2,m+1}^{(1)} = (m+3) - \sum_{r=0}^{m} \frac{\alpha_{2,r}^{(1)}}{m+2-r}\ , \\
&\alpha_{1,0}^{(p)} = \alpha_{1,1}^{(p-1)}, \alpha_{1,m+1}^{(p))} = \alpha_{1,m+2}^{(p-1)} - \sum_{r=0}^{m} \frac{\alpha_{1,r}^{(p)}}{m+2-r}\ , \\
&\alpha_{2,0}^{(p)} = \alpha_{2,1}^{(p-1)}, \alpha_{2,m+1}^{(p))} = \alpha_{2,m+2}^{(p-1)} - \sum_{r=0}^{m} \frac{\alpha_{2,r}^{(p)}}{m+2-r}\ , \text{for } p = 2,3,...,d.
\end{aligned} \tag{6}$$

Note that formula (3) can be written in the form

$$\begin{aligned}
\begin{bmatrix} y_{n+1}^{(n-p)} \\ y_{n+2}^{(n-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(n-p)} \\ y_n^{(n-p)} \end{bmatrix} + h\begin{bmatrix} y_n^{(n-p+1)} \\ 2y_n^{(n-p+1)} \end{bmatrix} + \frac{h^2}{2!}\begin{bmatrix} y_n^{(n-p+2)} \\ 2^2 y_n^{(n-p+2)} \end{bmatrix} + ... \\
\frac{h^{p-1}}{(p-1)!}\begin{bmatrix} y_n^{(n-1)} \\ 2^{p-1} y_n^{(n-1)} \end{bmatrix} + h^p \begin{bmatrix} \sum_{m=0}^{k-1} \beta_{k-1,m}^{(1,p)} f_{n-m} \\ \sum_{m=0}^{k-1} \beta_{k-1,m}^{(2,p)} f_{n-m} \end{bmatrix}
\end{aligned} \tag{7}$$

where

$$\beta_{k-1,m}^{(j,p)} = (-1)^m \sum_{r=m}^{k-1} \binom{r}{m} \alpha_{j,r}^{(p)} \tag{8}$$

The equation in (7) has no data dependencies and the computation at $y_{n+1}^{(n-p)}$ and $y_{n+2}^{(n-p)}$ can, therefore, be performed simultaneously.

4. Derivation of the Parallel 3-Point Explicit Block Method

This method is the extension of parallel 2-point explicit block method as discussed earlier. Here, we attempt to find numerical solutions at three points simultaneously.

The interval $[a,b]$ is divided into series of blocks with each block containing three points, i.e x_{n-2}, x_{n-1} and x_n in the first block while x_{n+1}, x_{n+2} and x_{n+3} in the second block (refer to Figure 4) where solutions to (1) are to be computed.

Similar to the previous method, the computation which proceeds in blocks is based on the computed values at the earlier blocks. The computation at the first and

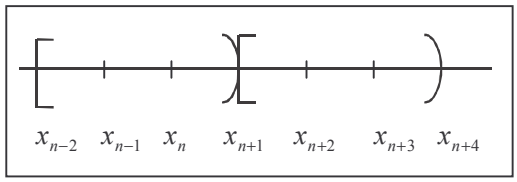

Figure 4. 3-Point Method.

second point (i.e x_{n+1} and x_{n+2}) is exactly the same as in the parallel 2-point explicit block method. In this section, we focus on determining the numerical solution at the third point. Integrating Equation (1) p times gives

$$\underbrace{\int_{x_n}^{x_{n+3}}\int_{x_n}^{x}\int_{x_n}^{x}\int_{x_n}^{x}\ldots\int_{x_n}^{x}}_{\leftarrow\, p \text{ times} \rightarrow} y^{d}(x,y,y',\ldots,y^{d-1})dx\ldots dx = \int_{x_n}^{x_{n+3}}\int_{x_n}^{x}\int_{x_n}^{x}\int_{x_n}^{x}\ldots\int_{x_n}^{x} f(x,y,y',\ldots,y^{d-1})\,dx\ldots dx \tag{9}$$

where

$$x_{n+3} = x_n + 3h$$

Define $P_{k,n}(x)$ as the interpolation polynomial which interpolates $f(x,y,y',\ldots,y^{d-1})$ at the k back values namely $\{x_{n-i} \mid i = 0,1,2,\ldots,k-1\}$ as follows

$$P_{k,n}(x) = \sum_{m=0}^{k-1}(-1)^m \binom{-s}{m} \nabla^{\mathrm{m}} f_n$$

where

$$s = \frac{x - x_n}{h}$$

Replacing $f(x,y,y',\ldots,y^{d-1})$ with $P_{k,n}(x)$ in (9) leads to

$$\begin{bmatrix} y_{n+1}^{(n-p)} \\ y_{n+2}^{(n-p)} \\ y_{n+3}^{(n-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(n-p)} \\ y_n^{(n-p)} \\ y_n^{(n-p)} \end{bmatrix} + h\begin{bmatrix} y_n^{(n-p+1)} \\ 2y_n^{(n-p+1)} \\ 3y_n^{(n-p+1)} \end{bmatrix} + \frac{h^2}{2!}\begin{bmatrix} y_n^{(n-p+2)} \\ 2^2 y_n^{(n-p+2)} \\ 3^2 y_n^{(n-p+2)} \end{bmatrix} + \ldots + \frac{h^{p-1}}{(p-1)!}\begin{bmatrix} y_n^{(n-1)} \\ 2^{p-1} y_n^{(n-1)} \\ 3^{p-1} y_n^{(n-1)} \end{bmatrix} + \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \tag{10}$$

where

$$A_3 = \int_{x_n}^{x_{n+3}}\int_{x_n}^{x}\int_{x_n}^{x}\ldots\int_{x_n}^{x}\sum_{m=0}^{k-1}(-1)^m \binom{-s}{m}\nabla^m f_n \, dx\ldots dx$$

Substituting $dx = hds$ and changing the limit of integration in (10) leads to

$$\begin{bmatrix} y_{n+1}^{(n-p)} \\ y_{n+2}^{(n-p)} \\ y_{n+3}^{(n-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(n-p)} \\ y_n^{(n-p)} \\ y_n^{(n-p)} \end{bmatrix} + h\begin{bmatrix} y_n^{(n-p+1)} \\ 2y_n^{(n-p+1)} \\ 3y_n^{(n-p+1)} \end{bmatrix} + \frac{h^2}{2!}\begin{bmatrix} y_n^{(n-p+2)} \\ 2^2 y_n^{(n-p+2)} \\ 3^2 y_n^{(n-p+2)} \end{bmatrix} + \ldots$$
$$\frac{h^{p-1}}{(p-1)!}\begin{bmatrix} y_n^{(n-1)} \\ 2^{p-1} y_n^{(n-1)} \\ 3^{p-1} y_n^{(n-1)} \end{bmatrix} + h^p \begin{bmatrix} \sum_{m=0}^{k-1}\alpha_{1,m}^{(p)}\nabla^m f_n \\ \sum_{m=0}^{k-1}\alpha_{2,m}^{(p)}\nabla^m f_n \\ \sum_{m=0}^{k-1}\alpha_{3,m}^{(p)}\nabla^m f_n \end{bmatrix} \tag{11}$$

where

$$\alpha_{3,m}^{(p)} = (-1)^m \int_0^3 \frac{(1-s)^{p-1}}{(p-1)!}\binom{-s}{m} ds$$

The generating functions $G_j^{(3)}(t)$ is defined as

$$G_3^{(p)}(t) = \sum_{m=0}^{\infty} \alpha_{3,m}^{(p)} t^m \tag{12}$$

Solving (12) gives to the following relationships

$$G_3^{(p)}(t) = \frac{3^{(p-1)} - (p-1)! G_3^{(p-1)}(t)}{(p-1)! \log(1-t)} \text{ for } p = 2,3,...,d \tag{13}$$

whose solution is

$$\begin{aligned} &\alpha_{3,0}^{(1)} = 3, \alpha_{3,m+1}^{(1)} = \frac{(m+4)(m+3)}{2} - \sum_{r=0}^{m} \frac{\alpha_{3,r}^{(1)}}{m+2-r}, \\ &\alpha_{3,0}^{(p)} = \alpha_{3,1}^{(p-1)}, \alpha_{3,m+1}^{(p))} = \alpha_{3,m+2}^{(p-1)} - \sum_{r=0}^{m} \frac{\alpha_{3,r}^{(p)}}{m+2-r} \text{ for } p = 2,3,...,d. \end{aligned} \tag{14}$$

Note that formula (11) can be written in the form

$$\begin{aligned} \begin{bmatrix} y_{n+1}^{(d-p)} \\ y_{n+2}^{(d-p)} \\ y_{n+3}^{(d-p)} \end{bmatrix} = \begin{bmatrix} y_n^{(d-p)} \\ y_n^{(d-p)} \\ y_n^{(d-p)} \end{bmatrix} + h \begin{bmatrix} y_n^{(d-p+1)} \\ 2y_n^{(d-p+1)} \\ 3y_n^{(d-p+1)} \end{bmatrix} + \frac{h^2}{2!} \begin{bmatrix} y_n^{(d-p+2)} \\ 2^2 y_n^{(d-p+2)} \\ 3^2 y_n^{(d-p+2)} \end{bmatrix} + ... \\ \frac{h^{p-1}}{(p-1)!} \begin{bmatrix} y_n^{(d-1)} \\ 2^{p-1} y_n^{(d-1)} \\ 3^{p-1} y_n^{(d-1)} \end{bmatrix} + h^p \begin{bmatrix} \sum_{m=0}^{k-1} \beta_{k-1,m}^{(1,p)} f_{n-m} \\ \sum_{m=0}^{k-1} \beta_{k-1,m}^{(2,p)} \alpha f_{n-m} \\ \sum_{m=1}^{k-1} \beta_{k-1,m}^{(3,p)} f_{n-m} \end{bmatrix} \end{aligned} \tag{15}$$

where

$$\beta_{k-1,m}^{(3,p)} = (-1)^m \sum_{r=m}^{k-1} \binom{r}{m} \alpha_{3,r}^{(p)}.$$

5. Parallel Implementation of the Methods

Initially, the explicit 1-point method(E1P) method was employed in the 2-point explicit block (2PEB) and 3-point explicit block (3PEB) methods until k backvalues (x_k) were obtained. Then depending on the numerical methods used, the numerical solution for thenext points were completed. At each step of integration, a test for checking the end point b was made. For example, in the 3PEB method a test was done to determine whether the x+3h value exceeds b. If the value did not exceed b then the 3PEB method was employed. Else, another condition was checked to find out whether the value x+2h was greater than b. The 2PEB method was employed if x+2h was smaller than b. Otherwise, a final test was made to determine whether or not the x+h value exceeds b. The E1P method was opted if b was still larger than x+h. Otherwise, the end point has been reached and the integration process was terminated. The same strategies were applied for the other two methods. For the E1P and sequential implementations of both 2PEB and 3PEB methods only one processor was used. Two and three processors were employed for

the parallel schemes of the 2PEB and 3PEB respectively. Since the tasks to be executed in parallel are identical, data partitioning was employed. The programs for E1P and the sequential implementation of the 2PEB and 3PEB methods were written in C language whereas parallel C language was used for the parallel implementation.

6. Test Problems

The following problems were tested on the Sequent Symmetry S27 using the 2-point and 3-point explicit block method.

Problem 1: $y''' = 2y'' - 4, \quad y(0) = 1,\ y'(0) = 2,\ y''(0) = 6,\ 0 \le x \le 1$
Solution: $y(x) = x^2 + e^{2x}$
Artificial Problem.
Problem 2: $y''' = 8y' - 3y - 4e^x, \quad y(0) = 2,\ y'(0) = -2,\ y''(0) = 10,\ 0 \le x \le 1$
Solution: $y(x) = e^x + e^{-3x}$
Source: [18]
Problem 3: $y^{(iv)} = (x^4 + 14x^3 + 49x^2 + 32x - 12)e^x,$
$y(0) = y'(0) = 0, y''(0) = 2, y'''(0) = -6,\ 0 \le x \le 1.$
Solution: $y(x) = x^2(1-x)^2 e^x.$
Source: [15]

7. Numerical Results

The abbreviations and notations are defined as follows:

h	Step size used.
STEPS	Total number of steps taken to obtain the solution.
MTD	Method employed.
MAXE	Magnitude of the maximum error of the computed solution
TIME	The execution time in micro-seconds needed to complete the integration in a given range using the parallel computer Sequent S27.
E1P	Explicit 1-Point method.
S2PEB	Sequential implementation of the 2-point explicit block method.
P2PEB	Parallel implementation of the 2-point explicit block method.
S3PEB	Sequential implementation of the 3-point explicit block method.
P3PEB	Parallel implementation of the 3-point explicit block method.

The maximum error is defined as follows

$$\text{MAXE} = \max_{1 \le i \le \text{STEPS}} \left(|y_i - y(x_i)| \right)$$

The comparison of the 2PEB and 3PEB methods with the E1P method for solving the test problems in terms of the total number of steps, maximum error and execution times are tabulated in Tables 1-3. Table 4 shows the ratio of steps and times of the 2PEB and 3PEB methods to E1P method. The ratios of the two parameters are obtained by dividing the parameters of the latter method with the corresponding parameters of the former methods. Hence, the ratios (also known as speedup) that are greater than one for both parameters indicate the efficiency of the 2PEB and 3PEB methods.

8. Comments on the Results and Conclusion

It is apparent from the results that the 2PEB and 3PEB methods outperform the E1P method in term of the total number of steps. As the step size becomes finer, the 2PEB and 3PEB methods reduce the number of steps to almost one half and one third respectively. These results are expected since both the 2PEB and 3PEBs method approximates the numerical solution at two and three points respectively at the same time, thus reducing the number of steps taken by the method.

Table 1. Comparison Between the E1P, 2PEB and 3PEB Methods for SolvingProblem 1.

h	MTD	STEPS	MAXE	TIME
10^{-2}	E1P	100	4.41035(-2)	122372
	S2PEB	53	8.35681(-2)	126950
	P2PEB	53	8.35681(-2)	232966
	S3PEB	37	1.75128(-1)	126945
	P3PEB	37	1.75128(-1)	274406
10^{-3}	E1P	1000	4.39119(-3)	1142734
	S2PEB	503	4.86177(-3)	1151880
	P2PEB	503	4.86177(-3)	909987
	S3PEB	337	6.28697(-3)	1123455
	P3PEB	337	6.28697(-3)	833854
10^{-4}	E1P	10000	4.38927(-4)	11430453
	S2PEB	5003	4.43709(-4)	11501547
	P2PEB	5003	4.43709(-4)	8801020
	S3PEB	3337	4.58567(-4)	11171714
	P3PEB	3337	4.58567(-4)	7986848
10^{-5}	E1P	100000	4.38908(-5)	114082053
	S2PEB	50003	4.39387(-5)	114518453
	P2PEB	50003	4.39387(-5)	87486778
	S3PEB	33337	4.40879(-5)	111392883
	P3PEB	33337	4.40879(-5)	78242291

Table 2. Comparison Between the E1P, 2PEB and 3PEB Methods for Solving Problem 2.

h	MTD	STEPS	MAXE	TIME
10^{-2}	E1P	100	1.18234(-1)	130511
	S2PEB	53	1.16165(-1)	135853
	P2PEB	53	1.16165(-1)	256096
	S3PEB	37	1.10115(-1)	135708
	P3PEB	37	1.10115(-1)	254185
10^{-3}	E1P	1000	1.17139(-2)	1223205
	S2PEB	503	1.17115(-2)	1240065
	P2PEB	503	1.17115(-2)	1034676
	S3PEB	337	1.17042(-2)	1209884
	P3PEB	337	1.17042(-2)	913258
10^{-4}	E1P	10000	1.17030(-3)	12235538
	S2PEB	5003	1.17029(-3)	12371365
	P2PEB	5003	1.17029(-3)	10051687
	S3PEB	3337	1.17029(-3)	12036763
	P3PEB	3337	1.17029(-3)	8798665
10^{-5}	E1P	100000	1.17019(-4)	122141796
	S2PEB	50003	1.17019(-4)	123347394
	P2PEB	50003	1.17019(-4)	99941712
	S3PEB	33337	1.17019(-4)	120063725
	P3PEB	33337	1.17019(-4)	87246831

Table 3. Comparison Between the E1P, 2PEB and 3PEB Methods for Solving Problem 3.

h	MTD	STEPS	MAXE	TIME
10^{-2}	E1P	100	1.00778(-2)	210474
	S2PEB	53	1.00778(-2)	222259
	P2PEB	53	1.00778(-2)	296038
	S3PEB	37	1.00779(-2)	201983
	P3PEB	37	1.00779(-2)	325724
10^{-3}	E1P	1000	1.00078(-3)	2006247
	S2PEB	503	1.00078(-3)	2076058
	P2PEB	503	1.00078(-3)	1623001
	S3PEB	337	1.00078(-3)	1850179
	P3PEB	337	1.00078(-3)	1583939
10^{-4}	E1P	10000	1.00008(-4)	20077275
	S2PEB	5003	1.00008(-4)	20727420
	P2PEB	5003	1.00008(-4)	15922113
	S3PEB	3337	1.00008(-4)	18504213
	P3PEB	3337	1.00008(-4)	15212065
10^{-5}	E1P	100000	1.00001(-5)	200307659
	S2PEB	50003	1.00001(-5)	206611648
	P2PEB	50003	1.00001(-5)	158493054
	S3PEB	33337	1.00001(-5)	184047416
	P3PEB	33337	1.00001(-5)	151403794

Table 4. The Ratio Steps and Execution Times of the 2PEB and 3PEB Methods to the E1P Method for Solving Higher Order ODEs.

TOL	MTD	RATIO STEP	RATIO PROB.1	TIME PROB.2	PROB.3
10^{-2}	S2PEB	1.88679	0.96394	0.96068	0.94698
	P2PEB	1.88679	0.52528	0.50962	0.71097
	S3PEB	2.70270	0.96398	0.96171	1.04204
	P3PEB	2.70270	0.44595	0.51345	0.64617
10^{-3}	S2PEB	1.98807	0.99206	0.98640	0.96637
	P2PEB	1.98807	1.25577	1.18221	1.23613
	S3PEB	2.96736	1.01716	1.01101	1.08435
	P3PEB	2.96736	1.37043	1.33939	1.26662
10^{-4}	S2PEB	1.99880	0.99382	0.98902	0.96863
	P2PEB	1.99880	1.29877	1.21726	1.26097
	S3PEB	2.99670	1.02316	1.01651	1.08501
	P3PEB	2.99670	1.43116	1.39061	1.31983
10^{-5}	S2PEB	1.99988	0.99619	0.99023	0.96949
	P2PEB	1.99988	1.30399	1.22213	1.26383
	S3PEB	2.99967	1.02414	1.01731	1.08835
	P3PEB	2.99967	1.45806	1.39996	1.32300

In term of accuracy, all methods have the same order of accuracy.

As expected, the execution times taken by the parallel implementation of the 2PEB and 3PEB methods are more than those taken by the sequential counterpart and the E1P method at $h = 10^{-2}$. This is because the number of steps taken is small and most of the execution times are dominated by the parallel overheads. However, the timings of the parallel version of the 2PEB and 3PEB methods are better then other methods when $h < 10^{-2}$. The reason for these gains is that as the step size gets smaller, more steps are taken to complete the computation. By using 2 or 3 processors instead of 1, the computation can be performed quicker. In other words, the parallelism in the new methods could really be exploited. The results also suggest that the new parallel methods are recommended for solving second order ODEs directly using finer step sizes.

I strongly believe that the new methods can also be implemented in the distributed parallel computer architecture with few modifications particularly in the programming aspects.

REFERENCES

[1] Abdullah R., Design and Analysis of Numerical Algorithms for the Solution of Linear Systems on Parallel and Distributed Architectures, Ph.D. Thesis, Loughborough University of Technology, United Kingdom, 1997.

[2] Akl S.G., *The Design and Analysis of Parallel Algorithms*, Englewood Cliffs: Prentice Hall, Inc., 1989.

[3] Birta L.G., and Abou-Rabia O., Parallel Block Predictor-Corrector Methods for ODEs, *IEEE Transactions on Computers*, 1987; **C-36 (1)**: 299-311.

[4] Burrage K., *Parallel and Sequential Methods for Ordinary Differential Equations*, New York: Oxford University Press Inc., 1995.

[5] Chu M.T. and Hamilton H., Parallel Solution of ODEs by Multi-Block Methods, *Siam J. Sci. Stat. Comput.*, 1987; **8(1)**: 342-353.

[6] Evans D.J. and Omar Z.B., The Parallel Implementation of Matrix Multiplication, Report No. 948, Department of Computer Studies, Loughborough University of Technology, United Kingdom, 1994.

[7] Flynn M.J, Some Computer Organizations and Their Effectivenes, *IEEE Transactions on Computers,* 1972; **C-21(9)**: 948-960.

[8] Gear C.W., The Numerical Integration of Ordinary Differential Equations, *Math. Comp.*, 1966; **21**: 146-156.

[9] Gear C.W., *Numerical Initial Value Problems in Ordinary Differential Equations*, New Jersey: Prentice Hall, 1971.

[10] Gear C.W., The Stability of Numerical Methods for Second-Order Ordinary Differential Equations, *SIAM J. Numer. Anal.*, 1978; **15(1)**: 118-197.

[11] Hall G., and Suleiman M.B., Stability of Adams-Type Formulae for Second-Order Ordinary Differential Equations. *IMA J. Numer. Anal.*, 1981; **1**: 427-428.

[12] Lewis T.G., and El-Rewini H., *Introduction to Parallel Computing*, Englewood Cliffs, New Jersey: Prentice Hall International Edition, 1992.

[13] Lilja D.J., Exploiting the Parallelism Available in Loops, *IEEE Computer,* 1994; **27(2)**: 13-26.

[14] Osterhaug A., *Guide to Parallel Programming on Sequent Computer Systems*, Oregon: Sequent Computer Systems, Inc., 1987.

[15] Russel R.D., and Shampine L.F., A Collocation Method for Boundary Value Problems, *Num. Math,* 1972; **19**: 1-28.

[16] Shampine L.F., and Watts H.A., Block implicit one-step methods*, Math. Comp.*, 1969; **23**: 731-740.

[17] Suleiman M.B., *Generalised Muiltistep Adams and Backward Differentiation Methods for the Solution of Stiff and Non-Stiff Ordinary Differential Equations,* Ph.D. Thesis, University of Manchester, United Kingdom, 1979.

[18] Suleiman M.B., Solving Higher Order

ODEs Directly by the Direct Integration Method, *Applied Math. and Computation*, 1989; **33**: 197-219.

[19] Tam H.W., Parallel Methods For The Numerical Solution Of Ordinary Differential Equations, Report No. UIUCDCS -R-89-1516, Department of Computer Science, University of Illinois at Urbana-Champaign, United States, 1989.

[20] Yaakub A.R., Computer Solution of Non-Linear Integration Formula for Solving Initial Values Problems, Ph.D. Thesis, Loughborough University of Technology, United Kingdom, 1996.