

Parallel Block Methods for Solving IVPs in ODEs

Bashir M. Khalaf & Abdulhabib Abdullah

Education College

Mosul University

الملخص

الغرض الرئيسي من هذا البحث هو تطوير الخوارزميات العددية المتوازية لحل المعادلات التفاضلية الاعتيادية الصلبة والتي هي مناسبة للتنفيذ على الحاسبات من نوع MIMD .

Abstract

The main purpose of this research is to develop parallel numerical algorithms for solution of stiff ordinary differential equation which are suitable for running on MIMD computers.

1:Introduction

A block is the set of all new function values which are evaluated during each application of the iteration formula. For a k-points block k new values of the solution are produced simultaneously in each computational step. Thus, a block method generates a set, or block, of new values in a single integration step.

Block methods appear to have been first proposed by Milne (1953) who advanced their use only as a means of obtaining starting values for predictor-corrector algorithms for general use.

Several authors (see, for example, [1,2,3,5,8,10,11,12,16,20]) have considered block methods for the parallel solution of the initial value problem (IVP).

$$y' = f(x, y), y(x_0) = y_0 \quad (1)$$

By means of a single application of a calculation unit, a block method yields a sequence of new estimates for y. If $k \geq 1$ is the block size, then in simple cases the values of x at which solution are computed will be evenly separated [19]. In other words, each basic cycle of the calculation has the potential to advance the solution by k new points in the x direction. Each such block can, therefore, be considered as a unit of calculation. Let y_n denotes the approximation to the exact solution $y(x_n)$ at $x = x_n$. Also, f_n denotes the value of $f(x_n, y_n)$, the approximation for $y'(x_n)$. For $n = m k$, a block of solution can be represented by the vector

$Y_m = (y_{n+1}, y_{n+2}, \dots, y_{n+k})^T$ with y_{n+j} ($1 \leq j \leq k$), the generated solution at $x_{n+j} = x_n + jh$, where x_n is the right-hand end point of the preceding block and is the uniform spacing between solution values.

Such procedures can be formulated either as implicit predictor-corrector methods [16]. In addition the underlying formulae may only refer to the end point of the previous block, so called one-step methods. In other words, by one-step methods, we mean methods that compute the block of values y_{n+i} , $i=1, \dots, k$, from the value of y_n only. Otherwise, some or all of the points in the previous block could be used (multi-step methods) or a number of previous blocks in which case the methods are referred to as multi-block methods.

2: Cash's Block Method For Nonstiff ODEs:

Cash [3] derived a block Runge-Kutta formula suitable for the approximate numerical integration of non stiff initial value problems for first order systems of ordinary differential equations. The central idea of Cash's formula is similar to one due to C.W. Gear [6] in developing Runge-Kutta starters for linear multi step methods. So that Cash's paper can be regarded as complementary to the paper by Gear. Cash lists some block integration formulae of order 1-4, and he leaves the derivation of higher order formulae as an area for future research and they can be considered as challenging problems. The third order formula, for example, derived by Cash has following form:-

0	0						
1/3	1/3	0					
2/9	4/27	2/27	0				
	1/12	0	1/4				
2/3	0	2/3	0				
4/9	8/81	-8/81	10/27	2/27	0		(2)
	9/96	0	21/96	7/96	27/96		
1	a_1	a_2	a_3	a_4	a_5	0	
	35/504	0	39/112	23/48	0	13/126	

Where $a_1=0.9173076923$, $a_2=-2.807692308$,
 $a_3=0.3923076923$, $a_4=1.073076923$, $a_5=1.425$,

So that

$$y_{n+1} - y_n = h\{1/4k_1 + 3/4k_3\}$$

$$y_{n+2} - y_n = h\{9/32k_1 + 21/32k_3 + 7/32k_4 + 27/32k_5\}$$

$$y_{n+3} - y_n = h\{105/504k_1 + 117/112k_3 + 69/48k_4 + 39/126k_6\}$$

The formula is obtained by using the standard RK formalism for block methods with the coefficients of the Butcher array for a P^{th} order formula given for the stepsize $H=ph$, and the weights for solution at internal points in the block are given under the dotted lines.

3: Cash's Block Formula for Stiff ODEs:

Cash [4] extends the algorithms described in [3] to include diagonally implicit Runge-Kutta (DIRK) formulae, to be suitable for the numerical integration of stiff differential systems, depending on the paper of Bond & Cash that derived a class of block cyclic integration formulae. Consider the second order implicit integration formula :

$$\begin{aligned} y_{n+i}^{(1)} &= y_{n+i-1}^{(1)} + hf(x_{n+i}, y_{n+i}^{(1)}), & i &= 1,2 & y_n^{(1)} &\equiv y_n, \\ y_{n+i}^{(2)} &= y_{n+i-1}^{(2)} + hf(x_{n+i}, y_{n+i}^{(2)}) - \frac{1}{2} \Delta_h^2 y_n^{(1)} & i &= 1,2,3 & y_n^{(2)} &\equiv y_n \end{aligned} \quad (3)$$

Where $\Delta_h y_n^{(1)} = y_{n+1}^{(1)} - y_n^{(1)}$. By using the well-known "Butcher Matrix Notation (3) can be represented in the form:

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

Using this notation the formula (3) can be written for the computation of $y_{n+3}^{(2)}$ as:

$$\begin{array}{c|cccccc} 1 & 1 & & & & & \\ 2 & 1 & & 1 & & & \\ 1 & 1/2 & & -1/2 & 1 & & \\ 2 & 1 & & -1 & 1 & 1 & \\ 3 & 3/2 & & -3/2 & 1 & 1 & 1 \\ \hline & 3/2 & & -3/2 & 1 & 1 & 1 \end{array} \quad (4)$$

This formula has important property that as well as providing a second order solution at x_{n+3} it also provides both first and second order solution at x_{n+2} . Therefore, the formulae used to compute $y_{n+1}^{(1)}, y_{n+2}^{(1)}, y_{n+1}^{(2)}, y_{n+2}^{(2)}, y_{n+3}^{(2)}$ are respectively:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \quad \begin{array}{c|cc} 1 & 1 & \\ 2 & 1 & 1 \\ \hline & 1 & 1 \end{array} \quad \begin{array}{c|ccc} 1 & 1 & \\ 2 & 1 & 1 \\ 1 & 1/2 & -1/2 & 1 \\ \hline & 1/2 & -1/2 & 1 \end{array}$$

$$\begin{array}{c|cccc}
1 & 1 & & & \\
2 & 1 & 1 & & \\
1 & 1/2 & -1/2 & 1 & \\
2 & 1 & -1 & 1 & 1 \\
\hline
& 1 & -1 & 1 & 1
\end{array} \quad \text{and (4)}$$

We note that five stage block formula (4) provides 5 separate solution.

4: Parallel Block Methods:

Block methods were around well before parallel computation. Milne in 1953 as we have seen before discussed method that generates four new values in a single step, he actually proposed it for generating starting values for a multi-step method. Rosser [15] started with this method and modified it to be more efficient as a computational procedure for step-by-step integration rather than as a starting method for a multistep method. It is useful to examine the original method proposed by Milne in a more succinct notation [7]:

$$y_i = y_0 + h \sum_{j=0}^4 \beta_{ij} f(x_j, y_j), \quad i=1, \dots, 4 \quad (5)$$

Where the stepsize $h_n = x_{n+1} - x_n$ is assumed to be constant. Since these equations are implicit in y, Milne handled them by functional iteration, using

$$y_i^{(m+1)} = y_0 + h \sum_{j=0}^4 \beta_{ij} f(x_j, y_j^{(m)}), \quad i=1, \dots, 4, m=0, 1, \quad (6)$$

The values of $y_i^{(0)}$ are calculated by use of the forward Euler formula separately over step of h, 2h, 3h, and 4h, which is exactly the same as applying equation (6) for m= -1 with $f(x_i, y_i^{(-1)}) = f(x_0, y_0)$. The parallelism inherent in equation (5) is obvious: a separate processor can be assigned for each I, so this system can use four processors.

This idea has been carried further by Shampine and watts [17] who consider block implicit methods techniques for their implementation, and then for parallel processing by Franklin [5] and Birta and Abou-Rabia [2] who use one-step block corrector methods coupled with multi-block predictor formulas, and Chu and Hamilton who use multi –block

methods. Other researchers also discussed parallel block methods such as Worland [20], Katz et al [10], Kerckhoffs [11], Khalaf [12] and Sommeijer et al [18].

In predictor – corrector block methods the value of the unknown vector is computed ahead simultaneously at a predetermined number of future points; the computation is based on the computed values of the vector at earlier points. The computation proceeds in blocks. Within a block it is possible to assign both the predictor and the corrector computation at each future point to a single computer, and to perform the computation at all future points simultaneously in parallel. Similarly, Bickart formulate composite multistep methods which are A-stable, thereby making them suitable for the numerical solution of stiff system of differential equations. These methods are also of block type and the parallelism can be used as before. On the other hand, in the method presented by Miranker and Linger [14] for example, parallelism is achieved in a different way. It is assumed in [14] that the predictor-corrector algorithm operates in a PECE mode (one predicted derivative evaluation and one corrected derivative evaluation) and that the processor performs either a predictor or a corrector calculation. New formulas are developed in which the corrector does not depend serially upon the predictor, so that the predictor and the corrector calculations can be performed simultaneously.

4.1: Parallel Block Implicit Methods:

Block implicit methods as described by Shampine and Watts [16] and by Rosser [15] have been shown to be competitive with standard methods for integrating ODEs. Worland [20] showed that block methods are good candidates for parallel processor implementation and are easily adapted to a parallel mode with little apparent degradation in the solution.

In block implicit methods, time is divided into a series of blocks with each block containing a number of steps at which solutions to system equations are to be found [5]. Block values are all obtained together in a single block advance and the block may be considered as unit calculation. The accuracy of the method can be changed by changing the number of steps in a block or the size of the steps. These changes can be made dynamically at the start of each block calculation on the basis of error condition occurring in the previous block.

In a k-point block method each pass through the algorithm simultaneously produces k new equally spaces solution values.

Block implicit methods can be applied in a one- step mode, in which only the last point in the block is used to compute the first approximation to the k values of the next block. Then, implicit formulas

are applied iteratively until convergence is achieved to the maximum order of accuracy obtainable [16].

An example of a parallel 4- point one – step implicit block scheme, based upon integration formulas which are basically of the Newton-Cotes type, is (see Worland (1976)[20]):

$$y_{n+r}^{(0)} = y_n + rhf_n \quad , \quad r = 1, 2, 3, 4$$

$$y_{n+1}^{(s+1)} = y_n + \frac{h}{720}(251f_n + 646f_{n+1}^{(s)} - 264f_{n+2}^{(s)} + 106f_{n+3}^{(s)} - 19f_{n+4}^{(s)})$$

$$y_{n+2}^{(s+1)} = y_n + \frac{h}{90}(29f_n + 124f_{n+1}^{(s)} - 24f_{n+2}^{(s)} + 4f_{n+3}^{(s)} - f_{n+4}^{(s)})$$

$$y_{n+3}^{(s+1)} = y_n + \frac{3h}{80}(9f_n + 34f_{n+1}^{(s)} - 24f_{n+2}^{(s)} + 14f_{n+3}^{(s)} - f_{n+4}^{(s)})$$

$$y_{n+4}^{(s+1)} = y_n + \frac{2h}{45}(7f_n + 32f_{n+1}^{(s)} - 12f_{n+2}^{(s)} + 32f_{n+3}^{(s)} - 7f_{n+4}^{(s)})$$

Where $s = 0, 1, \dots, S$ is the iterations number, r indicates a node in a block, $y_i = y_i^{(s+1)}$, $f_i = f(x_i, y_i^{(s+1)})$ and $f_i^{(s)} = f(x_i, y_i^{(s)})$.

However, on a parallel machine y_{n+r} and f_{n+r} are computed on processor r , and thus they are calculated simultaneously for different r .

Block implicit methods can also be adapted to a predictor – corrector mode, as we shall see later; in this case the solution values of a block may be used to predict a solution at each node of the next block.

4.2: Parallel Block One- Step Method:-

In the sequential one – step approach, a sequence of open and closed Newton – Cotes formulas is used to generate initial approximate to the solution $y(x)$ at each point in the current block. Then implicit formulas are applied iteratively until convergence is obtained to the maximum order of accuracy obtainable. Only the solution at the last point in the block is used to start the next block.

The modifications of these algorithms to obtain effective schemes for parallel processor are straightforward. In the sequential approach, the sequence of equations is applied in a "Gauss-Seidel" sense, each newly computed approximation being used in the next formula in sequence. On parallel machine these equation must be applied in the "Jacobi" sense [20].

Lets us take a block which consists of k equidistant points $x_{n+r} = x_n + rh$, $r=1, \dots, k$ where h is the step size, and kh represents the block length. And let y_n represents the approximate solution of a given

first order differential equation at x_n the initial point of the current block, and $y(x_n)$ is the exact solution at x_n .

The formulas for sequential two- point block scheme are:-

$$y_{n+1}^{(0)} = y_n + hf_n \quad (7)$$

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}(f_n + f_{n+1}^{(0)}) \quad (8)$$

$$y_{n+2}^{(1)} = y_n + 2hf_{n+1}^{(1)} \quad (9)$$

$$y_{n+1}^{(2)} = y_n + \frac{h}{3}(5f_n + 8f_{n+1}^{(1)} - f_{n+2}^{(1)}) \quad (10)$$

$$y_{n+2}^{(s+1)} = y_n + \frac{h}{3}(f_n + 4f_{n+1}^{(s)} - f_{n+2}^{(s)}), \quad s = 1, (2) \quad (11)$$

The sequence (7) – (11), as given by Rosser [15], is used minimize the number of derivative evaluation required.

The corresponding formulas for the parallel are:

$$y_{n+r}^{(0)} = y_n + rhf_n, \quad r = 1, 2$$

$$y_{n+1}^{(s+1)} = y_n + \frac{h}{12}(5f_n + 8f_{n+1}^{(s)} - f_{n+2}^{(s)})$$

$$y_{n+1}^{(s+1)} = y_n + \frac{h}{3}(f_n + 4f_{n+1}^{(s)} - f_{n+2}^{(s)})$$

Where $s = 0, 1, 2, (3)$. On parallel machine $y_{n+1}^{(s+1)}$ and $y_{n+2}^{(s+1)}$ are obtained simultaneously for each s .

A similar algorithm for the four point block can be used. In general, three steps of the one step block methods can be given:

- (1) each point of the block methods can be given:
- (2) approximation for the solution is evaluate by Euler's method.
- (3) Values are improved by Newton-Cotes $k+1$ points formula

4.3: Parallel Block Predictor- Corrector (PBPC) Methods:-

Block implicit methods can also be adapted to a PC mode; in this case all the solution values of the block may be used to predict a solution at each node of the next block. Block implicit PC scheme have been given for example in [20] and [5]. The block predictor- corrector (BPC) method provides a procedure for conveniently distributing over a set of processors, the computational workload involved in generating the numerical solution of a set of ordinary differential equations.

The procedure is inherently parallel in the sense that the workload distribution occurs without any need for partitioning of the given equations [2]. The term "block" in the BPC method refers to the set of

new solution values that are produced with each execution of the formulas associated with the method. When the method is used in its basic form, a k-processor configuration is associated with a block having k points and each pass through the formulas, which are distributed over the k processors, simultaneously produces the k new solution values within the block. Correspondingly, the solution is advanced by kh where h is the spacing between the solution values within the block.

As an example consider the two point case block PC method. This is a process of two parallel prediction following by two parallel correction:

- (1) first, the initial values of the solution have to be computed or known.
- (2) Predictor formulae are used to calculate new values (in this case two values).
- (3) Corrector formulae are used iteratively.

This procedure can be easily applied using two processors, where each processor is allocated to doing both predictor and corrector calculations for one point of the block.

To make it clear consider a fourth order block PC for the numerical integration of a system of a set of ODEs, presented by Shampine and Watts [16] is as follows:

The predictor equations are:

$$Y_{n+1}^p = \frac{1}{3}(Y_{n-2}^c + Y_{n-1}^c + Y_n^c) + \frac{h}{6}(3F_{n-2}^c - 4F_{n-1}^c + 13F_n^c), \quad (12)$$

$$Y_2 = \frac{1}{3}(Y_{n-2}^c + Y_{n-1}^c + Y_n^c) + \frac{h}{12}(29F_{n-2}^c - 72F_{n-1}^c + 79F_n^c), \quad (13)$$

The corrector equations are:

$$Y_{n+1}^c = Y_n^c + \frac{h}{12}(5F_n^c + 8F_{n+1}^p - F_{n+2}^p) \quad (14)$$

$$Y_{n+2}^c = Y_n^c + \frac{h}{3}(F_n^c + 4F_{n+1}^p + F_{n+2}^p) \quad (15)$$

In this case, each block consists of two steps, n+1 and n+2. The predictor equations are dependent on values taken from the previous block (n, n-1, n-2). The corrector equations depend on a single value from the previous block (n) and the predicted values of the current block (n+1, n+2). The first predictor is used to set up the first corrector; then the second predictor can be computed to set up the second corrector. The equations within each part are independent of each other even though they refer to successive time steps. Thus the equation can be easily mapped onto a two-processor system where one processor is devoted to point n+1 and the other to point n+2. The processors have to exchange

information twice per block, once after the predictor equations and corresponding function evaluations have been evaluated, and once after the corrector equation and corresponding function evaluation have been evaluated. Solution of each block, however, provides two Y values, where the function evaluation and the Y variable calculation for these two are performed in parallel. This effectively halves the time required for function evaluation and Y variable calculations [5].

A timing diagram for a single block which corresponds to a two processors implementation of (12) through (15) is given in the following figure:

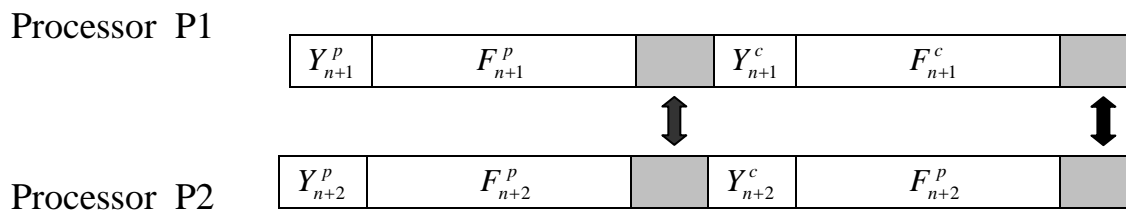

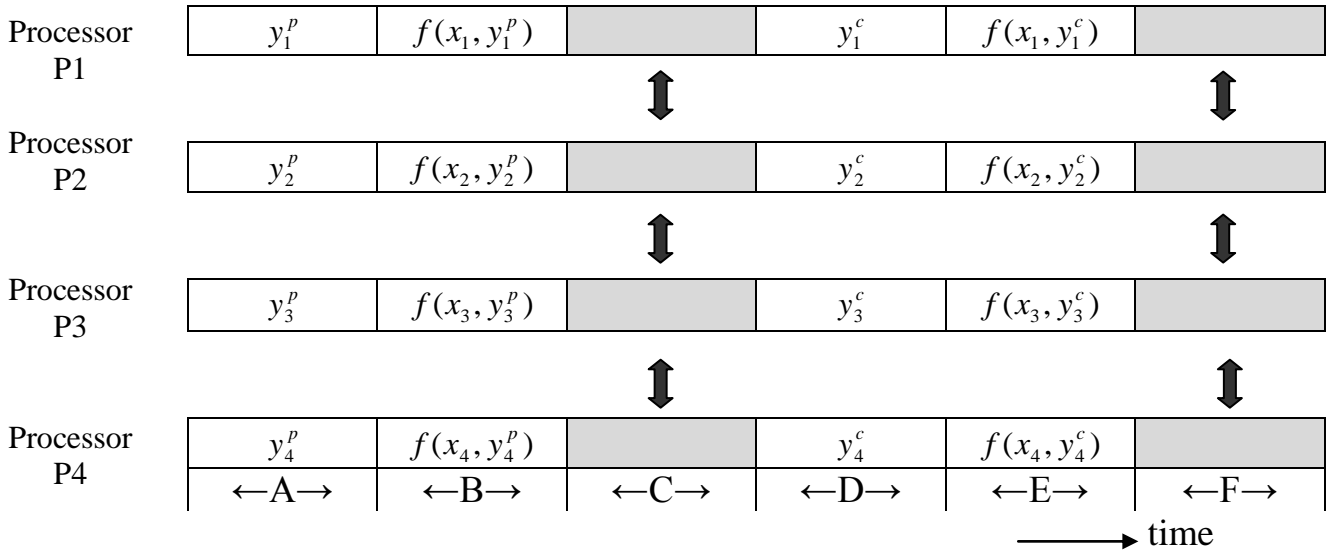


Fig. (1) : timing for PBPC integration algorithm  Data are being exchanged.

Each processor in the figure performs both the predictor and corrector evaluations associated with a single point, processor 1 point $n+1$ and processor2 point $n+2$.

Franklin [5] says that if more processors are available, then it is possible to increase the number of steps per block and change the block size such that each processor is allocated to one or more points per block. In general, the method's parallelism guarantees that the function evaluation times will effectively decrease directly as the number of processors used increases. The following figure (2) shows the timing diagram for the case $k = 4$



- A: evaluate the predictor formulas.
- B: evaluate the derivative function.
- C: exchange derivative values.
- D: evaluate the corrector formulas.
- E: evaluate the derivative function.
- F: exchange derivative values.

Fig.(2): Timing diagram for four processor case.

We note that with PCBC algorithm in the two processor system, processors are assigned doing to both predictor and corrector calculations, but on successive step, with PPC algorithm, processors are assigned to doing only predictor or corrector calculations but on a single step.

4.4: Parallel Block Methods – Fixed Order and Fixed Length:

Adopting the notation used by Birta and Abou-Rabia [2], the formula of the block method can be expressed as:

$$Y_m = ey_n + hdf_n + hBF(Y_m) \quad (16)$$

Where e and k-vectors, B is a $k \times k$ matrix, and F is a k-factor whose j^{th} entry is $f_{n+j} = f(x_{n+j}, y_{n+j})$, $1 \leq j \leq k$. As (5.16) is implicit in y_m it has to be solved iteratively using, in the first instance, predicted solution values. A predictor equation for Y can be expressed in the form:

$$Y_m^{(0)} = ey_n + h\tilde{d}f_n, \quad (17)$$

Where \tilde{d} is k-vector. Substitution of $Y_m^{(0)}$ into the right-hand side of (5.16) yields the block predictor-corrector (BPC) method:

$$Y_m = ey_n + hdf_n + hBF(ey_n + h\tilde{d}f_n) \quad (18)$$

We can write (18) in the form:

$$Y_m = ey_n + hdf_n + AY_m^{(0)} + hBf_n \quad (19)$$

Where A is $k \times k$ matrix.

In accordance with the terminology used in the linear multistep case, this application is called PEC mode. Of course, one can continue this process by substituting the result of (18) into the right-hand side of (16) arriving at $P(EC)^\nu E^{1-\nu}$ mode, in which $\nu=0$ indicates that a final evaluation is done before proceeding to the next block. Abbas and Devles [19] considered this approach using an explicit Euler predictor and then corrected twice by a trapezoidal corrector applied in the composition case. This method can be computed in three steps for each equidistant step point $r = 1, \dots, k$ as:

$$\begin{aligned} Y_{n+r}^{(0)} &= y_n + rhf(x_n, y_n), \\ Y_{n+r}^{(1)} &= y_n + h/2f(x_n, y_n) + h \sum_{i=1}^{r-1} f(x_{n+i}, y_{n+i}^{(0)}) + h/2f(x_{n+r}, y_{n+r}^{(0)}), \\ Y_{n+r}^{(2)} &= y_n + h/2f(x_n, y_n) + h \sum_{i=1}^{r-1} f(x_{n+i}, y_{n+i}^{(1)}) + h/2f(x_{n+r}, y_{n+r}^{(1)}). \end{aligned} \quad (20)$$

With $Y_m^{(0)}$ by (17), method (20) has $P(EC)^2$ from

$$\begin{aligned} Y_m^{(0)} &= ey_n + h\tilde{d}f_n, \\ Y_m^{(s+1)} &= ey_n + hdf_n + hBF(Y_m^{(s)}), \quad s = 0, 1 \end{aligned}$$

Where

$$e = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \tilde{d} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ k \end{bmatrix}, \quad d = \begin{bmatrix} 1/2 \\ 1/2 \\ \vdots \\ 1/2 \end{bmatrix}, \quad B = \begin{bmatrix} 1/2 & 0 & \dots & 0 \\ 1 & 1/2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & \dots & 1 & 1/2 & 0 \\ 1 & \dots & 1 & 1 & 1/2 \end{bmatrix}$$

As we have seen that the simplest initial estimate for elements of Y_m is obtained by the one step Euler formula as in equation (17). following an application of equation (19) or (17), equation (16) may be applied iteratively through:

$$Y_m^{(i+1)} = ey_n + hdf_n + hBF_m^{(i)} \quad (21)$$

Where $Y_m^{(0)} = Y_m$ and $F_m^{(i)}$ is a k -vector holds the derivatives $f(x_n + jh, y_j^p)$, $j = 1, 2, \dots, k$.

Thus by splitting up the F_m^P vector and matrix A over 2 processors with both processors knowing $h/2 f_0$ and processor 2 knowing estimate of $\sum_{r=1, \dots, k/2} f_r$ then estimates for $Y_r^{(i)}$, $r=1, \dots, k/2$ and $Y_s^{(i)}$, $s=k/2+1, \dots, k$ can be obtained in parallel.

It should now be apparent that A can be split across any number of processors. In general for a P processors where $k/P = q$ we follow the procedure of splitting up as in equation (30), i.e

$$Y_m^{(i+1)} = ey_0 + hA_0 F_{m0}^P + hA_1 F_{m1}^P + hB_2 G_{m2}^P \quad (31)$$

Where $F_{m0}^P = [f_0, 0, 0, \dots, 0]^T$, $F_{m1}^P = [0, f_1, f_2, \dots, f_q, 0, \dots, 0]^T$ and

$G_{m2}^P = [0, 0, \dots, 0, f_{q+1}, f_{q+2}, \dots, f_k]^T$ are $(k+1) -$ vectors.

The splitting up continue with B_2 :

$$B_2 = \begin{pmatrix} 0 & & & & & & 0 \\ & 1/2 & & & & & \\ & 1 & 1/2 & & & & \\ & 1 & 1 & 1/2 & & & \\ 0 & \vdots & \vdots & \vdots & & & 0 \\ & 1 & 1 & 1 & \dots & 1/2 & \\ \hline & 1 & 1 & 1 & \dots & 1 & \\ 0 & \vdots & & & \dots & \vdots & 0 \\ & 1 & 1 & 1 & & 1 & \\ & 1 & 1 & 1 & \dots & 1 & \end{pmatrix} + \begin{pmatrix} 0 & 0 & & & & & 0 \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & 0 & 0 & & & & 0 \\ \hline & & & & & & \\ 0 & 0 & & & 1/2 & & \\ & & & & \vdots & & \\ & & & & 1 & \dots & 1/2 \\ & & & & 1 & \dots & 1 & 1/2 \end{pmatrix} \quad (32)$$

$$\text{Or } B_2 = A_2 + B_3 \quad (33)$$

Then $F_{m2}^P = [0, \dots, 0, f_{q+1}, f_{q+2}, \dots, f_{2q}, 0, \dots, 0]^T$,

And $G_{m2}^P = [0, \dots, 0, f_{2q+1}, f_{2q+2}, \dots, f_k]^T$

Similarly we split B_3 , into A_3 and B_4 , and continue until finally:

But as we have calculated $y_j^{(i)}$ before we start to calculate $y_{j+1}^{(i)}$ we could find f_j^p and use this equation (39) to find an update solution for y_{j+1} . Then for all points of a sub-block except the first one, the equation for $y_r^{(i)}$ ($r \neq 1$) is:

$$y_r^{(i)} = y_{r-1}^{(i)} + h/2(f_{r-1}^i + f_r^p) \quad (40)$$

4.6: Parallel Block Methods For Stiff Equations:

Many algorithms for numerically solving initial value problems for ordinary differential equations (ODEs):

$$y' = f(x, y(x)), y(x_0) = y_0 \quad (41)$$

Are based on implicit linear multi step methods (LM methods), in particular on Backward Differentiation methods (BDF methods). The main reason for their popularity is the relatively low computational effort per step, at least when compared with other suitable methods for stiff equations, such as implicit Runge-Kutta methods. However, the BDFs have one serious disadvantage: they are subject to the so-called "second Dahlquist barrier", which says that the order cannot exceed two if the method has to be A-stable. Thus the higher-order BDFs lack the property of A-stability. This means that if a higher-order formula is selected (dictated by accuracy considerations), then it may happen that – for certain types of stiff ODEs the algorithm encounters stability problems which usually result in a dramatic degradation of the performance. To circumvent this behavior it is highly desirable to have A-stable methods of high order without increasing the computational effort per step.

It is our aim to construct block methods. Block methods can be considered as a set of simultaneously applied linear multi step methods to obtain several numerical approximations within one application.

Initially, the block methods were introduced to circumvent the restriction that applies to linear multi step methods: the limitation on the order because of zero-stability (known as the "first Dahlquist barrier") and the order-restriction with respect to A-stability (which is usually called "Dahlquist's second barrier"). Both restrictions can be avoided by changing from the linear multi step methods to the block methods. Moreover, parallelism can be achieved in a very natural way.

For the numerical integration of stiff ODEs, a method should preferably:

- (i) be A-stable, and
- (ii) have a high order of accuracy.

However, it is well known that these are conflicting demands for linear multi step methods (this is the so-called 'second Dahlquist barrier').

One possible way to achieve the goals (i) and (ii) is to consider implicit block methods.

4.7: Block Runge-Kutta Methods:

Let us start with the conventional s-stage RK method:

$$y_{n+1}^{(i)} = y_n + h \sum_{j=1}^s b_{ij} f(y_{n+1}^{(j)}), \quad i=1, \dots, s+1;$$

$$y_{n+1} = y_{n+1}^{(s+1)}, \quad n=0, 1, \dots \quad (42)$$

The general structure of the block Runge-Kutta (BRK) methods considered in this work is a direct generalization of this conventional method. We introduce block vectors y_n , the components of which are numerical approximations to the exact solution values at k points. To be more precise, let y_{n+1} be defined by:

$$Y_{n+1} = (y_{n,c1}, y_{n,c2}, \dots, y_{n,ck})^T, \quad c_k = 1$$

where $y_{n,c}$ denotes a numerical approximation to the exact solution value $y(t_n + ch)$. For scalar ODEs, we now define the s-stage block RK(BRK) method:

$$Y_{n+1}^{(i)} = A_i Y_n + h \sum_{j=1}^s B_{ij} f(x_{n+1}, y_{n+1}^{(j)}), \quad i=1, \dots, s+1; \}$$

$$Y_{n+1} = Y_{n+1}^{(s+1)}, \quad n=0, 1, \dots \quad (43)$$

where A_i and B_{ij} are k-by-k matrices and where we use the conventional that for any given vector $v = (v_j)$, $f(v)$ denotes the vector with entries $f(v_j)$. This method can be considered as the block analogue of (42). It is straightforwardly extended to systems of ODEs and therefore also to nonautonomous equations. In order to start the method, one needs the initial vector Y_0 , which requires as many starting values as there are distinct values c_j ($j = 1, \dots, k$).

In analogy with the Butcher array for describing the RK methods We may describe the BRK methods (43) by the $k(s+1)$ - by - $k(s+1)$ array

b_{11}	\dots	b_{1s}
\vdots	\vdots	\vdots
b_{s1}	\dots	b_{ss}
$b_{s+1,1}$	\dots	$b_{s+1,s}$

We may describe the BRK methods (43) by the $k(s+1)$ - by - $k(s+1)$ array:

$$\begin{array}{c|ccc}
 A_1 & B_{11} & \cdots & B_{1s} \\
 \vdots & \vdots & \vdots & \vdots \\
 A_s & B_{s1} & \cdots & B_{ss} \\
 \hline
 A_{s+1} & B_{s+1,1} & \cdots & B_{s+1,s}
 \end{array}$$

This notation is particularly convenient when more than two stages are involved. It frequently happens that the two last rows of this array are identical. In such cases, we shall omit the last row in order to save space. We call the method explicit if the matrices B_j vanish for $j \geq I$, and implicit otherwise, the k component of the blocks $F(Y_{n+1}^{(j)})$ can be computed in parallel; hence if k processors are available, then (explicit) BRK methods require not more than S (sequential) right-hand side evaluations per step. However, the required number of processors is often less than k , without causing the number of (sequential) right-hand side evaluations per step to exceed s . For instance, it may happen that in the formula for a particular component of Y_{n+1} no right-hand side evaluations occur, that is, all rows in the matrices B_{ij} corresponding to this component vanish. In such cases, the processor assigned to this component is not needed. Similarly; if the r^{th} column of all matrices B_{ij} vanishes, then the computation of the corresponding component of Y_{n+1} does not require any right-hand side evaluation not already occurring in the formulas for the other components, so that there is no need to assign a processor to this component. We define the optimal number of processors as the number of processors for which the number of (sequential) right-hand side evaluations per step is minimal.

The points t_n and $t_{n+c_j}h$ ($j \neq k$) will respectively be called step points and block points. Block points coincide with step points if the corresponding value of c_j is an integer. Upon completion of the integration process, the accuracy of the numerical solution obtained does not necessarily be the same at all points $t_n + c_j h$. Points where the corresponding components of Y_{n+1} do have the same order as the components corresponding to the step points t_n will be called output points.

The general explicit one- and two-stages methods are respectively given by:

$$\begin{array}{c|c} A_1 & 0 \\ \hline A_2 & B_{21} \end{array} \quad \text{i.e., } Y_{n+1} = A_2 Y_n + hB_{21}f(A_1 Y_n)$$

And

$$\begin{array}{c|cc} A_1 & 0 & 0 \\ A_2 & B_{21} & 0 \\ \hline A_3 & B_{31} & 0 \end{array}$$

$$\text{i.e., } Y_{n+1} = A_3 Y_n + hB_{31}f(A_1 Y_n) + hB_{32}f(A_2 Y_n + hB_{21}f(A_1 Y_n)).$$

Here, 0 denotes the k-by-k matrix with zero entries

As a numerical example of an (explicit) 3-stage method, we present the modified multi step method of Butcher of order 5 as a BRK method: the block point vector is given by $c = (0,1)^T$ and the Butcher array assumes the form :

$$\begin{array}{cc|cccc} 1 & 0 & & & & & & \\ 0 & 1 & & & & & & \\ & & & & & & & \\ 1 & 0 & 3/8 & 9/8 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ & & & & & & & \\ -\frac{23}{5} & \frac{28}{5} & -\frac{26}{15} & 0 & \frac{32}{15} & -4 & & \\ & & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & & \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{31} & \frac{32}{31} & -\frac{1}{93} & \frac{12}{93} & \frac{64}{93} & 0 & \frac{15}{93} & 0 \end{array} \quad c = (0,1)^T$$

The construction of higher-order BRK methods is rather difficult in the general case. Houwen and Sommeijer [9] construct high-order methods of a special form which are obtained by using the predictor-corrector (PC) technique. The starting point is the special implicit two-stage method

$$\text{i.e., } Y_{n+1} = AY_n + hBf(Y_n) + hCf(Y_{n+1}). \quad (44)$$

If C does not vanish, then we can use this method as corrector and if $c = 0$, then it can be used as (a one-stage) predictor formula, e.g

$$\text{i.e., } Y_{n+1} = AY_n + hBf(Y_n) \quad (45)$$

From this pair we can generate higher-stage BRK methods by PC iteration provided that the block point vectors $c_1 (c_1, \dots, C_k)$ are

identical. For example, in PECE mode we obtain the special two-stage BRK method

$$\text{i.e., } Y_{n+1} = AY_n + hBf(Y_n) + hCf(DY_n + hEf(Y_n)). \quad (46)$$

Finally, it should be remarked that (45) is also the representation of the so-called general linear methods introduced by Butcher in 1966. Most methods from the literature (including the BRK method (43) can be cast into the form (45). However, although the original method is explicit, the general linear method version is often implicit. For example, the explicit two-stage BRK method (46) can be rewritten in the form (45) by redefining the matrices A , B and C in (45), but C will not be a zero matrix.

In the following subsections, we present in BRK form two methods which have been proposed for use on parallel computers. In particular, we give examples of the predictor-corrector methods of Miranker and Liniger [14] and Shampine and Watts [16]. A discussion of block methods for parallel computation may be found in Gear [7].

4.8: Methods of Miranker and Liniger:

The methods of Miranker and Liniger [14] can be represented as explicit, one-stage BRK methods. For example, their second-order method can be represented by the array

$$\begin{array}{cc|cc} 1 & 0 & & \\ 0 & 1 & & \\ \hline 0 & 1 & 2 & 0 \\ 0 & 1 & 1/2 & 1/2 \end{array} \quad c = (2,1)^T \quad (47)$$

And their fourth – order method by

$$\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 1 & & & & \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{-1}{3} & \frac{4}{3} & \frac{8}{3} & \frac{-5}{3} \\ 0 & 0 & 0 & 1 & \frac{1}{24} & \frac{-5}{24} & \frac{9}{24} & \frac{19}{24} \end{array} \quad c = (-1,0,2,1)^T \quad (48)$$

Both methods require only two processors and respectively two and four starting values when implemented in BRK form.

4.9: Predictor-Corrector Method of Sham pine and Watts:

The PC method of Shampine and Watts [17] is based on the block method of Clippirige and Dimsdale (1958), which can be presented in the form (44) as:

$$\begin{array}{cc|ccc}
 1 & 0 & & & \\
 0 & 1 & & & \\
 \hline
 0 & 1 & 0 & \frac{5}{24} & \frac{1}{3} & \frac{-1}{24} \\
 0 & 1 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6}
 \end{array} \quad c = (1/2, 1)^T \quad (49)$$

And on the predictor defined by

$$\begin{array}{cccc|cccc}
 1 & 0 & 0 & 0 & & & & \\
 0 & 1 & 0 & 0 & & & & \\
 0 & 0 & 1 & 0 & & & & \\
 0 & 0 & 0 & 1 & & & & \\
 \hline
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{4} & \frac{-1}{3} & \frac{-13}{12} \\
 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{29}{24} & -3 & \frac{79}{24}
 \end{array} \quad c = (-1/2, 0, 1/2, 1)^T \quad (50)$$

Method (49) is one of the oldest block methods proposed in the literature. Sham pine and Watts proved that this corrector method is fourth-order accurate at the step points. They also proved that the predictor method is third-order accurate and possesses favorable stability properties. This predictor can also be applied as a method on its own and requires four starting values and one processor.

In order to apply the PC pair (50) - (49) using the BRK format, we rewrite the corrector in the form:

$$\begin{array}{cccc|cccccccc}
 1 & 0 & 0 & 0 & & & & & & & & \\
 0 & 1 & 0 & 0 & & & & & & & & \\
 0 & 0 & 1 & 0 & & & & & & & & \\
 0 & 0 & 0 & 1 & & & & & & & & \\
 \hline
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{5}{24} & 0 & 0 & \frac{1}{3} & \frac{-1}{24} \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 & \frac{2}{3} & \frac{1}{6}
 \end{array} \quad c = (-1/2, 0, 1/2, 1)^T \quad (51)$$

The *PC* method of Sham pine and Watts was implemented by Worland [20] On two processors.

5: New Parallel Block 6 (5) Runge-Kutta Method:-

Runge-Kutta formulae have been widely used for the numerical integration of initial value problem:

$$y' = f(x, y), \quad y(x_0) = y_0$$

An explicit q- stage formula can be written in the general form:

$$y_{n+1} = y_n + h \sum_{i=1}^q b_i k_i, \quad k_i = f(x_{n+1} + c_i h, y_n + h \sum_{j=1}^{i-1} a_{ij} k_j);$$

where the b_i , c_i and a_{ij} are constants that define the formula and $h = x_{n+1} - x_n$ is the step length. The approach used involves the derivation of a Runge-Kutta pair of orders p and p-1. This allows the computation of two separate solutions:

$$y_{n+1} = y_n + h \sum_{i=1}^q b_i k_i, \quad (\text{order } p),$$

$$\bar{y}_{n+1} = y_n + h \sum_{i=1}^q \bar{b}_i k_i \quad (\text{order } p-1),$$

The idea presented was to derive parallel block formulae which compute the solution at more than one step point.

Our approach for the derivation of a Parallel block 6(5) formula in stages is as follows:

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4 + b_5 k_5 + b_6 k_6 + b_7 k_7 + b_8 k_8),$$

$$\bar{y}_{n+1} = y_n + h(\bar{b}_1 k_1 + \bar{b}_2 k_2 + \bar{b}_3 k_3 + \bar{b}_4 k_4 + \bar{b}_5 k_5 + \bar{b}_6 k_6 + \bar{b}_7 k_7 + \bar{b}_8 k_8),$$

Where

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + c_2 h, y_n + h a_{21} k_1),$$

$$k_3 = f(x_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)),$$

$$k_4 = f(x_n + c_4 h, y_n + h(a_{41} k_1 + a_{42} k_2)),$$

$$k_5 = f(x_n + c_5 h, y_n + h(a_{51} k_1 + a_{52} k_2 + a_{53} k_3 + a_{54} k_4)),$$

$$k_6 = f(x_n + c_6 h, y_n + h(a_{61} k_1 + a_{62} k_2 + a_{63} k_3 + a_{64} k_4 + a_{65} k_5)),$$

$$k_6 = f(x_n + c_6 h, y_n + h(a_{61} k_1 + a_{62} k_2 + a_{63} k_3 + a_{64} k_4 + a_{65} k_5)),$$

$$k_7 = f(x_n + c_7 h, y_n + h(a_{71} k_1 + a_{72} k_2 + a_{73} k_3 + a_{74} k_4 + a_{75} k_5 + a_{76} k_6)),$$

$$k_8 = f(x_n + c_8 h, y_n + h(a_{81} k_1 + a_{82} k_2 + a_{83} k_3 + a_{84} k_4 + a_{85} k_5 + a_{86} k_6 + a_{87} k_7)),$$

To find the coefficients of the method we make some assumptions and we compare these coefficients with Taylor series expansion about $y(x_{n+1})$, we get:

$$c_2 = a_{21},$$

$$c_3 = a_{31} + a_{32},$$

$$c_4 = a_{41} + a_{42} + a_{43},$$

$$\begin{aligned}
c_5 &= a_{51} + a_{52} + a_{53} + a_{54}, \\
c_6 &= a_{61} + a_{62} + a_{63} + a_{64} + a_{65}, \\
c_7 &= a_{71} + a_{72} + a_{73} + a_{74} + a_{75} + a_{767}, \\
c_8 &= a_{81} + a_{82} + a_{83} + a_{84} + a_{85} + a_{86} + a_{87}, \\
b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8 &= 1 \\
b_2c_2 + b_3c_3 + b_4c_4 + b_5c_5 + b_6c_6 + b_7c_7 + b_8c_8 &= 1 \setminus 2, \\
b_2c_2^2 + b_3c_3^2 + b_4c_4^2 + b_5c_5^2 + b_6c_6^2 + b_7c_7^2 + b_8c_8^2 &= 1/3, \\
b_2c_3^3 + b_3c_3^3 + b_4c_4^3 + b_5c_5^3 + b_6c_6^3 + b_7c_7^3 + b_8c_8^3 &= 1/4, \\
b_2a_{21}c_2 + b_3a_{32}c_3 + b_4a_{43}c_4 + b_5a_{54}c_5 + b_6a_{65}c_6 + b_7a_{76}c_7 + b_8a_{87}c_8 &= 1/3,
\end{aligned}$$

Let $b_2 = 0$

$$\sum b_i a_{ij} = b_j (1 - c_j), \quad 1 \leq j \leq 8$$

Which tends to :

$$b_3 a_{31} + b_4 a_{41} + b_5 a_{51} + b_6 a_{61} + b_7 a_{71} + b_8 a_{81} = b_1 (1 - c_1)$$

$$b_3 a_{32} + b_4 a_{42} + b_5 a_{52} + b_6 a_{62} + b_7 a_{72} + b_8 a_{82} = 0$$

$$b_4 a_{43} + b_5 a_{53} + b_6 a_{63} + b_7 a_{73} + b_8 a_{83} = b_3 (1 - c_3)$$

$$b_5 a_{54} + b_6 a_{64} + b_7 a_{74} + b_8 a_{84} = b_4 (1 - c_4)$$

$$b_6 a_{65} + b_7 a_{75} + b_8 a_{85} = b_5 (1 - c_5)$$

$$b_7 a_{76} + b_8 a_{86} = b_6 (1 - c_6)$$

$$b_8 a_{87} = b_7 (1 - c_7)$$

For parallel purposes consider $a_{86} = a_{76} = 0$

Set:

$$c_8 = 1. \text{ Choose } b_6, c_6, c_7 \text{ arbitrarily}$$

$$\text{Let } b_6 = 0, c_6 = 1 \text{ \& } c_7 = 0.0666667$$

Solve the linear equations:

$$\sum_{j=1}^5 b_i c_i^j = \frac{1}{(j+1)} \text{ for } b_3, b_4, b_5, b_7, b_8.$$

By consider $c_2 = 0.1666667, c_3 = 0.2666667, c_4 = 0.666667, c_5 = 0.833333$

and by using Gauss elimination (we wrote a Pascal program for this technique) we get:

$$b_3 = 0.39012728, b_4 = 0.31922927, b_5 = 0.13521570, b_7 = 0.01057827, b_8 = 0.06976138.$$

Set

$$b_1 = 1 - \sum_{i=2}^8 b_i \Rightarrow b_1 = 0.0750881.$$

Choose a_{64}, a_{65} arbitrarily and solve for a_{62} and a_{63}

$$\sum a_{6i}c_i = 1/2c_6^2 \quad , \quad \sum a_{6i}c_i^2 = 1/3c_6^3$$

$$\sum b_i(1-c_i)(c_i-c_7)a_{ij}c_j^3 = \frac{1}{168} - \frac{1}{120}c_7$$

Set

$$a_{61} = c_6 - \sum_{i=3}^5 a_{6i}$$

Choose a_{74} arbitrarily solve for a_{75}, a_{73} and a_{72} :

$$\sum a_{7i}c_i = 1/2c_7^2 \quad , \quad \sum a_{7i}c_i^2 = 1/3c_7^3$$

$$\sum b_i(1-c_i)a_{ij}a_{j3} = 0$$

$$\sum b_i(1-c_i)a_{ij}a_j^4 = \frac{1}{210}$$

$$\sum b_i(1-c_i)a_{ij}a_{jk}a_k^3 = \frac{1}{840}$$

Set

$$a_{71} = c_7 - \sum_{i=3}^6 a_{7i}$$

Set

$$a_{8j} = \left[b_j(1-c_j) - \sum_{k=j+1}^7 b_k a_{kj} \right] / b_8 \quad , \quad j = 3,4,5,6$$

$$a_{87} = b_7(1-c_7) / b_8$$

$$a_{81} = c_8 - \sum_{j=3}^7 a_{8j}$$

This defined all the coefficients of the formula of order 6.

$$a_{21} = 0.16666667 \quad a_{31} = 0.05333333 \quad a_{32} = 0.21333333$$

$$a_{41} = 0.83333333 \quad a_{42} = -2.66666667 \quad a_{43} = 2.5$$

$$a_{51} = -2.578125 \quad a_{52} = 9.16666667 \quad a_{53} = -6.640625$$

$$a_{54} = 0.88541667 \quad a_{61} = 2.4 \quad a_{62} = -8$$

$$a_{63} = 6.5045752 \quad a_{64} = -0.3055556 \quad a_{65} = 0.34509804$$

$$a_{71} = -0.55086667 \quad a_{72} = 1.65333333 \quad a_{73} = -0.94558824$$

$$a_{74} = -0.324 \quad a_{75} = 0.23378824 \quad a_{81} = 2.03546512$$

$$a_{82} = -6.97674419 \quad a_{83} = 5.64817982 \quad a_{84} = -0.13738157$$

$$a_{85} = 0.28630227 \quad a_{87} = 0.141526042.$$

It is now straight forward to derive the embedded formula of order 5 and this can be done in the following way:

Choose \bar{b}_7, \bar{b}_8 arbitrarily, set $\bar{b}_2 = 0$

Solve the linear equations

$$\sum \bar{b}_i c_i^j = 1/(j+1) \quad , \quad j=1,2,3,4$$

$$\sum \bar{b}_i a_{ij} c_j^3 = 1/20 \quad \text{for } \bar{b}_3, \bar{b}_4, \bar{b}_5, \bar{b}_6.$$

$$\text{set } \bar{b}_1 = 1 - \sum_{i=2}^8 \bar{b}_i$$

Thus we get by using Gauss elimination technique:

$$\bar{b}_3 = 0.39682041, \quad \bar{b}_4 = 0.31268063, \quad \bar{b}_5 = 0.140988713,$$

$$\bar{b}_6 = 0.06823801, \quad \bar{b}_1 = 0.08127382$$

Since we are dealing with stiff ODEs, we need that the integration method to be implicit, so by putting $h=-h$ we obtain such formula.

Example 1:

Consider the stiff equation:

$$y'' + 1001y' + 1000y = 0 \quad , \quad x \in (0,1) \quad (52)$$

which can be rewritten as a first order system as following :

$$y_1' = y_2 \quad , \quad y_1(0) = 1 \quad (53)$$

$$y_2' = -1001y_2 - 1000y_1 \quad , \quad y_2(0) = -1 \quad (54)$$

Applying the of section (5) for the system we get the following results ($h=0.1$):

Table (1) results of the method of section (5) for the above system

x	Apr. y_1	Apr. y_2	Apr. y_3	Apr. y_4	Exact
0.1	.9048392	-.9048392	.9048373	-.9048373	.9048374
0.2	.8187342	-.8187342	.8187305	-.8187305	.8187308
0.3	.7408234	-.7408234	.7408178	-.7408178	.7408182
0.4	.6703269	-.6703269	.6703195	-.6703195	.6703201
0.5	.6065392	-.6065392	.6065299	-.6065299	.6065307
0.6	.5488218	-.5488218	.5488107	-.5488107	.5488116
0.7	.4965970	-.4965970	.4965841	-.4965841	.4965853
0.8	.4493421	-.4493421	.4493274	-.4493274	.4493290
0.9	.4065843	-.4065843	.4065678	-.4065678	.4065697
1.0	.3678954	-.3678954	.3678771	-.3678771	.3678794

Where

Apr. y_1 =correction approximation of (53) for the method of order 6

Apr. y_2 =correction approximation of (54) for the method of order 6

Apr. y_3 =correction approximation of (53) for the method of order 5

Apr. y_4 =correction approximation of (54) for the method of order 5

6: Conclusion:-

Parallel block methods provide interesting schemes. These schemes contain many tasks which can be computed in parallel and the performance of the particular parallel block scheme depends on the number of these independent tasks. We conclude that the algorithms developed are suitable for executing in fully parallel systems, i.e. MIMD computers. Moreover the solutions of stiff ODEs using parallel computers offer a promising field for future research.

7: Future work:-

We suggest driving and developing parallel block methods of higher order for solving stiff IVPs.

References

1. O. Abou-Rabia and L.G. Birta, "Some Variations on the BPC parallel integration method", in R. Crosbie and P. Luker (eds.) Proceeding of the "**1986 Summer computer Simulation conference**" (1986), 37-42.
2. L. G. Birta and O. Abou-Rabia, "Parallel block Predictor – corrector methods for ODEs", *IEEE Trans. On Computers*, Vol c- 36, No. 3 (1987), 299-311.
3. _____, "Block Runge-Kutta methods for the numerical integration of initial value problems in ordinary differential equations, Part I: The non stiff case", *Mathematics of Computation*, Vol. 40, No.161 (1983), 193-206.
4. _____, "Block Runge-Kutta methods for the numerical integration of initial value problems in ordinary differential equations, Part II: The stiff case", *Mathematics of Computation*, Vol. 40, No.161 (1983), 193-206.
5. M. A. Franklin, "Parallel solution of ordinary equations", *IEEE Trans. On Computers* Vol.C-27, No. 4 (1978), 413-420.
6. C.W. Gear, "Runge – Kutta starters for multi step methods" *ACM Trans. Math. Soft ware*, Vol.6 (1980) 263-279.
7. C.W. Gear, "Parallel methods for ordinary differential equations", *Calcolo Journal of Numerical analysis and theory of computation, Rome, Italy*, Vol. 25 Nos. 1-2 (1988) 1-20.
8. S. K. Ghoshal, M. Gupta and V. Rajaraman, "A parallel multi step predictor-corrector algorithm for solving ordinary differential equations" *J. of Parallel and Distributed Computing*, Vol.6 (1989), 636-648.
9. P.J. van Der Houwen and B. P. Sommeijer, "Block Runge-kutta methods on parallel computers" *Z. Angew, Math. Mech.* 72 (1992), 3-18.

10. N. K. M. A. Franklin and A. Sen, " Optimally stable parallel Predictors for Adams-Moulton Correctors" , *Comp. & Math. with Apply*. Vol3 (1977), 217-233.
11. E. J. Kerckhoffs, " Parallel algorithms for ordinary differential equations : An introductory review" , in : R. Crosbie and P. Luker (eds.), Proceeding of the " *1986 Summer Computer Simulation Conference*" , (1986), 947-952.
12. B.M.S. Khalaf , "Parallel numerical algorithms for solving ordinary differential equations " *Ph.D. Thesis , University of Leeds , U.K.*, 1990.
13. J.D. Lambert , " Computational methods in ordinary differential equations " , *John Wiley & Sons Inc.*, 1974.
14. W.L. Miranker , " A survey of parallelism in numerical analysis" , *SIAM Review* , Vol.13 , No. 4 (1971), 524-5247.
15. J.B. Rosser , " A Runge-Kutta for all seasons " , *SIAM Review*, Vol. 9 (1967), 417-452.
16. L. F. Sham pine and H. A. Watts, "Block implicit one – step methods" , *Mathematics of Computation*, Vol.23 (1969), 731-740.
17. T. E. Shoup, " Applied numerical methods for the micro – computers" *Prentice- Hall, New Jersey* , 1984.
18. P.J. Van der Hooven , B. P. Sommeijer and W. Couzy , Embedded diagonally implicit Runge-Kutta algorithms on parallel computers" *Mathematics of Computation*, Vol. 58, (1992), 135-159.
19. D.A. Voss and S. Abbas , " Block predictor – corrector schemes for the parallel solution of ODEs", *Computers Mathematics Application* , Vol. 6 , (1997), 65-72.
20. P. B. Worland , " Parallel methods for the numerical solution of ordinary differential equations" *IEEE Trans. On Computers* , (1976), 1045-1048.