Survey of Parallel Block Methods

Bashir M. Khalaf **Mohammed A. Al-Taee**
Department of Computers Sciences / College of Education
University of Mosul

Abdulhabib Abdullah

Received **Accepted**
16 / 12 / 2008 **06 / 04 / 2009**

الملخص

الغرض الرئيسي من هذا البحث هو دراسة تطوير خوارزميات بلوك العددية المتوازية لحل المعادلات التفاضلية الاعتيادية الصلبة والتي هي مناسبة للتنفيذ على حاسبات ذات عمليات متعددة في آن واحد.

Abstract

The main purpose of this research is the survey of the development Block parallel numerical algorithms for solute stiff ordinary differential equations which are suitable for running on MIMD (Multiple instruction streams with multiple data streams) computers.

1: Itroduction

The novel property of these methods which we shall discuss here is that of simultaneously production approximations to the solution of the initial value problem at a block of points $x_{n+1}, x_{n+2}, ..., x_{n+N}$. Although these methods will be formulated in term of linear multi step methods and shall see that they are equivalent to certain Runge-Kutta methods and preserve the traditional Runge-Kutta advantages of being setf-stavting and of permitting easy change of step-length. Their advautage over conventional Runge-Kutta methods lies in the fact that they are les, expensive in terms of function evaluations for given order.

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 predicator-corrector algorithms for general use.

Several authors (see, for example, [1,2,3,5,6,7,8,9,12,15]) have considered block methods for the parallel solution of the initial value problem (IVP).

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

By means of a single application of a calculation unit, a block method yields a sequence of new estimates for y. If k ≥1 is the block size, then in simple cases the values of x at which solution are computed will be evenly separated [14]. 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 $Y_m = (y_{n+1}, y_{n+2}, ---, y_{n+k})^T$ with y_{n+j} (1 ≤ j ≤ 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 [12]. In addition the underlying formulae may only rfer to the end point of the previous block, so called one-step methods. In order words, by one-step methods, we mean methods that compute the block of values y_{n+i}, i=1, …,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 and stiff ODEs:

Each has the following form for Nonstiff case:

$$y_{n+1} - y_n = h\{1/4k_1 + 3/4k_3\}$$
$$y_{n+2} - y_n = h\{9/32k_1 + 21/32k_3 + 7/32k_4 + 27/32k_5\}$$
$$y_{n+3} - y_n = h\{105/504\, k_1 + 117/112\, k_3 + 69/48\, k_4 + 39/126\, k_6\}$$

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.

And has the following form stiff case to compute $y_{n+1}^{(1)}, y_{n+2}^{(1)}, y_{n+1}^{(2)}, y_{n+2}^{(2)}, y_{n+3}^{(2)}$ respectively:

										1	1								
				1	1					2	1	1							
1	1			2	1	1				1	1/2	−1/2	1						
	1				1	1					1/2	−1/2	1						

1	1				
2	1	1			
1	1/2	−1/2	1		
2	1	−1	1	1	
	1	−1	1	1	

and **(2)**

We note that five stage block formula (2) provides 5 separate solution.

3.1: Parallel Block Implicit Methods:

Block implicit methods as described by Shampine and Watts [12] and Rosser [11] have been shown to be competitive with standard methods for integrating ODEs. Worland [15] 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 [12].

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)[15]):

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

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

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

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

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

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

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.

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

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 easy 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 [12] is as follow:

The predictor equations are:

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

$$Y_{n+2}^{p} = \frac{1}{3}(Y_{n-2}^{c} + Y_{n-1}^{c} + Y_{n}^{c}) + \frac{h}{12}(29F_{n-2}^{c} - 72F_{n-1}^{c} + 79F_{n}^{c}), \quad \textbf{(4)}$$

The corrector equations are:

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

$$Y_{n+2}^{c} = Y_{n}^{c} + \frac{h}{3}(F_{n}^{c} + 4F_{n+1}^{p} + F_{n+2}^{p}) \quad \textbf{(6)}$$

In this case, each block consist 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 pint 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 (3) through (6) is given in the following figure:

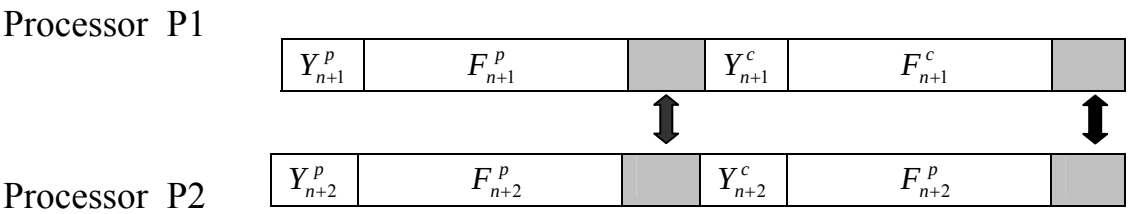
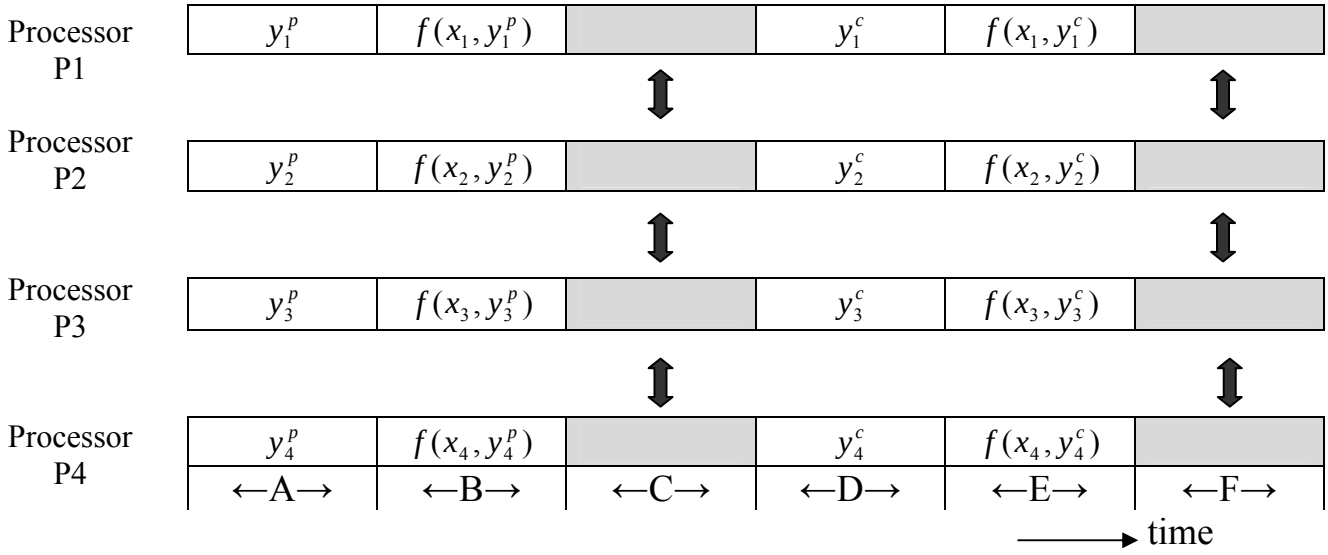


Fig. (1): timing for PBPC integration algorithm Data are being exchanged[4].

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[4].

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.

3.3: 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:

$$Y_m = ey_n + hdf_n + hBF(Y_m) \quad \textbf{(7)}$$

Where e and k-vectors, B is a k× k matrix, and F is a k=factor whose j^{th} entry is $f_{n+j} = f(x_{n+j}, y_{n+j})$, $1 \le j \le k$. As (7) 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:

$$Y_m^{(0)} = ey_n + h\tilde{d}f_n, \quad \textbf{(8)}$$

Where is k-vector. Substitution of into the right-hand side of (7) yields the block predictor-corrector (BPC) method:

$$Y_m = ey_n + hdf_n + hBF(ey_n + h\tilde{d}f_n) \quad \textbf{(9)}$$

We can write (9) in the form:

$$Y_m = ey_n + hdf_n + AY_m^{(0)} + hBf_n \quad \textbf{(10)}$$

Where A is k× 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 (9) into the right-hand side of (7) arriving at $P(EC)^{v}E^{1-y}$ mode, in which γ=0 indicates that a final evaluation is done before proceeding to the next block. Abbas and Devles [14] 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, …, k as:

$$Y_{n+r}^{(0)} = y_n + rhf(x_n, y_n),$$

$$Y_{n+r}^{(1)} = y_n + h/2\, f(x_n, y_n), h\sum_{i=1}^{r-1} f(x_{n+i}, y_{n+i}^{(0)}) + h/2\, f(x_{n+r}, y_{n+r}^{(0)}), \qquad \textbf{(11)}$$

$$Y_{n+r}^{(2)} = y_n + h/2\, f(x_n, y_n), h\sum_{i=1}^{r-1} f(x_{n+i}, y_{n+i}^{(1)}) + h/2\, f(x_{n+r}, y_{n+r}^{(1)}).$$

With $Y_m^{(0)}$ by (8), method (11) has $P(EC)^2$ from

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

$$Y_m^{(s+1)} = ey_n + hdf_n + hBF(Y_m^{(s)}), \qquad s = 0, 1$$

Where

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

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 (8). following an application of equation (10) or (8), equation (7) may be applied iteratively through:

$$Y_m^{(i+1)} = ey_n + hdf_n + hBF_m^{(i)} \qquad \textbf{(12)}$$

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

Inspection of equations (7), (10) and (12) shows the potential for parallelism in the calculation of y_m (equation (10)), $F_m^{(i)}$ (equation (12)), and also in the calculation of the right-hand side of equation (7). the granularity of the parallelism is variable and depends upon both the block size and the computation effort to calculate the derivative function f(x , y).

3.4: Parallel Block Methods – Fixed Order and Varying Length:

This method uses equation (8) to predict solutions to the problem then applies the equation (13) iteratively:

$$Y_m^{(i+1)} = ey_0 + hAF_m^p \qquad \textbf{(13)}$$

Where $Y_m^{(i+1)}$ is as in (12), e is a unit k-vector, A is a k × k matrix and F_m^p is a (k+1)-vector whose j^{th} entry is $f_{j-1} = f(x_0 + (j-1)h, y_{j-1}^p)$,

With y_{j-1}^p the latest estimate for y_j.

Lets us consider an implementation of the method with A define by:

$a_{r1} = a_{r(r+1)} = 1/2,$ where r=1, …, k **(14)**

And $a_{rj} = 1$ where r=2, …, k and j=2, …,r **(15)**

It can be seen that there is a potential for parallelism in the calculation of the predicted y values i.e Y_m^p (equation (8)) and hence the predicted derivative values F_m^P. The right – hand side of (13) can be split up into sub-block work packets and farmed out to processors, allowing a high degree of parallelism to be accomplished.

For example with 2 processors, equation (13) can be manipulated thus:

First with A in its complete form:

$$y_m^{(i+1)} = e\,y_0 + h\begin{bmatrix} 1/2 & 1/2 & & & & & & & & \\ 1/2 & 1 & 1/2 & & & & & & & \\ 1/2 & 1 & 1 & 1/2 & & & & & & \\ \vdots & \vdots & \vdots & \vdots & & & & & & \\ 1/2 & 1 & 1 & 1 & \dots & 1/2 & & & & \\ 1/2 & 1 & 1 & 1 & \dots & 1 & 1/2 & & & \\ \vdots & \vdots & & & & \vdots & \vdots & & & \\ 1/2 & 1 & 1 & 1 & \cdots & 1 & 1 & 1/2 & & \\ 1/2 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1/2 \end{bmatrix} F_m^p \qquad \textbf{(16)}$$

The calculation of A F_m^p is readily parallelized using a splitting of A.

If we write A as:

$$A = \begin{pmatrix} 1/2 & 1/2 & & & & & \\ 1/2 & 1 & 1/2 & & & & \\ 1/2 & 1 & 1 & 1/2 & & & 0 \\ \vdots & \vdots & \vdots & \vdots & & & \\ 1/2 & 1 & 1 & 1 & \dots & 1/2 & \\ 1/2 & 1 & 1 & 1 & \dots & 1 & \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & 0 \\ 1/2 & 1 & 1 & 1 & \dots & 1 & \\ 1/2 & 1 & 1 & 1 & \dots & 1 & \end{pmatrix} + \begin{pmatrix} & & & & & \\ 0 & & 0 & & & \\ & & & & & \\ & 1/2 & & & & \\ 0 & \vdots & & & & \\ & & & & & \\ & 1 & \dots & 1/2 & & \\ & 1 & \dots & 1 & 1/2 & \end{pmatrix}, \qquad \textbf{(17)}$$

And further split A up to:

$$A=\left(\begin{array}{c|cc}1/2 & & \\ 1/2 & 0 & 0 \\ 1/2 & & \\ \vdots & & \\ 1/2 & & \\ \hline 1/2 & & \\ \vdots & 0 & 0 \\ 1/2 & & \\ 1/2 & & \end{array}\right)+\left(\begin{array}{c|ccccc|c} & 1/2 & & & & & \\ & 1 & 1/2 & & & & \\ 0 & 1 & 1 & 1/2 & & & \\ & \vdots & & \vdots & & & 0 \\ & 1 & 1 & 1 & \cdots & 1/2 & \\ \hline & 1 & 1 & 1 & \cdots & \vdots & \\ 0 & \vdots & & \vdots & \cdots & \vdots & 0 \\ & 1 & 1 & 1 & & 1 & \\ & 1 & 1 & 1 & \cdots & 1 & \end{array}\right)$$

$$\left(\begin{array}{cc|ccccc}0 & 0 & & & 0 & & \\ \hline 0 & 0 & 1/2 & & & & \\ & & \vdots & & & & \\ & & 1 & \cdots & 1/2 & & \\ & & 1 & \cdots & 1 & 1/2 & \end{array}\right) \qquad (18)$$

Or $A_0 + A_1 + A_2$ **(19)**

Then (16) could be represented by:

$$Y_m^{(i+1)} = ey_0 + h\{A_0 + A_1 + A_2\}F_m^P \qquad \textbf{(20)}$$

Or $Y_m^{(i+1)} = ey_0 + hA_0F_{m0}^P + hA_1F_{m1}^P + A_2F_{m2}^P$ **(21)**

Where

$F_{m0}^P = [f_0,0,0,0,....,0]^T, F_{m1}^P = [0, f_1, f_2,..., f_{k/2},0,...,0]^T$ and

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

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 $\sum f_r$, r=1, …, k/2 then estimates for $Y_r^{(i)}$, r=1 ,… , k/2 and $Y_s^{(i)}$, s=k/2+1 ,… , 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 (21)[4], i.e

$$Y_m^{(i+1)} = ey_0 + hA_0F_{m0}^P + hA_1F_{m1}^P + hB_2G_{m2}^P \qquad \textbf{(22)}$$

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

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

The splitting up continue with B_2:

$$B_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \begin{matrix} 1/2 & & & & \\ 1 & 1/2 & & & \\ 1 & 1 & 1/2 & & \\ \vdots & \vdots & \vdots & & \\ 1 & 1 & 1 & \cdots & 1/2 \end{matrix} & 0 \\ 0 & \begin{matrix} 1 & 1 & 1 & \cdots & 1 \\ \vdots & & & \cdots & \vdots \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{matrix} & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \begin{matrix} 1/2 & & & \\ \vdots & & & \\ 1 & \cdots & 1/2 & \\ 1 & \cdots & 1 & 1/2 \end{matrix} \end{pmatrix} \quad (23)$$

Or $B_2 = A_2 + B_3$ **(24)**

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

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

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

$$B_{p-1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \begin{matrix} 1/2 & & & & \\ 1 & & & & \\ 1 & 1 & 1/2 & & \\ \vdots & \vdots & \vdots & & \\ 1 & 1 & 1 & \cdots & 1/2 \end{matrix} & 0 \\ 0 & \begin{matrix} 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{matrix} & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \begin{matrix} 1/2 & & & \\ \vdots & & & \\ 1 & \cdots & 1/2 & \\ 1 & \cdots & 1 & 1/2 \end{matrix} \end{pmatrix} \quad (25)$$

Or $B_{p-1} = A_{p-1} + A_p$ **(26)**

Giving us:

$$Y_m^{(i+1)} = ey_0 + hA_0F_{m0}^P + h\sum_{r=1}^{P} A_rF_{mr}^PG_{m2}^P \quad \textbf{(27)}$$

Where $F_{me}^P = [0,...,0, f_{(r-1)q+1},...., f_{rq},0,....,0]^T$ and the A_r matrices are defined as above.

Thus the work of calculating k y-values may be farmed out by splitting up F_m^p and A into P+1 sub-vectors and matrices. As each processor knows the value of y_0 and f_0, processor P needs only to receive an estimate of $\sum f_r$, r= 1, …, (p-1)q, to calculate its sub – block of y-values, $y_r^{(i)}$, r = (p-1)q+1, pq[4].

On inspection of the method it was found that increased accuracy could be achieved in the corrector by updating the estimate of f_{j-1} in the

calculation of $Y_j^{(i)}$ through the sub-block. E.g. consider the first point in a sub-block $Y_j^{(i)}$. The equation to calculate this point can be written:

$$y_j^{(i)} = y_0 + h/2\, f_0 + h\sum_{r=0}^{j-1} f_r^p + h/2\, f_j^p \qquad \textbf{(28)}$$

Where y_0, f_0, and f_j^p are known locally and the sum of f_j^p's are communicated.

The next point in the block is calculated by:

$$y_{j+1}^p = y_0 + h/2\, f_0 + h\sum_{r=0}^{j-1} f_r^p + h f_j^p + h/2\, f_{j+1}^p \qquad \textbf{(29)}$$

Or $$y_{j+1}^{(i)} = y_j^{(i)} + h/2(f_j^p + f_{j+1}^p) \qquad \textbf{(30)}$$

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 (30) 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 ≠ 1) is:

$$y_r^{(i)} = y_{r-1}^{(i)} + h/2(f_{r-1}^i + f_r^p) \qquad \textbf{(31)}$$

3.5: Methods of Miranker and Liniger:

The methods of Miranker and Liniger [10] can be represented as explicit, one-stage BRK methods. For example, their second-order method can be represented by the array

$$\begin{array}{cc|cc} 1 & 0 & & \\ 0 & 1 & & \\ \hline 0 & 1 & 2 & 0 \\ 0 & 1 & 1/2 & 1/2 \end{array} \qquad c = (2,1)^T \qquad \textbf{(32)}$$

And their fourth – order method by

$$\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 1 & & & & \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{-1}{3} & \frac{4}{3} & \frac{8}{3} & \frac{-5}{3} \\ 0 & 0 & 0 & 1 & \frac{1}{24} & \frac{-5}{24} & \frac{9}{24} & \frac{19}{24} \end{array} \qquad c = (-1,0,2,1)^T \qquad \textbf{(33)}$$

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

3.6: Predictor-Corrector Method of Shampine and Watts:

The PC method of Shampine and Watts [13] is based on the block method of Clippirige and Dimsdale (1958), which can be presented in the form (44) as:

$$\begin{array}{cc|ccc} 1 & 0 & & & \\ 0 & 1 & & & \\ \hline 0 & 1 & 0 & \frac{5}{24} & \frac{1}{3} & \frac{-1}{24} \\ 0 & 1 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \qquad c=(1/2,1)^T \qquad (34)$$

And on the predictor defined by

$$\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ \hline 0 & 0 & 0 & 1 & & & & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{4} & \frac{-1}{3} & \frac{-13}{12} \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{29}{24} & -3 & \frac{79}{24} \end{array} \qquad c=(-1/2,0,1/2,1)^T \qquad (35)$$

Method (34) 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 (35) - (34) using the BRK format, we rewrite the corrector in the form:

$$\begin{array}{cccc|cccccccc} 1 & 0 & 0 & 0 & & & & & & & & \\ 0 & 1 & 0 & 0 & & & & & & & & \\ 0 & 0 & 1 & 0 & & & & & & & & \\ 0 & 0 & 0 & 1 & & & & & & & & \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{5}{24} & 0 & 0 & \frac{1}{3} & \frac{-1}{24} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 & \frac{2}{3} & \frac{1}{6} \end{array} \qquad c=(-1/2,0,1/2,1)^T \qquad (36)$$

The ***PC*** method of Sham pine and Watts was implemented by Worland [15]On two processors.

4: Parallel Block Runge-Kutta Method:-

Lets us take a block which consists of k equidistant points $x_{n+r} = x_n + rh,$ r=1, …, 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:-

$$y_{n+1}^{(0)} = y_n + hf_n \quad \textbf{(37)}$$

$$y_{n+1}^{(1)} = y_n + \frac{h}{2}(f_n + f_{n+1}^{(0)}) \quad \textbf{(38)}$$

$$y_{n+2}^{(1)} = y_n + 2hf_{n+1}^{(1)} \quad \textbf{(39)}$$

$$y_{n+1}^{(2)} = y_n + \frac{h}{3}(5f_n + 8f_{n+1}^{(1)} - f_{n+2}^{(1)}) \quad \textbf{(40)}$$

$$y_{n+2}^{(s+1)} = y_n + h/3(f_n + 4f_{n+1}^{(s)} - f_{n+2}^{(s)}), \qquad s = 1,(2) \quad \textbf{(41)}$$

The sequence (28) – (41) is used minimize the number of derivative evaluation required.

The corresponding formulas for the parallel are:

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

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

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

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.

Second processor calculates:

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

We can reduce the sequence (28) – (41) to following Runge-Kutta form:

$$y_{n+2} - y_n = \frac{h}{3}(k_1 + 4k_5 + k_6)$$

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

$$k_2 = f(x_n + h, y_n + hk_1)$$

$$k_3 = f\left(x_n + h, y_n + \frac{h}{2}k_1 + \frac{h}{2}k_2\right) \quad (42)$$

$$k_4 = f(x_n + 2h, y_n + 2k_3)$$

$$k_5 = f\left(x_n + h, y_n + \frac{h}{12}(5k_1 + 8k_3 - k_4)\right)$$

$$k_6 = f\left(x_n + 2h, y_n + \frac{h}{3}(k_1 + k_4 + 4k_5)\right)$$

This form can be converted to Parallel form as follows:

1- we produce the value $y_1 \text{ and } y_2$ sequentially or in parallel by using one or two sequential forms of Runge-kutta type methods, then the parallel calculation continues as follow:

$$y_{n+2} - y_n = \frac{h}{3}(k_1 + 4k_5 + k_6) , \qquad n = 1,3,5,... \tag{43}$$

$$y_{r+2} - y_r = \frac{h}{3}(L_1 + 4L_5 + L_6) , \qquad r = 2,4,6,...$$

Where $k_1, k_2, k_3, k_4, k_5 \text{ and } k_6$ are as given (42) and r erplace each n in the form of L's.

Form (43) is a parallel form suitable for two Processor and (43) can be easily modified for forms suitable for 3,4,5 and more processors.

5: 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) Murshed, A. A. A: "An Investigation Of Numerical Algorithms for solving stiff ODEs Suitable for parallel computers", Ph.D. thesis, University of Mosul, (2000).

5) M. A. Franklin, "Parallel solution of ordinary equations", ***IEEE Trans. On Computers*** Vol.C-27, No. %(1978), 413-420.

6) S. K. Ghoshal, M. Gupta and V. Rajaraman, "A parallel multi step predictor-corrector algorithm for solving ordinary differential equations" ***J. of Parallel and Distriuted Computing***, Vol.6 (1989), 636-648.

7) P. J. van Der Houwen and B. P. Sommeijer, "Block Runge-kutta methods on parallel computers" ***Z. Angew, Math. Mech***. 72 (1992), 3-18.

8) 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.

9) B. M. S. Khalaf, "Parallel numerical algorithms for solving ordinary differential equations" ***Ph.D. Thesis, University of Leeds, U.K.***, 1990.

10) W. L. Miranker , " A survey of parallelism in numerical analysis" , ***SIAM Review*** , Vol.13 , No. 4 (1971), 524-5247.

11) J.B. Rosser, "A Runge-Kutta for all seasons", ***SIAM Review***, Vol. 9 (1967), 417-452.

12) L. F. Sham pine and H. A. Watts, "Block implicit one – step methods" , ***Mathematics of Computation***, Vol.23 (1969), 731-740.

13) T. E. Shoup, "Applied numerical methods for the micro – computers" ***Prentice- Hall, New Jersey***, 1984.

14) D. A. Voss and S. Abbas, "Block predictor – corrector schemes for the parallel solution of ODEs", ***Computers Mathematics Application***, Vol. 6 , (1997), 65-72.

15) P. B. Worland, "Parallel methods for the numerical soluation of ordinary differential equations" ***IEEE Trans. On Computers***, (1976), 1045-1048.