SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

PARALLEL SOLUTION OF ODE's BY MULTI-BLOCK METHODS*

MOODY T. CHU† AND HANS HAMILTON†

Abstract. The notion of linear multi-step methods for solving ordinary differential equations is generalized to a class of multi-block methods. In a multi-block method step values are all obtained together in a single block advance which is accomplished by allocating the parallel tasks on separate processors. The expected benefit of multi-block methods is the speedup in the computation of solutions. The basic formulation is described. Examples are given to demonstrate the existence of such schemes. The predictor-corrector type combination is formed and the resulting stability problem is considered. Test results of one of these multi-block methods on the Denelcor HEP machine are reported.

Key words. multi-block methods, parallel processing, absolute stability, Denelcor HEP

AMS(MOS) subject classification. 65L05

1. Introduction. We are interested in the problem of speeding up the numerical integration of a system of ordinary differential equations by parallel computations. We shall study a class of multi-block methods and report the test results made on the Denelcor HEP machine.

Parallel methods for numerical solutions of systems of ordinary differential equations have been proposed by various researchers. Generally speaking, the parallelism may be achieved by partitioning the tasks either across the algorithm or across the system of equations. In the review paper [4], Franklin exemplified this idea with three classes of parallel algorithms: the equation segmentation methods used quite heavily by engineers in the field of dynamic system simulation, the parallel predictor-corrector methods developed by Miranker and Liniger [8], and the parallel block implicit methods considered by Shampine and Watts [13], [14], Rosser [12], and Worland [15]. Other methods designed or adaptable for parallel processing also exist. Readers can find them in, for example, papers by Birta and Abou-Rabia [2], Donelson and Hansen [3], Mitra [9], Nievergelt [10], Andria et al. [1], and also several other papers cited by Poole and Voigt in [11].

In this paper, we consider parallel block methods. The procedures, described as k-block, r-point methods, generalize those developed by Shampine and Watts in the sense that their schemes correspond to a special case of our 1-block, r-point methods. Also, the cyclic composite multi-step methods originally proposed by Donelson and Hansen [3] for serial computations to circumvent the very real barrier in the condition of zero-stability for multi-step methods, can be easily reformulated in terms of our multi-block settings.

In a k-block, r-point method, time is divided into a series of blocks with each block containing r steps at which solutions to differential systems are to be found; the computation proceeds in blocks, i.e., the computation is based on the computed values at the earlier k blocks. Within a block it is possible to assign the computational tasks at each time step to a single processor and, thus, the computations at all future steps can be performed simultaneously in parallel. As a consequence, speedup in the integration and the function evaluation is expected.

The basic setting of a general k-block r-point method is discussed in § 2 where we also give four examples to show the feasibility of this approach. Section 3 is devoted to the concept of predictor-corrector methods. In § 4 we examine the absolute stability

* Received by the editors May 29, 1985, and in revised form December 23, 1985.
† Department of Mathematics, North Carolina State University, Raleigh, North Carolina 27695-8205.

properties of block methods. Some experimental results are included there. Finally, we discuss the implementation and testing of one particular block method on the Denelcor HEP machine. The test results are reported in § 5.

2. Multi-block methods. We shall be interested in obtaining a numerical solution of

$$\begin{aligned} y'(x) &= f(x, y(x)), \\ y(a) &= y_0, \qquad a \leqq x \leqq b, \end{aligned} \tag{2.1}$$

where we make the usual assumption that f is continuous and satisfies a Lipschitz condition on the region $[a, b] \times (-\infty, \infty)$. The discussions in what follows can be generalized to higher dimensional systems in an obvious way.

For $h \in (0, h_0)$ let $x_k = a + kh$, y_k represent a numerical approximation to the exact solution $y(x_k)$ and $f_k = f(x_k, y_k)$. We also introduce $m = 0, 1, 2, \cdots$ representing the block number, $n = mr$ the first step number in the m-th block, and r the proposed block size (usually the same as the number of processors).

DEFINITION 2.1.1. Let Y_m and F_m be vectors defined by

$$Y_m = [y_n, y_{n+1}, \cdots, y_{n+r-1}]^T, \tag{2.2}$$

$$F_m = [f_n, f_{n+1}, \cdots, f_{n+r-1}]^T. \tag{2.3}$$

Then a general k-block, r-point method is a matrix finite difference equation of the form

$$Y_m = \sum_{i=1}^{k} A_i Y_{m-i} + h \sum_{i=0}^{k} B_i F_{m-i}, \tag{2.4}$$

where all A_i's and B_i's are certain properly chosen $r \times r$ matrix coefficients.

If $r = 1$, then (2.4) is just the classical k-step method. When $B_0 = 0$, we say that (2.4) is an explicit method; otherwise it is implicit. Note that when $r > 1$, there is a combinatorial question of parcelling out time steps among blocks. For instance, when $r = 2$, one can define the block by either $Y_m = [y_{2m}, y_{2m+1}]^T$ as in (2.2) or $Y_m = [y_{2m+1}, y_{2m+2}]^T$ as in [13], [14]. The possible significant effect of these different block definitions is reflected only in the necessary starting procedures where one definition may demand more starting values than the other, but not in the successive block advancing procedures.

DEFINITION 2.1.2. Let $y(x)$ be the exact solution of (2.1) and

$$Z_m = [y(x_n), y(x_{n+1}), \cdots, y(x_{n+r-1})]^T. \tag{2.5}$$

Then the local truncation error E_m of the scheme (2.4) at the mth block is defined to be

$$E_m = Z_m - \sum_{i=1}^{k} A_i Z_{m-i} - h \sum_{i=0}^{k} B_i Z'_{m-i}. \tag{2.6}$$

There are several ways to determine the matrix coefficients involved in (2.4); the Taylor expansion method and the method of undetermined coefficients are probably the most straightforward ones. When applying any of these methods to derive the scheme (2.4), one may often suppress the full usage of the information of certain blocks by setting certain columns of the corresponding matrix coefficients to be identically zero. This action is especially suitable when the desired order of accuracy is much lower than what would have been achieved if all steps inside the blocks were active. We now give some examples below to demonstrate the existence of schemes of the form (2.4). Also listed are the corresponding principal local truncation errors.

Example 1. $k=2$, $r=2$, explicit, partial usage of block values.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 0, & 1+a_1+a_2 \\ 0, & 1+a_3+a_4 \end{bmatrix} Y_{m-2} + \begin{bmatrix} -a_1, & -a_2 \\ -a_3, & -a_4 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{12}\begin{bmatrix} 0, & 9+5a_1+4a_2 \\ 0, & 32+5a_3+4a_4 \end{bmatrix} F_{m-2} \\
&+ \frac{h}{12}\begin{bmatrix} 8a_1+16a_2, & 27-a_1+4a_2 \\ -64+8a_3+16a_4, & 80-a_3+4a_4 \end{bmatrix} F_{m-1},
\end{aligned} \tag{2.7}
$$

$$
\text{Local error} = \frac{1}{72}\begin{bmatrix} (27+3a_1)h^4y^{(4)} \\ (192+3a_3)h^4y^{(4)} \end{bmatrix}. \tag{2.8}
$$

Example 2. $k=1$, $r=2$, implicit, lower order of accuracy.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1, & -a_1 \\ 1+a_2, & -a_2 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{72}\begin{bmatrix} 56+46a_1-24b_1, & 72b_1 \\ 27+27a_2, & 81+57a_2 \end{bmatrix} F_{m-1} \\
&+ \frac{h}{72}\begin{bmatrix} 120+42a_1-72b_1, & -32-16a_1+24b_1 \\ 81-15a_2, & 27+3a_2 \end{bmatrix} F_m,
\end{aligned} \tag{2.9}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (320+190a_1-240b_1)h^4y^{(4)} \\ (-27-19a_2)h^5y^{(5)} \end{bmatrix}. \tag{2.10}
$$

Example 3. $k=2$, $r=2$, explicit, full usage of block values.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1+a_2+a_3, & -a_1 \\ 1+a_4+a_5+a_6, & -a_4 \end{bmatrix} Y_{m-2} + \begin{bmatrix} -a_2, & -a_3 \\ -a_5, & -a_6 \end{bmatrix} Y_{m-1} \\
&+ \frac{h}{24}\begin{bmatrix} 9a_1+8a_2+9a_3, & 64+19a_1+32a_2+27a_3 \\ -55+9a_4+8a_5+9a_6, & 275+19a_4+32a_5+27a_6 \end{bmatrix} F_{m-2} \\
&+ \frac{h}{24}\begin{bmatrix} -32-5a_1+8a_2+27a_3, & 64+a_1+9a_3 \\ -325-5a_4+8a_5+27a_6, & 225+a_4+9a_6 \end{bmatrix} F_{m-1},
\end{aligned} \tag{2.11}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (224-19a_1-8a_2-27a_3)h^5y^{(5)} \\ (2125-19a_4-8a_5-27a_6)h^5y^{(5)} \end{bmatrix}. \tag{2.12}
$$

Example 4. $k=1$, $r=2$, implicit, higher order of accuracy.

$$
\begin{aligned}
Y_m = &\begin{bmatrix} 1+a_1, & -a_1 \\ 1+a_2, & -a_2 \end{bmatrix} Y_{m-1} + \frac{h}{24}\begin{bmatrix} 8+9a_1, & 32+19a_1 \\ 9+9a_2, & 27+19a_2 \end{bmatrix} F_{m-1} \\
&+ \frac{h}{24}\begin{bmatrix} 8-5a_1, & a_1 \\ 27-5a_2, & 9+a_2 \end{bmatrix} F_m,
\end{aligned} \tag{2.13}
$$

$$
\text{Local error} = \frac{1}{720}\begin{bmatrix} (-8-19a_1)h^5y^{(5)} \\ (-27-19a_2)h^5y^{(5)} \end{bmatrix}. \tag{2.14}
$$

It is interesting to note that the predictor used in [13] is a special case of (2.7) with parameter values $a_1=a_2=a_3=a_4=-\frac{1}{3}$, and the well-known Clippinger and Dimsdale formula used as the corrector in [13] corresponds to (2.9) with parameter values $a_1=a_2=-1$ and $b_1=\frac{5}{12}$.

It was pointed out in [13] that the stability problem would appear to be the most serious limitation of block methods when used as a predictor-corrector type combination. We shall see in § 4 that by taking advantage of the free parameters of the preceding formulas, the stability properties can be greatly improved.

The procedure (2.4) may be regarded as a package of r multi-step methods (possibly with different step numbers) working independently. The convergence of this procedure, therefore, is assured by the standard arguments for multi-step methods. We shall omit the formal proof of convergence in this presentation.

3. Predictor-corrector methods. Based on the usual setting for multi-step methods, one can easily set up the analogies of PC modes for multi-block methods. Henceforth, let the quantities related to the predictor be denoted by the mark "*". Then the P(EC)^{t}E mode, for example, is defined as follows:

$$
\begin{aligned}
&\text{P: } Y_m^{(0)} = \sum_{i=1}^{k^*} A_i^* Y_{m-i}^{(t)} + h \sum_{i=1}^{k^*} B_i^* F_{m-i}^{(t)};\\
&\text{E: } F_m^{(s)} = [f(x_n, y_n^{(s)}, \cdots, f(x_{n+r-1}, y_{n+r-1}^{(s)})]^T,\\
&\text{C: } Y_m^{(s+1)} = \sum_{i=1}^{k} A_i Y_{m-i}^{(t)} + h \sum_{i=1}^{k} B_i F_{m-i}^{(t)} + hB_0 F_m^{(s)} \quad \text{for } s = 0, \cdots, t-1,\\
&\text{E: } F_m^{(t)} = [f(x_n, y_n^{(t)}), \cdots, f(x_{n+r-1}, y_{n+r-1}^{(t)})]^T.
\end{aligned}
\tag{3.1}
$$

Let us consider now the effect of such a combination on the local truncation error. Under the localizing assumption (i.e., assuming that $Z_{m-i} = Y_{m-i}^{(t)}$ for all $i = 1, 2, \cdots, k$), it can be shown that

$$
\begin{aligned}
Z_m - Y_m^{(s+1)} &= hB_0(Z'_m - F_m^{(s)}) + E_m\\
&\approx hLB_0(Z_m - Y_m^{(s)}) + E_m,
\end{aligned}
\tag{3.2}
$$

where L is bounded by the Lipschitz constant of f with respect to y. It is seen that formally the local truncation error of this multi-block algorithm behaves in exactly the same way as that of the multi-step algorithm. For the discussion of the latter case readers are referred to [6]. The expression (3.2) gives us a practical idea of choosing the predictor and the corrector when forming the combination (3.1). Let us consider the simplest case when $t = 1$ as an example. Then (3.2) suggests that one should use a predictor in which all components have the same order of accuracy, because the multiplication by the matrix B_0 will usually mix up all components and, hence, contaminate the superior accuracy if there is any. It also suggests that for economical purposes one should use a corrector in which each component has equal or one order higher accuracy than the predictor. In this case, the local truncation error of the PECE mode is of the same order as the corrector. Finally, it suggests that restricting oneself to the case $k^* = k+1$ is generally sufficiently for the practical purpose.

4. Stability. From now on let it be assumed that $k^* = k+1 = p$. We find that the following definitions of vectors facilitate discussions of the stability problem.

Let $U_m^{(s)}$ and $V_m^{(s)}$ for $s = 0, 1, \cdots, t$ be $pr \times 1$ vectors defined by

$$
U_m^{(s)} = \begin{bmatrix} Y_m^{(s)} \\ Y_{m-1}^{(t)} \\ \vdots \\ Y_{m-p+1}^{(t)} \end{bmatrix}, \qquad V_m^{(s)} = \begin{bmatrix} F_m^{(s)} \\ F_{m-1}^{(t)} \\ \vdots \\ F_{m-p+1}^{(t)} \end{bmatrix}.
\tag{4.1}
$$

It is easy to see that (3.1) is equivalent to the following scheme:

$$U_m^{(0)} = \begin{bmatrix} A_1^* & A_2^* & \cdots & A_{p-1}^* & A_p^* \\ I & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ & & & 0 & 0 \\ 0 & \cdots & & I & 0 \end{bmatrix} U_{m-1}^{(t)}$$

$$+ h \begin{bmatrix} B_1^* & B_2^* & \cdots & B_{p-1}^* & B_p^* \\ 0 & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & & \cdots & & 0 \end{bmatrix} V_{m-1}^{(t)} \tag{4.2}$$

$$\equiv \mathbf{A}^* U_{m-1}^{(t)} + h\mathbf{B}^* V_{m-1}^{(t)};$$

$$U_m^{(s+1)} = \begin{bmatrix} A_1 & A_2 & \cdots & A_{p-1} & 0 \\ I & 0 & & 0 & 0 \\ 0 & & & & \vdots \\ \vdots & & & 0 & 0 \\ 0 & \cdots & & I & 0 \end{bmatrix} U_{m-1}^{(t)}$$

$$+ h \begin{bmatrix} B_0 & B_1 & \cdots & B_{p-2} & B_{p-1} \\ 0 & 0 & & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & & \cdots & & 0 \end{bmatrix} V_m^{(s)} \tag{4.3}$$

$$\equiv \mathbf{A} U_{m-1}^{(t)} + h\mathbf{B} V_m^{(s)},$$

where $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ are $pr \times pr$ square matrices defined accordingly, and I represents an $r \times r$ identity matrix. When applying this scheme to the model problem

$$y' = \lambda y \quad \text{with} \quad \lambda < 0 \tag{4.4}$$

and letting $\mathbf{h} = \lambda h$, we find that

$$U_m^{(0)} = (\mathbf{A}^* + \mathbf{hB}^*) U_{m-1}^{(t)}, \tag{4.5}$$

$$U_m^{(s+1)} = \mathbf{A} U_{m-1}^{(t)} + \mathbf{hB} U_m^{(s)}, \tag{4.6}$$

and hence

$$\begin{aligned} U_m^{(t)} &= \{[I + (\mathbf{hB}) + \cdots + (\mathbf{hB})^{t-1}]\mathbf{A} + (\mathbf{hB})^t(\mathbf{A}^* + \mathbf{hB}^*)\} U_{m-1}^{(t)} \\ &\equiv T(\mathbf{h}; \mathbf{A}, \mathbf{B}, \mathbf{A}^*, \mathbf{B}^*) U_{m-1}^{(t)}. \end{aligned} \tag{4.7}$$

From (4.7) it is clear that the PC block algorithm (3.1) is absolutely stable for a value $\mathbf{h}$ if and only if

$$s(T(\mathbf{h}; \mathbf{A}, \mathbf{B}, \mathbf{A}^*, \mathbf{B}^*)) < 1 \tag{4.8}$$

where $s(T)$ represents the spectral radius of the matrix T.

As we have seen earlier, in accordance with certain prescribed orders of accuracy, generally one can determine the components in the matrix coefficients $\mathbf{A}$, $\mathbf{B}$, $\mathbf{A}^*$, $\mathbf{B}^*$

up to some free parameters. These parameters are not totally free in practice because some of them must satisfy the constraint of the so called zero-stability.

DEFINITION 4.1. A multi-block method (2.4) is said to be zero-stable if solutions of the equation

$$Y_m = \sum_{i=1}^{k} A_i Y_{m-i} \tag{4.9}$$

remain bounded for all m.

In terms of (4.2) and (4.3), the zero-stability is equivalent to, respectively, that there are no eigenvalues of $\mathbf{A}^*$ and $\mathbf{A}$ lying outside the unit circle and that the eigenvalues on the unit circle have simple elementary divisors.

Now an interesting optimization problem has arisen: Given $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ being determined up to some free parameters (so that a certain prescribed order of accuracy is fulfilled), we want to find the infimum of $\mathbf{h}$ from all possible parameter values which are subject to the constraint that the corresponding matrices $\mathbf{A}^*$, $\mathbf{B}^*$, $\mathbf{A}$ and $\mathbf{B}$ satisfy $s(T(\mathbf{h}; \mathbf{A}^*, \mathbf{B}^*, \mathbf{A}, \mathbf{B})) < 1$, and that $\mathbf{A}$ is zero-stable. (It turns out that the zero-stability of $\mathbf{A}^*$ is immaterial.) This problem is by no means simple because both the objective and the constraint functions are implicitly described, and can be shown numerically to possess many discontinuous or nondifferentiable points. Some experimental results concerning the left endpoint H_B of the interval of absolute stability for the PECE mode using (2.7) as the predictor and (2.9) as the corrector are collected in Table 1. As is seen, the PC pair studied by Shampine and Watts in [13] is only a special case of a 7-parameter family of methods and certainly does not give the best absolute stability. Table 2 contains similar results for the same mode using (2.11) as the predictor and (2.13) as the corrector.

Note that the PC pair of (2.11) and (2.13) constitutes a fourth order block method. At the time this paper was written, the best stability boundary we had obtained was

TABLE 1

P				C			PECE
a_1	a_2	a_3	a_4	a_1	a_2	b_1	H_B
−1/3	−1/3	−1/3	−1/3	−1.0	−1.0	5/12	−0.439
−1/3	−1/3	−1/3	−1/3	−1.0	0.0	5/12	−0.848
2.0	−7/3	−1/3	−1/3	−1.0	−1/5	7/12	−0.955
7/3	−8/3	−1/3	−1/3	−1.0	−3/5	11/12	−1.130
7/3	−8/3	−1/3	−1/3	−1.0	−3/5	1.0	−1.249

TABLE 2

P						C		PECE
a_1	a_2	a_3	a_4	a_5	a_6	a_1	a_2	H_B
−1.000	−0.667	−0.333	−1.000	−0.667	−0.667	−0.500	−1.000	−0.311
−1.000	−0.667	−0.333	−1.000	−0.667	−0.667	−1.000	−0.500	−0.603
−1.233	−0.567	−0.333	−1.133	−0.567	−0.567	−0.999	−0.200	−0.680
−1.333	−0.567	−0.133	−1.333	−0.467	−0.567	−1.000	−1.000	−0.692
−1.233	−0.467	−0.033	−1.333	−0.767	−0.367	−1.000	−1.000	−0.740

$H_B = -0.810$ by setting

$$(4.10) \qquad \text{P:} \begin{cases} a_1 = -1.23000, \\ a_2 = -0.46300, \\ a_3 = -0.02770, \\ a_4 = -1.33000, \\ a_5 = -0.77212, \\ a_6 = -0.37000, \end{cases} \qquad \text{C:} \begin{cases} a_1 = -1.01000, \\ a_2 = -0.04520. \end{cases}$$

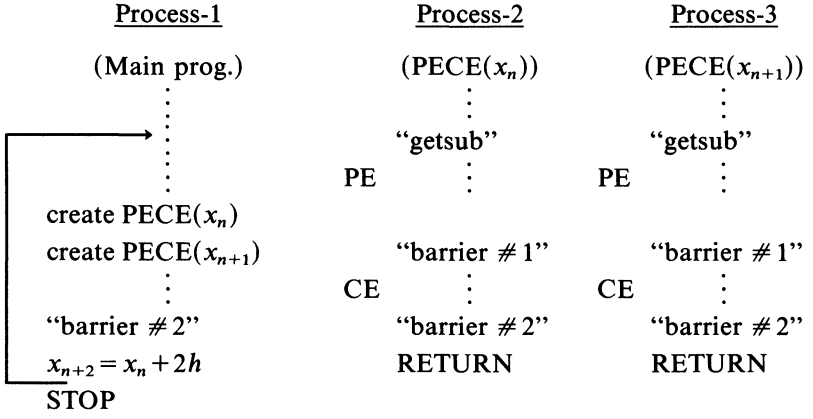
We recall that the stability boundary H_{ABM} of the fourth order Adams-Bashforth-Moulton (ABM) pair in PECE mode is $H_{\text{ABM}} = -1.285$ [5]. Thus, it seems that our multi-block method is severely restricted by its stability property and is not competitive with the fourth order ABM method. One must be aware, however, that each call of an r-point, multi-block method with step size h advances the time by the total length rh. Consider the case $r = 2$. Suppose the length of integration for a particular problem is T and both methods take the maximum allowable stepsize for this problem. Then the ABM method will take T/H_{ABM} steps and $2T/H_{\text{ABM}}$ function evaluations. The multi-block method will take $T/(2H_B)$ blocks and $2T/H_B$ function evaluations. Although the multi-block method uses more function evaluations than the ABM method, the parallel nature of the function evaluation effectively halves the time needed for these evaluations, i.e. it will need only T/H_B real function evaluation time. This argument is applicable to general comparisons and of compatible (same order, same mode, etc.) multi-step and multi-block methods.

The data we have furnished in (4.10) are certainly not conclusive. We propose to continue experimenting with that optimization problem for other multi-block schemes. Before ending, we note that a similar problem of optimizing the stability region for a class of parallel predictor-corrector formulas developed in [8] has been studied by Katz et al. [5]. They showed that if the corrector is fixed to be the 4th order Adams-Moulton corrector then the optimally stable parallel predictor is the Adams-Bashforth predictor shifted to the right by one integration step. In this case, the left endpoint of the interval of absolute stability is -0.846. It is also worth noting that in [14] Watts and Shampine derived a class of implicit 1-block methods from Newton-Cotes type interpolatory formulas and conclusively established that these block methods for size $r = 1, 2, \cdots, 8$ are A-stable whereas those for $r = 9, 10$ are not.

5. Implementation on HEP. A prototype code of the PECE scheme from formulas (2.11) and (2.13) has been implemented with fixed stepsize to study the real-time savings in using a multi-processor machine. The machine used was the Denelcor HEP at Argonne National Laboratory. Before examining the results of our tests, let us briefly consider some details of the HEP and how our code was implemented to take advantage of the structure of the HEP. Obviously, most of the ideas discussed in the following about the interprocess communication and synchronization should serve as general patterns on other multi-processing systems as well.

The HEP (Heterogeneous Element Processor) is a large-scale scientific multi-processor system which can execute a number of sequential (SISD) or parallel (MIMD) programs simultaneously. The HEP supports an enhanced version of Fortran-77 which allows for parallel execution of multiple tasks within a single program. These extensions are implemented in HEP-Fortran, but are most commonly used through higher level monitors (macros) supplied by the installation at Argonne. These macros allow for the scheduling of multiple tasks, and other associated concepts useful in multi-processing. Readers are referred to [7] for more detailed discussions.

Various tasks to be done on the HEP are referred to as "processes". The objective of the macros is to schedule the execution of the processes and provide the necessary synchronization between these processes. The macros "getsub" and "barrier" used in our code accomplish these two jobs. Recall that (2.11) predicts at x_n and x_{n+1} and (2.13) corrects at x_n and x_{n+1}. The PECE at x_n is handled by one process while PECE at x_{n+1} is handled by a second process. The main program represents a third process. The allocation of PECE at x_n or x_{n+1} to a process is handled by the "getsub" macro. Since the corrector step requires the function evaluation from the predictor step, the "barrier" macro is used to synchronize the process after PE has been completed by both processes. The "barrier" is again used to synchronize processes after CE has been completed. This may be visualized as follows:



In the above, Process-2 and -3 may be viewed as subroutines running concurrently with Process-1, the main program. The "create" statements are equivalent to a "CALL" in Fortran except that execution continues after the statement is encountered. This provides the concurrent execution of the processes. The "getsub" is contained in subroutine PECE and assigns the process the x_n or x_{n+1} step. The "barrier" statements cause the processes to "wait" until all other processes in that category have completed their work before continuing. The "RETURN" statements cause both Process-2 and -3 to "die" off and the main program to prepare for another block of two steps.

The problem we tested on the HEP was

$$y' = -2y, \qquad y(0) = 1. \tag{5.1}$$

This equation was duplicated to simulate higher-dimensional problems as well as three types of overhead demand of the derivative evaluation routine. This is a "make-work" problem, but suited our purpose better than any specific problem we could find.

The timing on the HEP is handled by the "clock" macro where time is measured in increments of 100 nanoseconds. The performance figures we have reported include only the call to the integrator. I/O and other "housekeeping" chores are not included. The serial code is simply the code that runs on our IBM-3081 system, i.e. it uses only 1 process on the HEP.

Each type of problem has been run with different integration interval lengths with the dimension being varied between 1 and 128 in powers of 2 for each interval. It turns out that the average time per unit length of integration is fairly constant for each dimension. Therefore, we shall report the average behavior only. Theoretically, with three processes we would expect a speedup of 66%, but actually only two processes

are heavily loaded while the main program is idle most of the time. Hence any speedup should be closer to 50%.

In the Type-1 problem we had the number of flops (one flop means either an addition or multiplication) per component in the derivative evaluation match the number of components. In Fig. 1 we have graphed time vs. dimension for the serial and parallel methods. As expected, time behaves like $O(\dim^2)$. The relation of speedup vs. dimension is shown in Fig. 2. As the dimension grows the speedup nears the theoretical 50%. To determine the cause of the less than optimal speedup for the lower-dimensional problems we then run the other two types of problems.

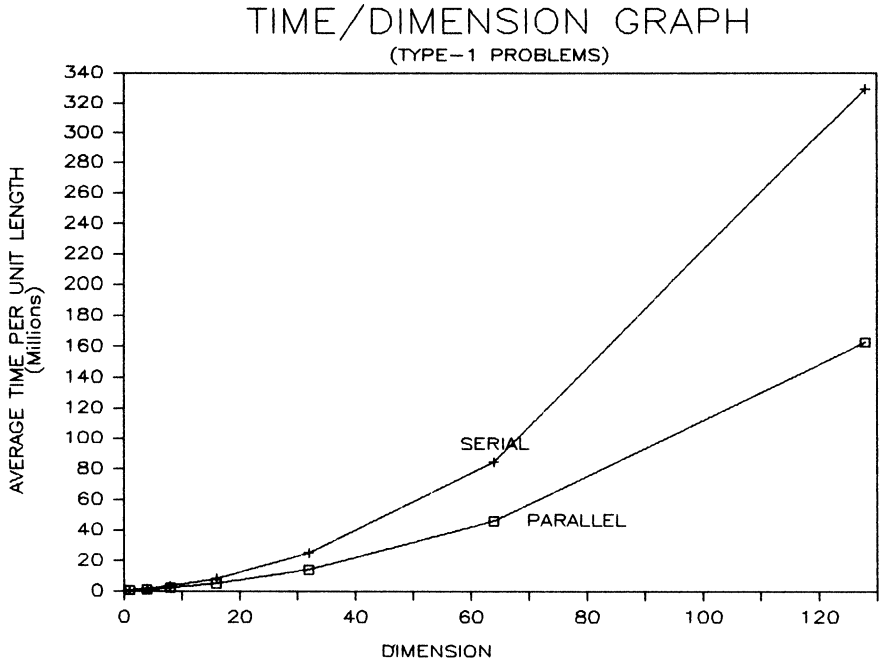


FIG. 1

In the Type-2 problem the number of flops was fixed at a very high level so that the cost of the derivative evaluation would be approximately the same for each dimension. The relation of time vs. dimension for this problem is shown in Fig. 3. In Fig. 4 we have speedup vs. dimension. The speedup is approximately 47% for all the dimensions.

In the Type-3 problem we fixed the number of flops per component at a relatively high level so that time would behave as $O(\dim)$ as the dimension varied. This result is shown in Fig. 5. Speedup vs. dimension is about the same as the Type-2 problem, so Fig. 3 is applicable to this version also.

The tests on the last two types of problems indicate that the theoretical speedup becomes possible when the derivative evaluation cost is high. In the Type-1 problem the derivative evaluation cost was low for the low-dimensional problems, so overhead costs in creating and synchronizing the processes decreased the speedup.

6. Conclusion. In the above we have discussed a class of multi-block methods for solving systems of ordinary differential equations. These block methods appears to be well suited to computers with parallel architectures and, hence, are expected to enhance

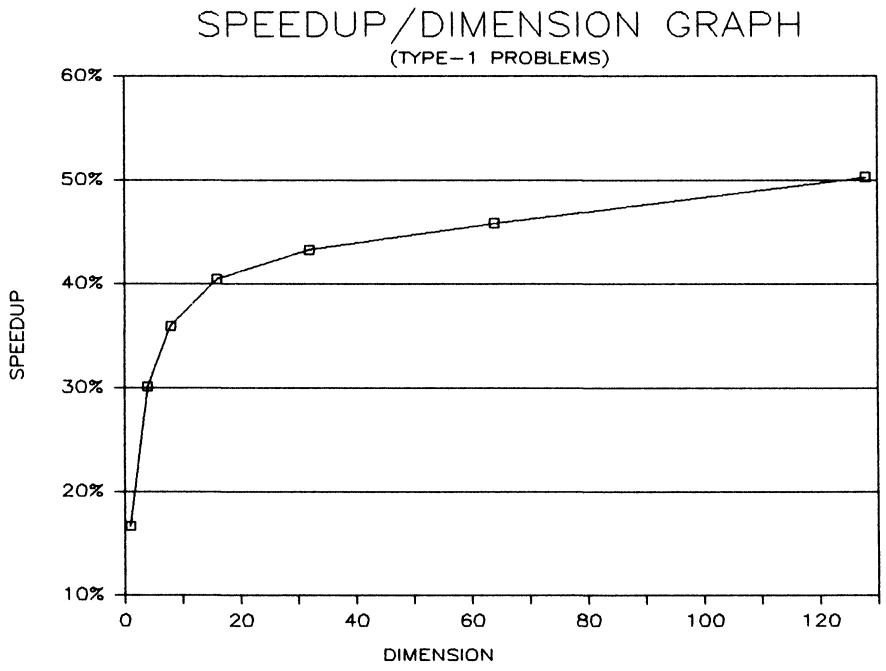


FIG. 2

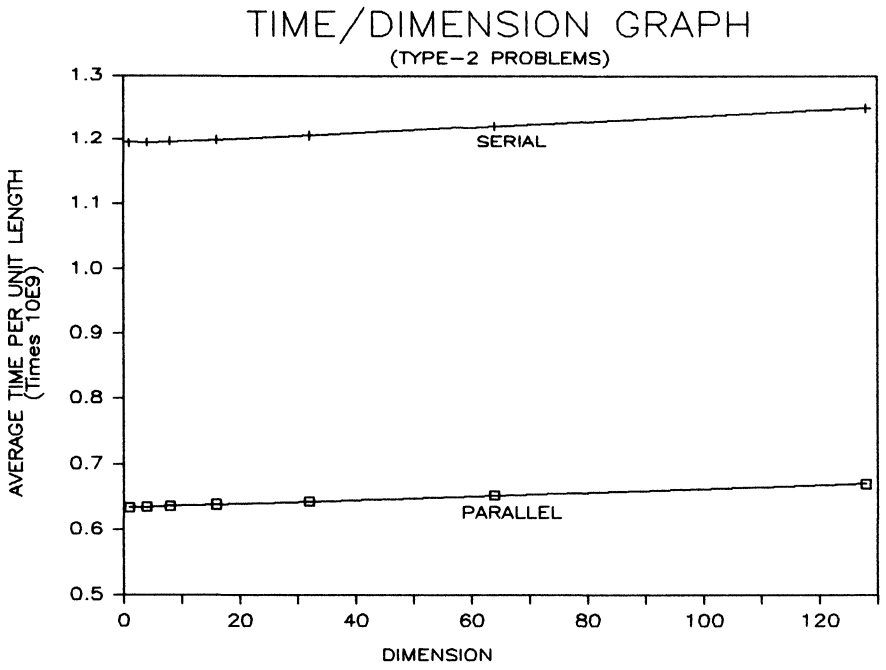


FIG. 3

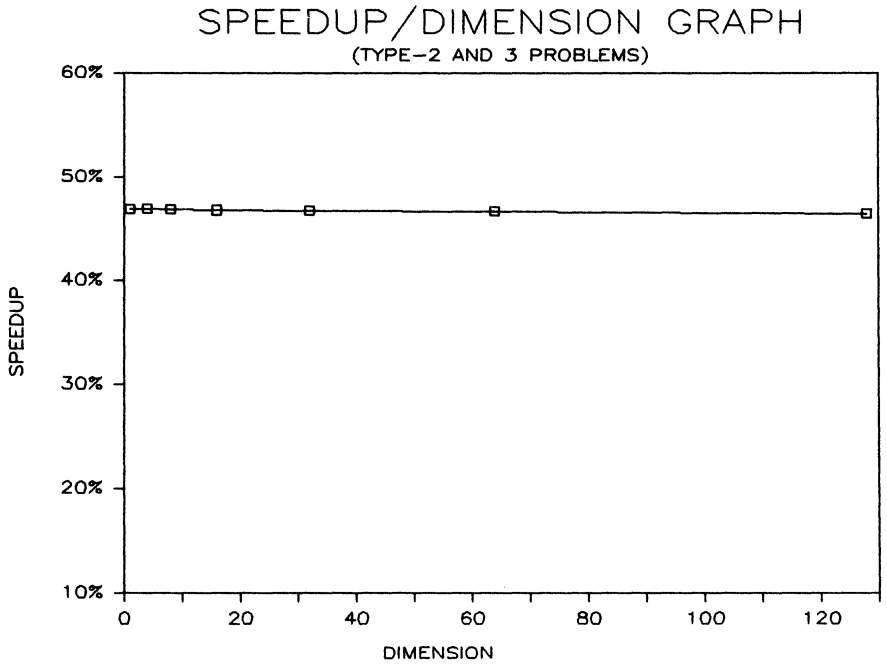


FIG. 4

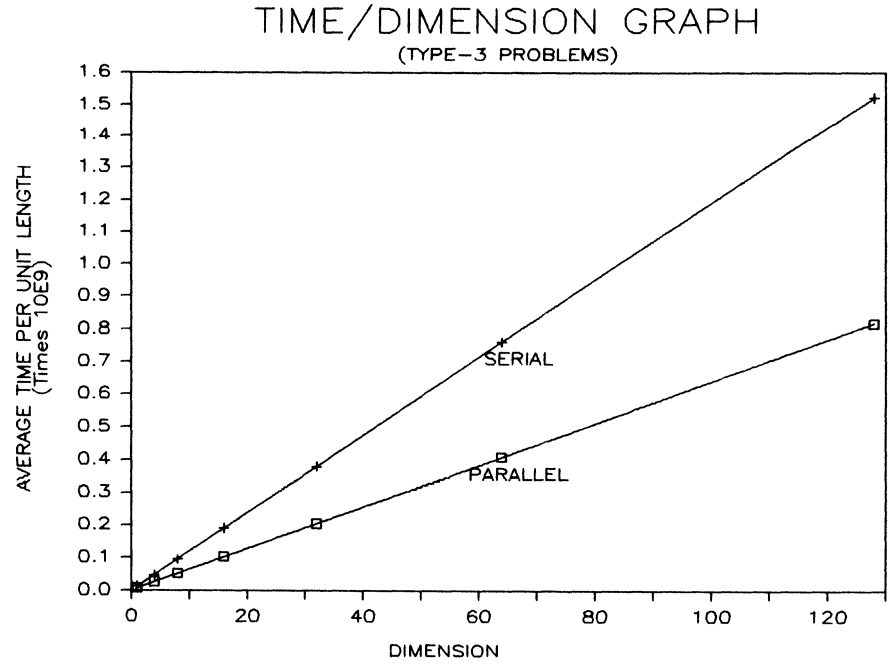


FIG. 5

the speedup of the computation of numerical solutions. The following conclusions are drawn from experiments that we have conducted thus far:

(1) The stability region, though smaller than that of a compatible Adams-Bashforth-Moulton method, is not a restriction as severe as has been reported in the literature. In fact, we have found sets of data which make the corresponding multi-block methods competitive as far as the absolute stability and the accuracy are concerned.

(2) The theoretical speedup 50% can almost be achieved on an MIMD machine, such as the Denelcor HEP system, when the cost of derivative evaluation is high. This is true even for lower-dimensional problems. When the cost of derivative evaluation is low, the overhead costs with creating and synchronizing the processes may have the strongest effect in degrading the performance of speeding-up. The total speedup, however, is still significant.

Finally, since the algorithm we have written is highly vectorizable, it is desirable to experiment with a multi-processor computer which has vector instructions. One such a machine is the CRAY X-MP. When this becomes available we shall report the results elsewhere.

REFERENCES

[1] G. D. ANDRIA, G. D. BYRNE AND D. R. HILL, *Natural spline block implicit methods*, BIT, 13 (1973), pp. 131-144.

[2] L. G. BIRTA AND O. ABOU-RABIA, *A multi-microprocessor system for continuous system simulation*, Technical Report TR8308, Dept. of Computer Science, Univ. Ottawa, Ontario, Canada, 1983.

[3] J. DONELSON AND E. HANSEN, *Cyclic composite multistep predictor-corrector methods*, SIAM J. Numer. Anal., 8 (1971), pp. 137-157.

[4] M. A. FRANKLIN, *Parallel solution of ordinary differential equations*, IEEE Trans. Comput., C-27 (1978), pp. 413-420.

[5] I. N. KATZ, M. A. FRANKLIN AND A. SEN, *Optimally stable parallel predictors for Adams-Moulton correctors*, Comput. Math. Appl., 3 (1977), pp. 217-233.

[6] J. D. LAMBERT, *Computational Methods In Ordinary Differential Equations*, John Wiley, London, 1973.

[7] E. L. LUSK AND R. A. OVERBEEK, *Implementation of monitors with macros: A programming aid for the* HEP *and other parallel processors*, Technical Report ANL-83-97, Argonne National Laboratory, Argonne, IL.

[8] W. L. MIRANKER AND W. M. LINIGER, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comput., 21 (1967), pp. 303-320.

[9] D. MITRA, *Chaotic, asynchronous relaxations for the numerical solution of differential equations by parallel processors*, preprint, AT&T Bell Laboratories, Murray Hill, NJ, 1985.

[10] J. NIEVERGELT, *Parallel methods for integrating ordinary differential equations*, Comm. ACM, 7 (1964), pp. 731-733.

[11] W. G. POOLE AND R. G. VOIGT, *Numerical algorithms for parallel and vector computers; An annotated bibliography*, Computing Rev., 15 (1974), pp. 379-388.

[12] J. ROSSER, *A Runge-Kutta for all seasons*, SIAM Rev., 9 (1967), pp. 417-452.

[13] L. F. SHAMPINE AND H. A. WATTS, *Block implicit one-step methods*, Math. Comp., 23 (1969), pp. 731-740.

[14] H. A. WATTS AND L. F. SHAMPINE, *A-stable block implicit one-step methods*, BIT, 12 (1972), pp. 252-266.

[15] P. B. WORLAND, *Parallel methods for the numerical solution of ordinary differential equations*, IEEE Trans. Comput., C-25 (1976), pp. 1045-1048.