NASA Technical Memorandum 102498
ICOMP-90-03

Parallel Algorithms for Boundary Value Problems

Avi Lin
Temple University
Philadelphia, Pennsylvania

and Institute for Computational Mechanics in Propulsion
Lewis Research Center
Cleveland, Ohio

January 1990

NASA

INSTITUTE FOR COMPUTATIONAL MECHANICS IN PROPULSION
NASA LEWIS RESEARCH CENTER
ICOMP
CASE WESTERN RESERVE UNIVERSITY

PARALLEL ALGORITHMS FOR BOUNDARY VALUE PROBLEMS

Avi Lin*
Department of Mathematics
Temple University
Philadelphia, Pennsylvania 19122

and Institute for Computational Mechanics in Propulsion
Lewis Research Center
Cleveland, Ohio 44135

1 Abstract

In the present paper we discuss a general approach to solve Boundary Value Problems numerically in a parallel environment, The basic algorithm consists of two steps: The local step where all the P available processors work in parallel, and the global step where one processor solves a tridiagonal linear system of the order P. The main advantages of this approach are two fold: First - this suggested approach is very flexible, especially in the local step and thus the algorithm can be used with *any* number of processors and with *any* of the SIMD or MIMD machines. Secondly - the communication complexity is very small and thus can be used as easily with shared memory machines. Several examples for using this strategy are discussed.

*Work partially funded by Space Act Agreement C-99066-G.

2 Preliminaries

One of the main problems that appears in mathematical computing when a new concept of computer hardware arrives is how to redesign efficiently the existing numerical schemes, or how to invent new numerical schemes so as to take advantage of the special capabilities of the new machine. Now it is quite evident that the forthcoming computers in the present decade will have substantial capabilities of parallelism. The present paper deals with reformulation of numerical schemes for boundary value problems (BVP hereafter), taking advantage of the availability of the parallel computers on the market, and trying to use their features to the full extent. Numerical methods for solving BVPs are well established. These methods were designed mainly for serial computers and have been tested intensively because of their relative importance: not only because they appear in many physical fields as the basic governing equations, but also because they appear as a sub-problem of solving Partial Differential Equations of the elliptic and parabolic type [Lin (1986d)]. The parallel algorithms for BVPs that will be presented here are based on the general strategy for creating high order three point finite difference (FD hereafter) numerical schemes that were presented before [Lin (1986a) and Lin (1986b)] and on the new parallel technique for solving inherent serial problems [Lin (1986c)]. These two elements will be briefly discussed later. Several different parallel engines are already in use by the scientific community and many more are coming along the line. With such a wide variety of parallel computer systems and architectures, the challenge is to develop parallel algorithms that are efficient and portable from one machine to the other. There are several possibilities for doing this for a given algorithm [Gannon et al (1985)], but expressing the algorithm in terms of modules with high level of granularity is optimal in some sense of portability. The algorithms that will be presented hereafter are well suited for most of the machines, keeping their features unchanged. However, in order to make the presentation and the analysis easy, we'll refer to a certain type of a parallel engine model.

The parallel engine model for which the present algorithms are easily applied is of the SIMD (single instruction stream multiple data stream) type [Flynn (1966)] as well as of the MIMD (multiple instruction multiple data) type, with any number of processor-elements (PE) P, where each PE

- has its own ALU (arithmetic logic unit) to perform standard calculations.
- may have its own local memory.
- can be masked by a disable signal, for leaving it idle during some period of time.

Although the categories of MIMD and SIMD are too crude, we can find generally two types of machines that are considered in the literature:

- the shared memory machines (where a common memory is available for all the PEs).
- the interconnection network machines (where all the PEs are connected via a specific network).

Parallel machines of the second type appear to be more realistic [Dekel et al (1981), the INTEL Hypercube machine - Seitz (1985) and the IBM RP3 machine - Pfister et al (1985)], mainly because the number of connections per PE is small ($\log P$ for the Cube network and 3 for the Shuffle-Exchange network), while for the shared memory type of machine this number is large. Therefore, although the algorithm can be applied for every machine, the interconnection network-like machine will be considered primarily hereafter.

The main goal of this work is to formulate a parallel numerical scheme for Boundary Value Problems and to evaluate its performance for the above type of parallel engines. Basically the present method uses the parallel technique idea developed in Lin (1986c). Let us consider the following general BVP:

$$\Phi_{xx} = g(x, \Phi, \Phi_x) \tag{1}$$

where we define

$$\Phi_x = \frac{d\Phi}{dx} \quad , \quad \Phi_{xx} = \frac{d^2\Phi}{dx^2} \quad , \quad \Phi_{x^n} = \frac{d^n\Phi}{dx^n} \tag{2}$$

g is a (nonlinear) functional of Φ , Φ_x and the coordinate x. Equation (1) has to be solved over the domain $\Omega \equiv (L, R)$, $L < R$ where L is its left

boundary and R is its right boundary. For simplicity we will consider the following Dirichlet boundary conditions for the BVP:

$$\Phi(L) = \Phi_L \quad ; \quad \Phi(R) = \Phi_R \tag{3}$$

where Φ_L and Φ_R are know quantities. We will deal here only with cases where equation (1) has a unique solution. This means that the quantities:

$$b = -\frac{\partial g}{\partial \Phi_x} \quad , \quad e = -\frac{\partial g}{\partial \Phi} \quad , \tag{4}$$

and g are continuous over Ω and $e > 0$ [Lin (1986a)].

When trying to solve numerically equation (1) using traditional approaches, we face two basic questions. The first is how to handle computationally the nonlinearity of g, and the second is how to represent numerically the spatial derivatives of Φ. When parallel computing is considered, there are some more questions that arises, for example: what do we mean by "solving equation (1)"? does it mean that we should be able to evaluate Φ (within certain accuracy) at each point $x \in \Omega$ or does it mean to suggest values for Φ at a finite number of points in Ω. It turns out that the design and the efficiency of parallel algorithms for BVP depend very much on this and other similar questions.

In the first step we would like to factor out the effects of the nonlinearity from the computational scheme since it is not a major issue of the present parallel algorithm. It was shown that the non-linearities of eq.(1) can be treated reasonably simply by using some kind of an iterative procedure, where at every iteration stage, a linear BVP is considered. Usually, a Newton-like quasi-linearization is used for nonlinear BVPs [Lin (1986b)], which results in the following linearized BVP of eq.(1):

$$\Phi_{xx} = g = d(x) - b(x)\Phi_x - e(x)\Phi \tag{5}$$

which has to be solved every iteration. From now on we'll concentrate on solving in parallel the boundary value problem associated with the above equation.

3 The problem P1$(g, \mathbf{X}; \mathbf{C})$.

Before discussing the parallel algorithms, we introduce the following $P1$ problem, which turns out to be an important sub-problem that the final

algorithm solves. The $P1$ problem is related to eq.(5) as is discussed in section 4. Let us define first the following variables:

- the real vector $\mathbf{X} : \mathbf{X}^T \equiv (x_1, x_2, x_3)$. If $L \leq x_1 < x_2 < x_3 \leq R$ then $\mathbf{X}$ is an acceptable vector. Thus an acceptable $\mathbf{X}$ is a vector of three increasing elements.
- the real vector $\phi : \phi^T \equiv (\Phi_1, \Phi_2, \Phi_3, 1)$, where $\Phi_i \equiv \Phi(x_i), \quad i = 1, 2, 3$. Usually we'll denote it by $\phi(\mathbf{X})$. Thus ϕ is a vector of four elements, where the first three are the values of $\mathbf{\Phi}$ at the three points given by the three elements of $\mathbf{X}$.
- the real vector $\mathbf{C} : \mathbf{C}^T \equiv (c_1, 1, c_3, c_4)$ is a vector of four scalars, three of which are unknown, and are part of the solution of **P1**.
- the forward and the backward spacings around the point x_2 are: $h \equiv x_3 - x_2$, $k \equiv x_2 - x_1$ where x_i are elements of an acceptable vector $\mathbf{X}$.

With these definitions, the following problem is defined:

The Problem P1: Given some acceptable vector $\mathbf{X}$, find a coefficient vector $\mathbf{C}$ such that

$$\mathbf{C^T}\phi = c_1\Phi(x_1) + \Phi(x_2) + c_3\Phi(x_3) + c_4 = 0 \qquad (6)$$

where the first three entries of $\phi(\mathbf{X})$ are the discrete values of the function Φ that fulfill eq.(5) [or eq.(1)].

As we shall see later this problem is well defined and its solutions exist. The relevance of this problem to our parallel algorithm is that in all of its versions each of the processors solves a problem of a similar type to that of **P1**. The elements of the vector $\mathbf{X}$ in these implementations are three successive elements of the key points set that will be discussed later in the parallel algorithm. One of the main features of **P1** that is needed later in the paper is the following theorem:

THEOREM: **1** *The solution* $\mathbf{C}$ *to the problem P1(g,*$\mathbf{X}$*;*$\mathbf{C}$*) exists and is unique.*

The following lemmas are needed for the proof of this theorem:

LEMMA: **1** *The n^{th} derivative of Φ that fulfills eq.(5) can be expressed generally by:*

$$\Phi^{(n)} = \alpha_n(x)\Phi' + \beta_n(x)\Phi + \gamma_n(x) \quad , \quad n = 2, 3, \cdots \tag{7}$$

where the coefficients of this equation are governed by the following recurrence system:

$$\begin{cases} \alpha_{n+1} = \alpha'_n - b\alpha_n + \beta_n & ; \quad \alpha_0 = -b(x) \\ \beta_{n+1} = \beta'_n - e\alpha_n & ; \quad \beta_0 = -e(x) \\ \gamma_{n+1} = \gamma'_n + d\alpha_n & ; \quad \gamma_0 = d(x) \end{cases} \tag{8}$$

where the primes are the derivatives with respect to x.

LEMMA: **2** *For a given P1$(g, \mathbf{X}; \mathbf{C})$ problem and a given integer t, the following are approximations for Φ_1 and Φ_3 of the order of $(t+1)$:*

$$\begin{cases} |[1 + \mathbf{Q}_t(\beta; h)]\Phi_2 + [\mathbf{Q}_t(\alpha; h) + h]\Phi'_2 + \mathbf{Q}_t(\gamma; h) - \Phi_3| \approx O(h^{t+1}) \\ |[1 + \mathbf{Q}_t(\beta; -k)]\Phi_2 + [\mathbf{Q}_t(\alpha; -k) - k]\Phi'_2 + \mathbf{Q}_t(\gamma; -k) - \Phi_1| \approx O(k^{t+1}) \end{cases} \tag{9}$$

where α , β and γ are the sets of functions $\alpha = \{\alpha_i\}$, $\beta = \{\beta_i\}$, $\gamma = \{\gamma_i\}$, and for any set of functions $f \equiv \{f_i(x)\}_{i=2}$ the functional $\mathbf{Q}$ is defined as:

$$\mathbf{Q}_t(f; z) = \sum_{i=2}^{t} \frac{z^i}{i!} f_i(x) \tag{10}$$

The proof of the first lemma is by induction, using the definitions of the coefficients in eq.(2). Algebraicially, the proof is simple and it assumes that these coefficients are smooth enough. The second lemma can be proven in a similar way to the proof of the correctness of the high order accurate numerical schemes for BVPs [Lin (1986a)]: First we expand Φ_1 and Φ_3 into a Taylor series around Φ_2. Then the high order derivatives of Φ appear in these expansions can be replaced by a linear combination of Φ_x and Φ

using equation (7) in Lemma 1 and Lemma 2 follows.

Now, by eliminating the derivative of Φ_2 in equations (9), it can be shown that Φ_1, Φ_2 and Φ_3 are linearly related where the coefficient of Φ_2 is always different from zero. This shows that the relation between all the components of ϕ is unique and thus theorem 1 is proved.

There are several ways for solving numerically the $P1$ problem for eq.(5), which, for a given BVP, is to find the dependency of the values of $\Phi(x)$, for some specific $x \in \Omega$, on the boundary conditions. We shall elaborate on one possibility, although other possibilities can be also considered as well, as long as they do not contradict the requirements that appear in the parallel algorithm scheme (section 4). Say that we spread n grid points over Ω, where the number of the point x_1 is 1, the number of the point x_3 is n and k is the number of the point x_2, $1 \leq k \leq n$. Given a desired accuracy order t, a FD approximation for eq.(5) that is spread over 3 grid points can be generated [Lin (1986a)], by applying equations (9) at the i^{th} grid point:

$$[\mathbf{Q}_t(\alpha; h_i) + h_i]\Phi_{i-1}$$

$$+\{[1 + \mathbf{Q}_t(\beta; h_i)][\mathbf{Q}_t(\alpha; -h_{i-1}] - [1 + \mathbf{Q}_t(\beta; -h_{i-1})][\mathbf{Q}_t(\alpha; h_i + h_i]\}\Phi_i$$

$$-[\mathbf{Q}_t(\alpha; -h_{i-1}) - h_{i-1}]\Phi_{i+1} =$$

$$= \mathbf{Q}_t(\gamma; -h_{i-1})[\mathbf{Q}_t(\alpha; h_i) + h_i] - \mathbf{Q}_t(\gamma; -h_{i-1})[\mathbf{Q}_t(\alpha; h_i) + h_i]$$

where

$$h_i = x_{i+1} - x_i$$

Doing this for all the internal grid points in Ω, the following tridiagonal like system is obtained:

$$l_j\phi_{j-1} + b_j\phi_j + r_j\phi_{j+1} = d_j + e_j\phi_1 + f_j\phi_n \quad , \quad j = 2, ..., n-1 \qquad (11)$$

with $l_2 = r_{n-1} = 0$, and the nature of the FD equations is that $e_j = 0$ for $j \leq 3$ and $f_j = 0$ for $j \leq n-2$. With this approach the solution vector C to the problem $P1$ is obtained in two steps:

step 1: the contribution of the lower diagonal entries, l_j, are eliminated from the 2^{nd} equation ($j = 3$) till the equation for $j = k$.

step 2: the contributions of the upper diagonal entries, r_j, are eliminated from the $n-1$ equation ($j = n-3$) backwards till the equation for $j = k$.

Following is a PASCAL program that realizes this algorithm:

```
PROCEDURE  Solve_P1 ( VAR a, b, c, d, e, f : VECTOR   ;
                      VAR C                : SOLUTION ;
                          n,k              : INDEX    );
VAR   j : INDEX ;
      m : REAL  ;
BEGIN
      FOR  j:=3  TO  k  DO
      BEGIN
            m    := a[j]/b[j-1]     ;
            b[j] := b[j] - m*c[j-1] ;
            d[j] := d[j] - m*d[j-1] ;
            e[j] :=      - m*e[j-1] ;
      END ;

      FOR  j:=n-2  DOWNTO  k  DO
      BEGIN
            m    := c[j+1]/b[j]     ;
            b[j] := b[j] - m*a[j+1] ;
            d[j] := d[j] - m*d[j+1] ;
            f[j] :=      - m*f[j+1] ;
      END ;

      c[1] := -e[k]/b[k] ;
      c[2] := 1          ;
      c[3] := -f[k]/b[k] ;
      c[4] := -d[k]/b[k] ;

END ;
```

For the simple case of $k = n-1$ we get the known folding algorithm [Wang(1981)]. In order to evaluate the performance of this program the following definitions and notations are needed:

"**AS**": The time needed to execute ADD or SUBTRACT on one processor.

"**M**": The time needed to execute MULTIPLICATION on one processor.

"**D**": The time needed to execute DIVISION on one processor.

"**CS**": The time needed to execute CHANGE SIGN on one processor.

Now, the complexity of the above program is given by the following lemma:

LEMMA: **3** *The time T needed for one processor to finish the execution of the program* **Solve_P1** *is independent of the location k of of the point x_2 inside the the set of n points -*

$$T = En \tag{12}$$

where

$$E \equiv 3M + 2AS + D + CS \tag{13}$$

The way the grid points are spread over Ω depends on the error requirements and on some pre-knowledge of the solution's behavior. However, as the order of accuracy is raised, this sensitivity is reduced [Lin (1986b)] for reasonable (polynomial) solutions. For very steep (exponential) solutions, an adaptive version [Lin (1990)] of this algorithm has to be considered. It should be noted that in order to solve $P1$ one does not need to solve for $\Phi(x)$ over Ω. Thus, for example, the use of a shooting method to solve this problem may have some disadvantage in terms of the computational complexity of the solution as in using this technique for solving $P1$ it is necessary to solve also the function itself over Ω. In order to solve for C, we need at least three independent shootings and thus its complexity is $T = \bar{E}n$ where n is the number of steps in Ω, and usually $\bar{E} > 2E$ for most of the second order accurate numerical schemes for initial value problems. Methods that use superposition and orthonormalization techniques [Scott et al (1977)] are favorable in this case since they may end up solving for C without solving for $\Phi(x)$; however, they will do it by solving a full linear system (and not form a simple tridiagonal system like here). Hereafter we'll present and discuss two basic parallel algorithms to solve the BVP in (5).

4 The parallel Algorithm PARA1.

The first parallel algorithm to solve numerically equation (5) uses a similar strategy developed in [Lin (1986c)]; this strategy consists of the following three major steps:

Step 1: Choose a set W of $P+2$ internal discrete points in Ω : $W = \{x_0, x_1, ..., x_P, x_{P+1}\}$ with the understanding that $x_0 = L$ and $x_{P+1} = R$ and $x_i < x_{i+1}$ for $0 \leq i \leq P$. These points are the *key points* which split the domain into P subdomains. The choice of W is usually based on some estimation of the subregions in Ω over which it is expected to have relatively large error in the solution, as well as on the upper bounds requirements on the errors and on the order of accuracy as will be discussed later in the paper. We define a set of P acceptable vectors $Y = \{\mathbf{Y}_i\}_{i=1}^{P}$ as follows:

$$\mathbf{Y}_i \equiv (x_{i-1}, x_i, x_{i+1})^T \quad , \quad i = 1, 2, .., P \tag{14}$$

Step 2: Solve in a parallel manner P problems of the type: $\mathrm{P1}(g, \mathbf{Y}_i; \mathbf{C}_i)$, $i = 1, 2, ..., P$, where the i^{th} processor solves the i^{th} problem independently of the other processors. Thus, each processor i suggests a relationship of the type:

$$\mathbf{C}_\mathbf{i}^\mathbf{T} \phi_\mathbf{i} = c_1^i \Phi(x_{i-1}) + \Phi(x_i) + c_3^i \Phi(x_{i+1}) + c_4^i = 0 \tag{15}$$

The important issue here is to find the set of vectors $C \equiv \{\mathbf{C}_i\}_{i=1}^{i=P}$ such that the accuracy requirements on C are fulfilled and that all the processors will finish this task in the same time. Later in the paper we'll discuss the sensitivity of this demand in cases where not all the PEs finish their task in the same time, and the tradeoffs between this demand and the accuracy requirements. This step is sometimes called the local step as it is local to the subdomain defined by $\mathbf{Y}_\mathbf{i}$ and local to the processor i.

Step 3: Solve the following tri-diagonal linear system for the set of vectors $\phi = \{\phi_i\}_{i=1}^{i=P}$:

$$\mathbf{C}_i^T \phi_i = 0 \quad , \quad i = 1, 2, ..., P \tag{16}$$

using one of the processors. This step is called the global step since the results of all the processors interact here to produce the final solution.

A possible way to execute step 2 is by letting each processor i spread a FD grid over $\mathbf{Y}_i$ (e.i. between the grid point x_{i-1} and x_{i+1}) and execute the program Solve_P1. The number of grid points and the way they are chosen in $\mathbf{Y}_i$ is determined by the requirements on the accuracy of the coefficient vector $\mathbf{C}_i$. Its accuracy is related to the accuracy of the solution of Φ over $\mathbf{Y}_i$. In any event, the resultant algebraic system is usually a banded system. Moreover, in Lin (1986a) it was shown that for every BVP it is possible to find a numerical scheme that will be accurate to any order of accuracy, and still keep the FD approximation spread over only a three-grid-point stencil. The idea behind this was explained before, and it relies on the recursive usage of equations (7) and (8) as is given by equation (9). Using this approach, let us spread m_i grid points over $\mathbf{Y}_i$ and construct the tri-diagonal linear system for the solution Φ that is governed by eq.(5) [or eq.(1)]. Now we can easily find, at any given internal grid the vector $\mathbf{C}_i$. According to Lemma 3, the time T_i needed for the i^{th} processor to finish the execution of Step 2, is independent of the location of the point x_2 inside the m_i points: $T_i = Em_i$. For the last step of this algorithm we have:

LEMMA: **4** *The time R needed to execute Step 3 is*

$$R = FP \tag{17}$$

where

$$F \equiv 4M + 3AS + 2D \tag{18}$$

The total time needed by the algorithm is defined by $T_{total} \equiv \max_i(T_i) + R$. This approach is somewhat similar to that of Kowalik et. al. (1985), where a parallel algorithm for solving a tridiagonal system was presented. In general we should mention the SIMD partition algorithm for tridiagonal systems of Wang (1981) and the partioning algorithm for banded systems of Dongarra et al (1984). In the present case, when all the PEs are identical to each other, it can be easily shown, that for all the processors to finish

Step 2 in the same time, all the m_i should be equal to each another, while usually the error is determined by the total number of points m in Ω:

$$m \equiv \sum_{i=1}^{i=P} m_i \tag{19}$$

The speedup is defined as the ratio between the time needed to solve the problem on the serial machine and the total time that is needed to solve this problem on a parallel machine using the suggested algorithm. Sometimes this definition is confusing not only because the algorithms for the serial machine and for the parallel one are not the same, but also because of the definition of the term "the same problem". For example in our case, finding the approximated values of ϕ at the set W of the key points (see Step 1 in the algorithm) is a different problem for the parallel machine than the problem of finding all the m approximated values of ϕ, while for the serial machine it will be the same problem. The efficiency of the algorithm, η, is defined as the ratio between the speedup and the number of processors P. The following property of η is applied in our case:

LEMMA: **5** *If only the P values of ϕ are required for the final solution, then $\eta = \frac{Fm_i}{Em_i+FP}$, otherwise $\eta = \frac{m_i}{m_i+P}$.*

This lemma is proven simply by substituting the expressions for T and R. An interesting case, yet not so important, is when the total number of grid points in Ω is fixed and we have the flexibility to choose the number of processors P. In this case the following lemma is applied:

LEMMA: **6** *The optimum number of processors for a given m is $P = K\sqrt{m}$ where $K = \sqrt{\frac{E}{F}}$ is a constant which depends on the machine features.*

This result can be verified by minimizing the total time with respect to P. The efficiency in this case is $\eta = \frac{1}{1+E/F}$ which is $\approx \frac{1}{2}$ for most of the machines [see also Kowalik et al (1985)]. The sensitivity of the total time to the number of processors in this case is $\Delta T_{total} = \frac{2F}{P_{optimal}}(\Delta P)^2$. It can

be seen that as the optimal number of processors increases the total time will not increase as much.

5 The parallel Algorithm PARA2.

The second parallel algorithm for solving numerically eq. (5) that will be considered here is similar to the first one, **PARA1**, in the general sense, but its local and global steps are much more closely tied to each another than in the first one. It consists of the following four steps:

Step 1: Similar to the first step of **PARA1**, choose a set W of P internal discrete points in Ω which are the key points: $W = \{x_0, x_1, ..., x_P, x_{P+1}\}$ with the understanding that $x_0 = L$ and $x_{P+1} = R$. Given a positive real number h, add additional $P+1$ points $Z = \{z_i\}$ so that $x_i - z_i = h_i$. For $i = 1, 2, ..., P+1$ define the subdomain Ω_i as $\Omega_i \equiv [z_i, x_{i+1}]$. $\mathbf{Y_i}$ is now a vector that has two vector components: $\mathbf{Y}_i^{(1)} = (z_i, x_i, z_{i+1})^T$ and $\mathbf{Y}_i^{(2)} = (x_i, z_{i+1}, x_{i+1})^T$.

Step 2: Solve in a parallel manner P systems of the type: $\mathrm{P1}(g, \mathbf{Y}_i^{(k)}\ ;\ \mathbf{C}_i^{(k)})\ ;\ k = 1, 2\ ;\ i = 1, 2, ..., P$, where the i^{th} processor solves the i^{th} system of the two problems independently of the others.

Step 3: Each processor i , $i = 1, ..., P-1$ sends its solution vector $\mathbf{C}_i^{(2)}$ to the processor $i+1$. Then each processor i substitutes the $\mathbf{C}_{i-1}^{(2)}$ and $\mathbf{C}_i^{(2)}$ results into the $\mathbf{C}_i^{(1)}$ vector result for z_i, eliminating the contributions of z_i and z_{i+1}. The new vectors $\mathbf{C}_i^{(1)}$ will be denoted by $\overline{\mathbf{C}_i}$.

Step 4: Solve the following tri-diagonal linear system for the set $\phi = \{\phi_i\}_{i=1}^{i=P}$:

$$\overline{\mathbf{C}_i^T}\phi_i = 0\ \ ,\ \ i = 1, 2, ..., P \tag{20}$$

using one of the processors.

The way step 2 is executed is very similar to that of step 2 of the previous algorithm; here each processor i spreads a FD grid over $[x_i, z_{i+1}]$ domain, with, say, n_i grid points and executes the program Solve_P1 once with

$k = n_i - 1$ to solve for $C^{(1)}$ and once with $k = 2$ to solve for $C^{(2)}$. The main difference between the two algorithms is that in the **PARA1** algorithm the domain $[x_i, x_{i+1}]$ is solved twice during Step 2 by two different processors, while in the **PARA2** algorithm this domain is solved twice by the same processor. It can be shown that the two algorithms have similar computational complexities and share the lemmas that have been discussed in the previous section. In some sense **PARA1** algorithm is similar to one that appears in Kowalik et al (1985) and **PARA2** algorithm is similar to that of Dongarra et. al. (1984). However, the effectiveness of **PARA2** can be observed when only $P/2$ subdomains are considered, and two processors are attached to each subdomain. Each of the two processors solves one of the two problems appear in step 2. Although step 3 is executed with $(P+1)/2$ processors, the efficiency is a little better than that of **PARA1**. In this result we did not take into consideration that the rest (idling) processors in step 3, can still help on a different (finer) parallel grain level.

6 Computational Tests and Analysis.

We have tested intensively the parallel numerical algorithms for different BVPs. To illustrate such a test and to demonstrate the potential of these algorithms to solve numerically BVPs in a parallel manner much faster and more exact than in the serial mode, we considered the following two point non-linear boundary value problem:

$$L(\Phi) \equiv \Phi'' - \frac{1}{2}[e^{2\phi} + (\phi')^2] = 0 \tag{21}$$

with $\phi(0) = 0$ and $\phi(1) = -\ln 2$. It has the exact solution $\phi(x) = -\ln(1 + x)$. Using the quasi-linear approach mentioned before, this equation is solved iteratively. Denoting by superscript the iteration's number, and the difference of two successive solutions by z:

$$z \equiv \Phi^{(j+1)} - \Phi^{(j)} \tag{22}$$

then the linear equation that is solved in the $j+1$ iteration is:

$$z'' - e^{2\Phi^{(j)}} z - \Phi'^{(j)} z' + L(\Phi^{(j)}) = 0 \tag{23}$$

iterations	P	number of grid points in method (1)	number of integration steps in method (2)	number of grid points in method (3)	η **PARA1**	η **PARA2**
1	3	150	100	150	0.914	0.909
	6	75	50	75	0.895	0.892
	9	50	35	50	0.832	0.829
	15	30	21	30	0.782	0.778
2	3	300	200	300	0.901	0.998
	6	150	100	150	0.881	0.878
	9	100	70	100	0.818	0.815
	15	60	42	60	0.765	0.763
3	3	600	400	600	0.888	0.887
	6	300	200	300	0.865	0.863
	9	200	140	200	0.807	0.805
	15	120	85	120	0.744	0.743
4	3	1200	800	1200	0.872	0.873
	6	600	400	600	0.850	0.850
	9	400	280	200	0.790	0.791
	15	240	160	240	0.722	0.723
5	3	1200	800	1200	0.856	0.859
	6	600	400	600	0.834	0.836
	9	400	280	200	0.771	0.773
	15	240	160	240	0.701	0.704
6	3	1200	800	1200	0.841	0.845
	6	600	400	600	0.818	0.822
	9	400	280	200	0.749	0.754
	15	240	160	240	0.679	0.685

Table 1: The efficiency of the parallel algorithms as function of the iterations.

We have used two parallel environments: the INTEL-Hypercube and the Alliant-FX/8. Although the results for the interconnection time are not as accurate, the measures for the CPU time in these models is quite good. In the present tests we have considered methods that use fourth order schemes. Three types of schemes to solve the P1 problem were considered: (1) the adaptive high order scheme of Pereyra et al (1979), (2)the shooting scheme of De Boor et. al. (1983) using the fourth order Runga-Kutta method and (3) the high order three point scheme of Lin (1986b). Out of the P available processors, processors number 1,4,7, ...use scheme 1, processors 2,5,8, ... use scheme 2 and the rest use scheme 3 (on the Alliant we ran only two of the schemes in the same time). Table 1 summarizes the main results. The first guess for Φ, $\Phi^{(0)}$, was a linear function and the iterations were stopped when $\| z \|_\infty \leq 10^{-16}$. It took about 6 iterations for both algorithms to converge. The variation in the efficiency with P is due to the different accuracy demands when computing the non-linear functions. Unlike other parallel algorithms [Dongarra et al (1984)], the algorithms used here do not require many more arithmetic operations than the appropriate sequential algorithm as is stated in the following lemma:

LEMMA: **7** *The redundancy of the* **PARA** *algorithms is* $O(P)$.

This lemma, which is simple to prove, means that the present parallel algorithms requires a total number of operations that is greater by a constant times P than this number for the serial machine. This result is not bad when compared to other parallel algorithms. The communication complexity is of the order of $O(MAXI \times P)$, where $MAXI$ is the maximum number of iterations (not counting the scattering of data to the processors, if needed, in the beginning of the algorithm). Another measure involves the computational cost and the communication cost is the ratio μ of the computational time needed for the local step, and that that is needed for the global step. Usually, as μ increases the algorithm is better in the above sense. For the present algorithms it can be proven the $\mu = \frac{m_i}{P}$, and as the accuracy demands increase (and thus also m_i), μ increases for a given machine.

References

[*De Boor et al (1983)*] De Boor, C., Hoog, F. and Keller, H.B., (1983), "The Stability of One-Step Schemes for First Order Two Point Boundary Value Problems", *SIAM J. Numer. Analys.*, 20, 6, pp. 1139-1153.

[*Dekel et al (1981)*] Dekel, E., Nassimi, D. and Sahni, S., (1981) , "Parallel Matrix and Graph Algorithms", *SIAM J. of Computing*, 10, 4, pp. 657-675.

[*Dongarra et al (1984)*] Dongarra, J. J. and Sameh, A., (1984), "On Some Parallel Banded System Solvers", *Argonne National Labs, Mathematics and Computer Science Division*, Technical Memorandum No. 27.

[*Keller (1968)*] Keller, H. B. , (1968) , **Numerical Methods for Two - Point Boundary Value Problems** , Waltham. Mass. : Blaisdell .

[*Feng (1981)*] Feng, Tse-yun, (1981) , "A Survey of Interconnection Networks", *Computer*, 14, 12, pp. 12-27.

[*Flynn (1966)*] Flynn, M. J., (1966) , "Very High Speed Computing Systems", *Proc. IEEE*, 54, pp. 1901-1909.

[*Gannon et al (1985)*] Gannon, D. and Van Rosendale, J., (1986), "On the Structure of Parallelism in a Highly Concurrent PDE Solver", *J. of Parallel and Distr. Comp.*, 3, pp. 106-135.

[*Kowalik et al (1985)*] Kowalik, J. S., and Kumar, S. P., (1985), **Parallel MIMD Computation: HEP Supercomputer and Its Applications**, The MIT Press, pp. 295-307.

[*Lin (1986a)*] Lin, A, (1986), "High Order Three Points Schemes for Boundary Value Problems : I) Linear Problems " , *SIAM J. Scient. and Stat. Computing* , 7, No. 3, 1986.

[*Lin (1986b)*] Lin, A., (1986), "High Order Three Points Schemes for Boundary Value Problems : II) Non-Linear Problems ", *the J. of Computational and App. Mathematics* , 15, pp. 269 - 282.

[*Lin (1986c)*] Lin, A., (1986), "On the Parallel Algorithm for Inherent Serial Techniques", *Linear Algebra and Applications*, 79, pp. 229-236.

[*Lin et al (1987)*] Lin, A. and Gunton, D. J., "Parallel and Super Computing of Elliptic Operators", in **Supercomputing**, Kartasheve, L. and Kartashev, S. editors, The International Supercomputing Institute, pp. 497-502.

[*Lin (1989)*] Lin, A., (1989), "Solving Numerically The Navier Stokes Equations on Parallel Systems", *The International Journal of Numerical Methods in Fluids*, to appear.

[*Lin (1990)*] Lin, A., (1989), "Parallel Numerical Algorithms for Fluid Dynamics Simulation", *AIAA paper 90-0333.*

[*Pereyra (1979)*] Pereyra, V. , (1979) , "PASVA3: An Adaptive Finite Difference FORTRAN Program for First-Order Nonlinear Ordinary Boundary Problems", in **Codes for Boundary Value Differential Equations**, B. Childs, M. Scott, J. W. Daniel, F. Denman and P. Nelson (Eds), Lecture Notes in Computer Science, 76, Springer-Verlage, New York, N.Y. , pp. 67-88 .

[*Pfister et al (1985)*] Pfister, G. F., Brantley, W. C., George, D. A., Harvey, S. L., Kleinfelder, W.J., McAuliffe, K.P., Melton, E.A., Norton, V. A. and Weiss, J., (1985), "The IBM RP3 – Introduction and Architecture", *IEEE Proc. of the 1985 Int. Conf. on Parallel Processing*, Session 11a, pp. 764-797.

[*Schwartz (1980)*] Schwartz, J., (1980) , "Ultracomputers", *ACM Trans. Program. Lang. Syst.*, 2, pp.484-521.

[*Scott et al (1977)*] Scott, M. R. and Watts, H. A., (1977), "Computational Solution of Linear Two-Point Boundary Value Problems via Orthonormalization", *SIAM J. Numr. Anal.*, 14, pp. 40-70.

[*Seitz (1985)*] Seitz, C.L., (1985) , "The Cosmic Cube", *Communications of the ACM*, 28, 1, pp. 22-33.

[*Wang (1981)*] Wang, H. H., (1981), "A Parallel Method for Tridiagonal Equations", *ACM Trans. on Mathematical Software*, 7, pp. 170-183.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-102498 ICOMP-90-03		

4. Title and Subtitle	5. Report Date
Parallel Algorithms for Boundary Value Problems	January 1990
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Avi Lin	E-5292
	10. Work Unit No.
	505-62-21

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	
	13. Type of Report and Period Covered
	Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546-0001	

15. Supplementary Notes

Avi Lin, Dept. of Mathematics, Temple University, Philadelphia, Pennsylvania 19122 and Institute for Computational Mechanics in Propulsion, Lewis Research Center, Cleveland, Ohio 44135. Work funded by Space Act Agreement C-99066-G. Space Act Monitor, Louis A. Povinelli.

16. Abstract

In the present paper we discuss a general approach to solve Boundary Value Problems numerically in a parallel environment. The basic algorithm consists of two steps: The local step where all the P available processors work in parallel, and the global step where one processor solves a tridiagonal linear system of the order P. The main advantages of this approach are two fold: First—this suggested approach is very flexible, especially in the local step and thus the algorithm can be used with any number of processors and with any of the SIMD or MIMD machines. Secondly—the communication complexity is very small and thus can be used as easily with shared memory machines. Several examples for using this strategy are discussed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Boundary value problems Parallel computation High order schemes	Unclassified – Unlimited Subject Category 64

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	21	A03

*For sale by the National Technical Information Service, Springfield, Virginia 22161