03 Aug 1993

Implicit Methods on Parallel Processors

Larry Reeves

Implicit Methods on Parallel Processors

Larry Reeves
August 3, 1993

CSC 93-18

Abstract

Since most partial differential equations (PDEs) do not have exact solutions, they are usually solved by some type of numerical method. Since a numerical method is commonly built from finite difference approximations derived from Taylor series expansions, such a development is derived. Stability and convergence of these methods is defined and the rate of convergence is defined and shown for a few simple methods. Of particular importance is the difference between implicit and explicit methods. Finally, the current applications and adaptations of implicit methods on parallel processors are examined and their strengths and weaknesses discussed.

Department of Computer Science
University of Missouri-Rolla
Rolla, Missouri 65401

Implicit Methods on Parallel Processors

Larry Reeves

University of Missouri-Rolla

August 3, 1993

Introduction

In computer science, a large bulk of the mathematics is discrete, for example algorithm analysis, logic and expert systems. However, there is also a large area that deals with continuous mathematics, like graphics, robotics and numerical methods. Within numerical methods exist many different specialties such as optimization, solution of systems of linear equations and solution (or approximation to solution) of non-linear equations. This paper will deal with the application of numerical methods to a subset of the non-linear equations group, namely, differential equations, specifically partial differential equations.

Partial differential equations (PDEs) are used in many different fields. The general types of equations are frequently given their own names such as the wave

equation, the heat equation, and the Navier-Stokes equations. Although it would be nice to have the family of solutions to any particular PDE, the equations are rarely that easy to solve and it is oftentimes sufficient to be able to closely approximate a particular solution, or sometimes the behavior of the solutions, in a given region over which the equations are valid. For convenience, the region is denoted as as a set of points, Φ, with a subset of points, $\delta\Phi$, denoted as the boundary. For the purposes of this paper, Φ will usually be a square grid of points, with $N+1$ points on each side, and $\delta\Phi$ will be a subset of the edges of this square. When the interior of the grid is mentioned, the set of points indicated is the set $\Phi - \delta\Phi$.

In the solution of PDEs, the type of equation is often a major consideration. To understand the equations, it is first necessary to know the types of PDEs. The basic structure of a PDE can be described by its linearity and its order. A PDE is linear if any power of the function is no higher than one and no two different partials of the function appear in the same term. The order of a PDE is the power of the highest differential in the equation. The examination of PDEs usually starts with second-order linear equations in two independent variables. There are three types of second order linear PDEs, classified as elliptic, parabolic and hyperbolic. To determine the type of equation, it is simply a matter of comparing it to the standard form of a linear second-order PDE:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu + G = 0, \qquad (1)$$

where A–G are constants or functions only of x and y. If $B^2 - 4AC$ is positive, the equation is hyperbolic, if $B^2 - 4AC$ is zero, then the equation is parabolic, otherwise the equation is elliptic.

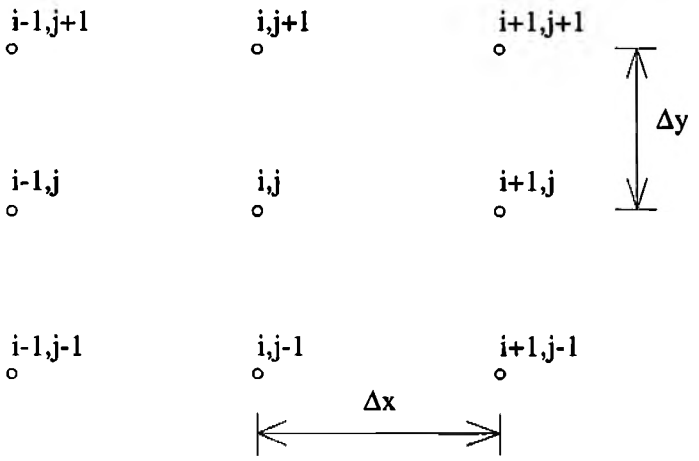
As an indicator of the methods commonly used to solve these different types of equations, they are commonly grouped into two categories: parabolic and hyperbolic equations are often called marching problems; while elliptic equations are frequently called jury problems. These descriptions point out the usual methods of solution since parabolic and hyperbolic equations are solved by marching, or wavefront, methods, while elliptic equations are solved by differencing techniques that require input from all of the boundary values on all sides. Thus, for hyperbolic and parabolic problems $\partial\Phi$ is usually just three sides of the grid, while for elliptic problems $\partial\Phi$ is all four edges. In the next sections, these methods are discussed in more depth and both explicit and implicit versions of these methods are examined.

Finite Differencing

In order to approximate the solution of a PDE, the region Φ must be defined. This usually results in a grid at whose intersection points the solution is to be obtained. If this grid is non-uniform, various methods are used to regularize the grid, or sometimes the grid is left as is and extra values are carried along to adapt for the irregularities. Even if we use the previous assumption that the grid is both regular and square, there must be some method to represent the continuous equations at the now discrete points of the region. This method is referred to as finite differencing.

The fundamental equation behind finite differences is the Taylor series. To

illustrate the use of the Taylor series, a second-order accurate central difference equation is derived. First, the points around the region of interest are defined:



Now, the value of the variable u at point $i+1, j$ ($u_{i+1,j}$) can be expressed by expanding u in a Taylor series about the point i, j:

$$u_{i+1,j} = u_{i,j} + (\Delta x)\left.\frac{\partial u}{\partial x}\right|_{i,j} + \frac{(\Delta x)^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + O((\Delta x)^3) \tag{2}$$

where $\Delta x = u_{i+1,j} - u_{i,j}$ and x increases in the direction of increasing i. Similarly, the value of $u_{i-1,j}$ can be defined:

$$u_{i-1,j} = u_{i,j} + (\Delta' x)\left.\frac{\partial u}{\partial x}\right|_{i,j} + \frac{(\Delta' x)^2}{2!}\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + O((\Delta' x)^3) \tag{3}$$

where, $(\Delta' x) = u_{i-1,j} - u_{i,j}$. Now, if the grid is uniform, $(\Delta x) = -(\Delta' x)$, so the equations can be rewritten as:

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} - \frac{\Delta x}{2}\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + O((\Delta x)^2) \tag{4}$$

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{\Delta x} + \frac{\Delta x}{2}\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + O((\Delta x)^2) \tag{5}$$

Adding these two equations and dividing by 2 results in the familiar equation:

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2(\Delta x)} + O((\Delta x)^2). \tag{6}$$

The equation is called central differenced because the point of interest is in the center of the two points used to evaluate the partial derivative, and the approximation is second-order, because the error due to truncation of the Taylor series is $O((\Delta x)^2)$.

Similarly, the first-order forward difference equation can be derived from equation 2 and the first-order backward difference equation can be derived from equation 3. Also, if equation 5 is subtracted from equation 4, then the first partial terms drop out and the remaining terms define a second-order central difference for the second partial derivative. Also, difference equations can be developed for mixed partials by using Taylor series expansions for multiple independent variables.

Explicit and Implicit Methods

Now that the fundamental differencing techniques have been described, the next step is to examine and explain the primary types and applications of these techniques. As mentioned earlier, these differencing methods can be used in both explicit and implicit schemes. Conceptually, the basic difference between explicit

and implicit methods is whether all of the values in the difference equation, except for the variable of interest, are known. If all of the values are known, the method is explicit, otherwise the method is implicit.

As an example, examine the dimensionless diffusion equation,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}. \tag{7}$$

Using a forward difference for $\frac{\partial u}{\partial t}$ at point i and a second-order centered difference for $\frac{\partial^2 u}{\partial x^2}$ gives the following difference equation:

$$\frac{w_{i,t+1} - w_{i,t}}{\Delta t} = \frac{w_{i+1,t} - 2w_{i,t} + w_{i-1,t}}{(\Delta x)^2}, \tag{8}$$

which, when rearranged becomes:

$$w_{i,t+1} = w_{i,t} + \frac{\Delta t}{(\Delta x)^2}(w_{i+1,t} - 2w_{i,t} + w_{i-1,t}), \tag{9}$$

where we use $w_{i,t}$ to represent the discretized approximation to the exact value $u_{i,t}$. This is the explicit form of equation 7. Now, ignoring x direction boundaries, if all the values of w at time $t = 0$ are known, then the values for time $t = 1$ can be directly calculated.

However, if instead of taking the central difference at time t the difference is expressed at time $t + 1$, then equation 9 becomes:

$$w_{i,t+1} = w_{i,t} + \frac{\Delta t}{(\Delta x)^2}(w_{i+1,t+1} - 2w_{i,t+1} + w_{i-1,t+1}), \tag{10}$$

which is the implicit form of equation 7. Since the only known value in this equation is $w_{i,t}$, it is not possible to directly solve equation 10 directly for all the

other variables. Instead, if the equations for all of the points in the region are written out, the result is a system of linear equations of the form:

$$AW_{t+1} = BW_t, \tag{11}$$

where A and B are matrices of constants and W is the vector of variables at either time t or $t+1$. Since the only unknowns are in W_{t+1}, the matrix A can be inverted to solve for the unknowns W_{t+1}.

The remaining question is which method to choose for a particular problem. There are three main considerations to making this decision: the amount of computer memory, the stability of the numerical method and the convergence of the method. If the local computer does not have a large amount of available memory, then explicit (or mixed explicit-implicit) methods may be the only choice due to the lessened need for variable storage. With explicit methods, the new values (e.g. $u_{i,t+1}$) are defined purely as functions of the old values, so the program includes the functional equation, and only 2-3 sets of values need to be stored. With implicit methods, the matrix A usually has to be stored, along with A^{-1} and, if the method is particularly complex, B. Thus, for explicit methods the necessary storage is $O(N)$ while implicit methods need $O(N^2)$ space. For example, for a two-dimensional problem with 100 grid points in each direction, an explicit method may only need to store approximately 30,000 values, while an implicit method may need as many as 300,000,000 values. A mixed explicit-implicit, or block implicit, method subdivides the region into blocks which are solved by an implicit method with the borders between blocks solved explicitly. For the above example, if there were four blocks, a block implicit method may need to store only 18,750,000 values and ten blocks might only use 3,000,000. The remaining

two factors in choosing the method, stability and convergence of the particular method, will be discussed in the next two sections.

Stability

A numerical method is called stable if for initial/boundary values sufficiently close together, the resulting solutions are also close together. An unstable method can produce results drastically different even when the initial guesses are arbitrarily close together. Another way to describe stability is that the solution of the approximation is continuous even in the presence of round-off errors. What this means is that if the values used in the successive iterations are not stored exactly, which in general they can't, then the results will still be close to thosc that would have been achieved by exact storage of values.

For a better understanding of stability we will again examine the diffusion equation, (equation 7) in the discretized form of equation 9. At any particular iteration, each value $w_{i,t} = w^*_{i,t} + \epsilon_{i,t}$, where $w^*_{i,t}$ is the exact solution of the finite difference equation and $\epsilon_{i,t}$ is the difference between this exact solution and the current solution of the finite difference equation. Note that at any particular time, $\epsilon_{i,t}$ includes both the round-off errors from the previous step and the remaining part of the error in the initial guess. If the numerical method is consistent, that is the approximation approaches the exact solution for smaller values of $\Delta t, \Delta x, \ldots$, then both $w_{i,t}$ and $w^*_{i,t}$ satisfy the approximation equation so $\epsilon_{i,t}$ must also satisfy

the equation. This gives us

$$\epsilon_{i,t+1} = \epsilon_{i,t} + \frac{\Delta t}{(\Delta x)^2}(\epsilon_{i+1,t} - 2\epsilon_{i,t} + \epsilon_{i-1,t}). \tag{12}$$

Now, we can build up these errors by a Fourier series, and since the method is linear, then each term of the series can be examined separately and the results combined by superposition. The individual terms have the form

$$\epsilon_m(x,t) = e^{at}e^{\iota k_m x}, \tag{13}$$

where m is the frequency of the error, $\iota = \sqrt{-1/2}$, a is some complex constant and k_m is a real constant, called the wave number. By substituting this equation into equation 12, we discover that the errors only satisfy the equation if $\frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$ (see [2, pp.71-75] or [1, pp.47-8] for more details). This means that this explicit method is only conditionally stable, i.e. the time step can only grow so fast with respect to the grid resolution or the errors will grow such that they no longer satisfy the approximation. Unconditionally stable methods are those that impose no restrictions on the relative sizes of the different space and time steps. By doing a similar analysis to the above, it can be shown that most implicit methods are generally unconditionally stable. Since time-dependent problems usually do one computational iteration for each time step of Δt, this means that in general implicit methods can arrive at a fixed time T in fewer iterations than explicit methods for a given grid. However, if the development of the solution as time progresses is of major importance, then for non-linear problems, the time step must already be small to maintain this development. Thus, for time-accurate problems, explicit methods may be a better choice, since the small time

satisfies the stability criteria, and explicit iterations involve fewer calculations than implicit iterations.

Doing the stability analysis for the general PDE case is not as simple as above. For the Fourier method to be valid, the finite difference must be linear. For most interesting problems, the PDEs are non-linear, and frequently the finite representations must also be non-linear to preserve some desired quality of the solution. If we can prove the stability, there is an extra benefit. Due to a result by Lax, (see [10]), if a particular method is both consistent and stable, then the method is also convergent. We will discuss convergence in the next section.

Convergence

In any discussion of convergence as applied to numerical methods, it is important to understand what is meant by the convergence of an iteration process. An application of any method is said to be converged if the change from one iteration of the method to the next is sufficiently small. There are two concepts of the definition that need further explanation. The first is the concept of change. In some cases, if a particular value is supposed to approach zero and can never reach zero due to some constraint, then the method might be considered converged if the difference in value between the last two iterations is less than some small percentage of the difference between the previous two iterations. This leads to the other concept: what, exactly, are the values the change is measuring?

The simplest value that we can measure the change of is that of a single variable. In this case, we are usually only interested in the maximum of all the individual changes of the variables. This is commonly referred to as the $\|L\|_\infty$ norm:

$$\|L\|_\infty = \max_{i \in \Phi} |f_i|, \tag{14}$$

where Φ is the set of points defined previously and f_i is some function of the values of interest. For the simple case above, $f_i = w_i$. Two other norms are also commonly used, $\|L\|_1$ and $\|L\|_2$:

$$\|L\|_1 = \sum_{i \in \Phi} |f_i|, \text{and} \tag{15}$$

$$\|L\|_2 = \left[\left(\sum_{i \in \Phi} (f_i)^2 \right)^{1/2} \right] \tag{16}$$

$\|L\|_1$ is used whenever the total change in all values is desired to be less than a certain amount, and $\|L\|_2$ is usually used when a weighted average of all differences is desired.

With these definitions, we can usually decide if a particular method has converged. This is the easy part. The more difficult part of convergence is finding out if a method is actually supposed to converge, or if it is just faulty programming. A particular method is said to converge if, for all possible values of independent parameters, the application of that method will converge. If the method does converge, the rate of convergence is also an important parameter to look at.

To examine these techniques, we will start with the model problem of Laplace's equation,

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0. \tag{17}$$

Using a central-difference for each of the two terms results in the finite difference form,

$$-\frac{w_{i+1,j}-2w_{i,j}+w_{i-1,j}}{(\Delta x)^2}-\frac{w_{i,j+1}-2w_{i,j}+w_{i,j-1}}{(\Delta y)^2}=0 \tag{18}$$

where the error terms have been dropped. Now, if $\Delta x = \Delta y$, then this can be written as

$$w_{i,j}=\frac{1}{4}(w_{i+1,j}+w_{i-1,j}+w_{i,j+1}+w_{i,j-1}) \tag{19}$$

or, in matrix format,

$$\mathbf{w}=\mathbf{A}\mathbf{w}. \tag{20}$$

For a purely explicit method, the $w_{i,j}$ term is set at iteration $k+1$ while all of the other values are taken from the kth iteration. This results in the equation

$$w_{i,j}^{(k+1)}=\frac{1}{4}(w_{i+1,j}^{(k)}+w_{i-1,j}^{(k)}+w_{i,j+1}^{(k)}+w_{i,j-1}^{(k)}) \tag{21}$$

or, similar to above,

$$\mathbf{w}^{(k+1)}=\mathbf{A}\mathbf{w}^{(k)}. \tag{22}$$

Define $\boldsymbol{\varepsilon}^{(k)}=\mathbf{w}^{(k)}-\mathbf{w}^{*}$ as the error between the values at the current iteration and the exact solution of the approximation equation. Then, subtracting 20 from 22 results in

$$\boldsymbol{\varepsilon}^{(k+1)}=A\boldsymbol{\varepsilon}^{(k)}. \tag{23}$$

Similarly, we can define $\boldsymbol{\varepsilon}^{(k)}, \boldsymbol{\varepsilon}^{(k-1)}, \ldots$, so this can be rewritten as

$$\boldsymbol{\varepsilon}^{(k+1)}=A^{k}\boldsymbol{\varepsilon}^{(0)}, \tag{24}$$

where $\boldsymbol{\varepsilon}^{(0)}$ is the error in the initial guess. Since the goal is to reduce the error to as close to zero as possible, we want

$$\lim_{k\to\infty}\boldsymbol{\varepsilon}^{(k)}=0, \tag{25}$$

which, if $\epsilon^{(0)} \neq 0$, reduces to

$$\lim_{k\to\infty} \mathbf{A}^k = 0, \tag{26}$$

which is a sufficient condition for convergence. From the Lax theorem of the previous section, however, this condition may not be necessary. From matrix theory, for $\mathbf{A}^k \to 0$, the spectral radius (eigenvalue of largest absolute value) $\rho(\mathbf{A})$ must be less than one. It can be shown [13, pp.202-3] that for this iteration scheme, that $\rho(\mathbf{A}) = \cos(\pi/N)$, where $N+1$ is the number of grid points as previously defined.

Going back to equation 18, and again assuming $\Delta x = \Delta y$, we can rewrite this as

$$(\mathbf{H} + \mathbf{V})\mathbf{w} = 0. \tag{27}$$

where

$$\mathbf{H}\mathbf{w} = w_{i+1,j} - 2w_{i,j} + w_{i-1,j}$$

and

$$\mathbf{V}\mathbf{w} = w_{i,j+1} - 2w_{i,j} + w_{i,j-1}.$$

This equation can be rewritten as the pair of equations

$$\begin{aligned}(\mathbf{H} + r\mathbf{I})\mathbf{w} &= (r\mathbf{I} - \mathbf{V})\mathbf{w} \\ (\mathbf{V} + r\mathbf{I})\mathbf{w} &= (r\mathbf{I} - \mathbf{H})\mathbf{w}\end{aligned} \tag{28}$$

These two equations can be solved iteratively, resulting in the *Peaceman- Rachford* alternating-direction implicit method,

$$\begin{aligned}(\mathbf{H} + r_{k+1}\mathbf{I})\mathbf{w}^{k+1/2} &= (r_{k+1}\mathbf{I} - \mathbf{V})\mathbf{w}^k \\ (\mathbf{V} + r_{k+1}\mathbf{I})\mathbf{w}^{k+1} &= (r_{k+1}\mathbf{I} - \mathbf{H})\mathbf{w}^{k+1/2}\end{aligned} \tag{29}$$

where the r_{k+1}'s are iteration parameters that can be chosen to improve speedup. To examine the convergence of 29, assume for simplicity that the r_{k+1}'s are each 1. Recombining the two equations, the resulting matrix form is

$$\mathbf{w}^{k+1} = \mathbf{T}\mathbf{w}^{k}, \tag{30}$$

where $\mathbf{T} = (\mathbf{V}+\mathbf{I})^{-1}(\mathbf{I}-\mathbf{H})(\mathbf{H}+\mathbf{I})^{-1}(\mathbf{I}-\mathbf{V})$. To derive $\rho(\mathbf{T})$ notice first that equation 30 can be modified to

$$\tilde{\mathbf{T}} = (\mathbf{V}+\mathbf{I})\mathbf{T}(\mathbf{V}+\mathbf{I})^{-1} \tag{31}$$

or

$$\tilde{\mathbf{T}} = [(\mathbf{I}-\mathbf{H})(\mathbf{H}+\mathbf{I})^{-1}][(\mathbf{I}-\mathbf{V})(\mathbf{V}+\mathbf{I})^{-1}]. \tag{32}$$

Since $\mathbf{T}$ and $\tilde{\mathbf{T}}$ are similar matrices, they share the same eigenvalues. Again by a method similar to [13, pp.213-6] it can be shown that

$$\rho(\mathbf{T}) = \left[\max\left\{\left|\frac{1-4\sin^2(\frac{\pi}{2N})}{1+4\sin^2(\frac{\pi}{2N})}\right|, \left|\frac{1-4\sin^2(\frac{\pi(N-1)}{2N})}{1+4\sin^2(\frac{\pi(N-1)}{2N})}\right|\right\}\right]^2, \tag{33}$$

which, for $N \geq 10$ is just

$$\rho(\mathbf{T}) = \left(\frac{1-4\sin^2(\frac{\pi}{2N})}{1+4\sin^2(\frac{\pi}{2N})}\right)^2. \tag{34}$$

As can be seen in figure 1 for values of $N \geq 5$, this simple implicit method has a smaller spectral radius than the explicit method. This indicates that the implicit method will converge faster than the explicit method. However, as N increases, the two values approach the same limit, so the benefits decrease for

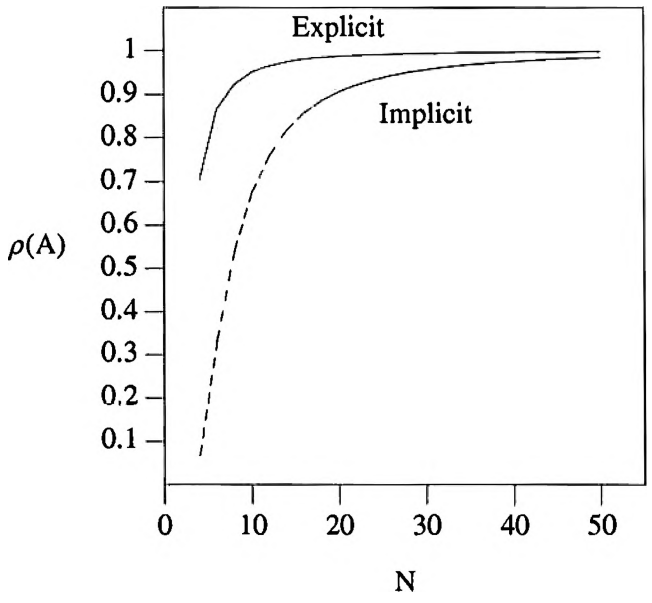


Figure 1: Comparison of Spectral Radii for Simple Methods

increasing N. Remember, however, that this is only a very simple approach to implementations of the two types of methods.

Another way to compare convergence is by using the spectral radius to get the average rate of convergence for m iterations of $\mathbf{A}$, $R(\mathbf{A}^m)$, which is defined as

$$R(\mathbf{A}^m) = \frac{-\ln \|\mathbf{A}^m\|}{m}, \tag{35}$$

where

$$\|\mathbf{A}^m\| = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}^m\mathbf{x}\|_2}{\|\mathbf{x}\|_2}. \tag{36}$$

However, since $R(\mathbf{A}^m)$ is usually so difficult to calculate, the value frequently used

to compare to methods is the asymptotic rate of convergence, (see [13, p.67])

$$R_\infty(\mathbf{A}^m) = \lim_{m\to\infty} R(\mathbf{A}^m) = -\ln \rho(\mathbf{A}). \tag{37}$$

Looking back at equation 29, note that the r_{k+1} parameters can be chosen dynamically for each iteration of the ADI method. Again from [13, pp.219-228], we see that if bounds on the eigenvalues can be calculated, then these parameters can be chosen in such a manner as to greatly speed up the convergence of the ADI method (mADI). In figure 2 is shown the advantage of using mADI over both the simple explicit and implicit methods derived previously, and even

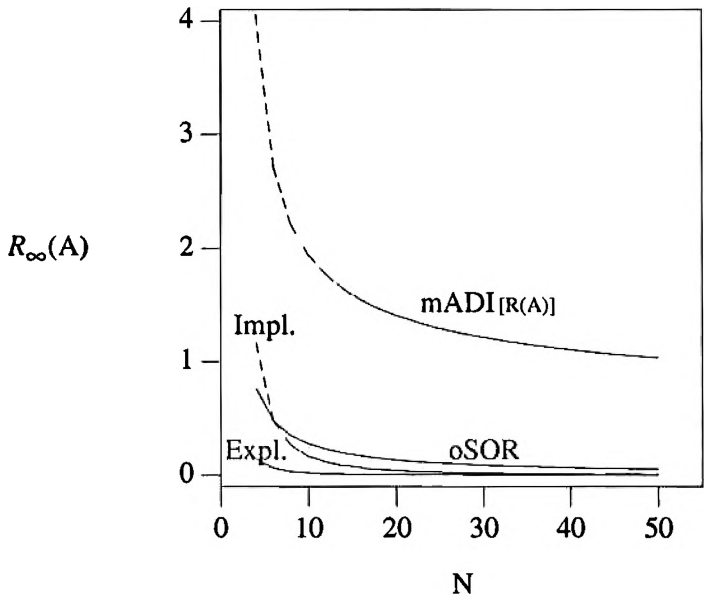


Figure 2: Rates of Convergence for Explicit and Implicit Methods

its distinct advantage over the SOR method with the over-relaxation parameter

chosen for optimal asymptotic rate of convergence (oSOR). Even though the mADI values are only for the average rate of convergence, [13, p.67] shows that $R(\mathbf{A}^m) \leq R_\infty(\mathbf{A}^m)$ for sufficiently large m, so mADI actually is better than illustrated. Also, in figure 3 we see that the advantage of the mADI method over oSOR continues even for large N, so that this implicit method maintains its advantages over the explicit methods.

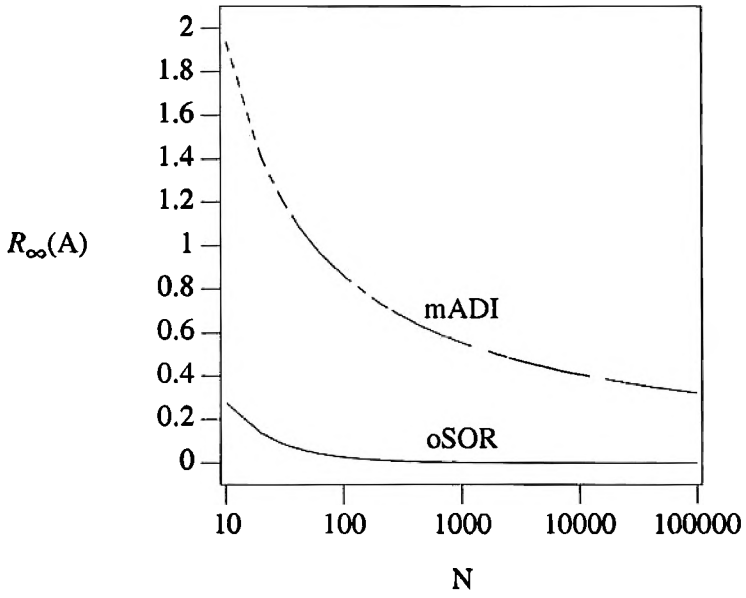


Figure 3: Rates of Convergence for large N

Implementations

The final question remaining now is: How well do these implicit methods work on parallel processors? The answer to this question is not as simple as either well or bad. Instead, how well implicit methods work depends on which methods were used and what was the configuration of the machine they were used on. One of the standard methods to compare the usefulness of a parallel code is to find its maximum speedup. The common way to do this is to determine the percentage of the code that must be performed sequentially, which is the reciprocal of the maximum speedup. A similar analysis to this was done by [8], using several factors. The problem with this analysis is that the equations are solved for a fixed problem size. In most cases, the sequential part of the problem does not scale up in proportion to the size of the problem. Thus, as the problem grows, the sequential portion decreases in percentage.

Shared Memory

When implementing different numerical methods, the architecture used is an important factor in how well the particular method will work. The classic complaint about shared memory parallel computers (multi-processors) is that as you scale up the number of processors, the communication link between the processors and the memory becomes saturated, and the speedup is thus limited by the memory-processor bandwidth. Although none of the papers in this overview

explicitly run into this limitation, [9] states that their configuration of array processors on an APTEC bus, would be limited to about twelve processors before hitting the bandwidth limit.

Another problem with multi-processors is that without careful programming, the processors have conflicts when attempting to read/write the same memory locations. This problem is illustrated in [4]. On a 8 processor VAX 6300, the ADI method has significantly worse speedup than the SOR method, due to the size of the memory cache on the processors. In the SOR method, the data can be accessed by either rows or columns, to allow for memory access methods. With ADI, the data has to be accessed by both rows and columns, so if the cache is not small enough, there can be significant conflicts. This problem is partially avoided in the work by [7] who use a four processor system that has global shared memory, but that also has some decent individual memory. By doing this, for the ADI method, the processors can be loaded with the appropriate rows or columns and then can operate independently on the local data. However, this introduces another problem. Instead of having conflicts while the computations are going on, the processors have to wait between sweeps of the ADI method to write and read the new data. Thus, they cannot do continual computations due to the memory synchronizations.

Another way to decrease the impact of memory conflict was used by [5]. They take the global domain and split it into sub-regions that only share a single point (in one-dimensional cases). By doing this, the individual processors can use the ADI method locally, and the only time they have to synchronize is when two adjacent domains finish a sweep. At this time they resolve any differences in

values for the shared point. By doing this, global synchronization is unnecessary and the processors can remain busy more of the time.

By now, it may be worth questioning whether or not implicit methods are better than explicit methods on multi-processors if there are so many problems. Part of the answer is that regardless of the method, the bus bandwidth problem will still remain. The other answer to this question is answered in much the same way by [4], [6] and [11]. Although the methods have some speedup and efficiency problems, the sequential versions are so much faster than explicit methods that even for relatively low speedups, they still outperform the explicit methods. None of these authors, however, actually managed to run enough processors to see if these results would continue beyond about twenty processors.

Distributed Memory

This brings us to distributed-memory parallel processors (multi-computers). Although this architecture avoids the problem of memory conflicts, there is still a problem with shared domain communication that must be answered, usually by message passing. The first problem encountered with multi-computers is how to decompose the computation domain. If the domain is broken up into strips, then distributed by strips over the processors, the affects on the ADI method are significant as shown by [3]. Since, in the ADI method, the data are computed by both rows and columns, either method of data decomposition will result in fully utilized processors in one direction, while in the other direction, the processors

will sit idle waiting for the results from its neighbor. This is avoided in [3] by skewing the data over the processors. The data is split into grids, and then each grid column past the first is shifted down a processor, so that both the first grid row and the first grid column are distributed over the processors. Although this mandates communication in both directions, the increased efficiency of the processors in the other direction more than makes up the difference.

A multi-computer method similar to that used in [5] was used in [6]. They split the domain into independent sub-regions, solving them implicitly, then uses a global preconditioned conjugate gradient method to solve the equations for the interfaces between sub-regions. The main problem with this method is that as the number of processors increases, the conjugate gradient part of the code will eventually overwhelm the time for the implicit solution of the sub-region.

Another approach to avoiding the global nature of the ADI method is suggested by [11]. He suggests that to decouple the resulting tridiagonal matrices, a series of W matrices are constructed to rearrange the unknowns so that they can be easily partitioned. Although this results in some duplicated computations on the processors, the time saved due to fewer communications can be significant.

The final problem with parallel computing is that development of the numerical methods is usually done for sequential computers and then adapted, within some analytic constraints, for use on a parallel computer. An example of this was found in [12], where they took a code that was developed for the CRAY and adapted it for use on the Intel iPSC/860. The net result of this is that for 128 processors, they only got their code to run at around 3.3 times the speed of the

CRAY. This represents a speedup of approximately 90. Although this is by far the most ambitious code of all the papers, the net results show that it may not be extremely useful to just adapt a sequential code without careful examination of potential difficulties.

Conclusions

With the exception of a couple of these reports, the major problem is that nobody is doing direct comparisons between explicit and implicit methods. This is important because on sequential machines, implicit methods are preferred over explicit methods, except with time-accurate problems, due to their improved stability and faster convergence rates. Although the ideas behind these comparisons seem to carry over to parallel machines, there seems to be few demonstrations to this effect. Those that do compare the two only present the elapsed time comparisons and say nothing about how well the particular methods are supposed to converge as opposed to how well the programs that implement them do converge. Another problem with the results from these papers is that most of them were done to test the abilities of a particular set of hardware, so even the implicit methods themselves cannot be compared to each other without some way to compare their respective hardware.

Thus a major focus in the area of implicit methods needs to be on the comparison of implicit and explicit methods on parallel processors. Also, the rates of convergence of the different methods needs to be addressed along with the

achievability of these rates on different parallel architectures. With these numbers in hand, researchers will have a more solid basis for choosing a particular method, and may be better able to adapt implicit methods to parallel processors to maintain the sequential benefits.

References

[1] William F. Ames. *Numerical Methods for Partial Differential Equations.* Computer Science and Applied Mathematics. Academic Press, Inc., second edition, 1977. 4th printing.

[2] Dale A. Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer.* Hemisphere Publishing Corporation, 1984.

[3] R. K. Cooper and D. A. Peshkin. Parallel alternating direction implicit method on a network of transputers. *Computer Systems Science & Engineering*, 5(1):47–52, January 1990.

[4] Zarka Cvetanovic, Edward G. Freedman, and Charles Nofsinger. Efficient decomposition and performance of parallel PDE, FFT, monte carlo simulations, simplex, and sparse solvers. *The Journal of Supercomputing*, 5:219–238, 1991.

[5] Peter G. Eltgroth and Mark K. Seager. The sub-implicit method: New multiprocessor algorithms for old implicit codes. *Parallel Computing*, 8(1–3):155–163, 1988.

[6] William D. Gropp and David E. Keyes. Domain decomposition on parallel computers. *Impact of Computing in Science and Engineering*, 1(4):421–439, December 1989.

[7] L. Mane and Ta Phuoc Loc. Simulation of unsteady flows at high reynolds numbers on an experimental multiprocessor system. *La Recherche Aérospatiale (English Edition)*, (2):49–58, 1987.

[8] Joseph F. McGrath, Darrell L. Hicks, and Lorie M. Liebrock. Parallel algorithms for computational continuum dynamics. *Applied Mathematics and Computation*, 20(1–2):145–173, September 1986.

[9] Elaine S. Oran and Jay P. Boris. Numerical methods in reacting flows, AIAA paper 87-0057. In *AIAA 25th Aerospace Sciences Meeting*, Reno, NV, January 12–15, 1987. AIAA.

[10] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* Interscience Publishers, second edition, 1967.

[11] G. Rodrigue. Advances and trends in parallel algorithms. In *Computational Mechanics — Advances and Trends*, pages 37–45. ASME, December 1986.

[12] James S. Ryan and Sisira Weeratunga. Parallel computation of 3-D Navier-Stokes flowfields for supersonic vehicles, AIAA paper 93-0064. In *31st Aerospace Sciences Meeting & Exhibit*, Reno, NV, January 11-14, 1993. AIAA.

[13] Richard S. Varga. *Matrix Iterative Analysis.* Automatic Computation Series. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1962. 7th printing.