SANDIA REPORT

SAND88—3291 • UC—405

Unlimited Release

Printed February 1990

A Time Stepping Algorithm for Parallel Computers

David E. Womble

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

***When printing a copy of any digitized SAND
Report, you are required to update the
markings to current standards.***

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

A Time Stepping Algorithm for Parallel Computers *

David E. Womble †

Abstract

Parabolic and hyperbolic differential equations are often solved numerically by time stepping algorithms. These algorithms have been regarded as sequential in time; that is, the solution on a time level must be known before the computation of the solution at subsequent time levels can start. While this remains true in principle, we demonstrate that it is possible for processors to perform useful work on many time levels simultaneously. Specifically, it is possible for processors assigned to “later” time levels to compute a very good initial guess for the solution based on partial solutions from previous time levels, thus reducing the time required for solution. The reduction in the solution time can be measured as parallel speedup.

We demonstrate this algorithm for both linear and nonlinear problems. In addition, we discuss the convergence properties of the method based on the convergence properties of the underlying iterative method, and we present an accurate performance model from which the speedup and other quantities can be estimated.

1 Introduction

One route to achieving the computing power required by scientists and engineers today is through the use of parallel computers. However, to use par-

*This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-76DP00789.

†Sandia National Laboratories, Albuquerque, NM 87185

allel computers effectively, existing algorithms must be reexamined to take advantage of inherent parallelism, and new algorithms must be developed and analyzed.

Time stepping methods are commonly used to numerically solve parabolic and hyperbolic partial differential equations (PDE's). In these methods, the solution to a PDE is determined at a specified set of times $t_1 < t_2 < \dots < t_N$ in sequence beginning with t_1 ; the solution at one time level is completed before the solution at the next time level is started. The discretization in space can be, for example, by finite differences or by finite elements, and the discretization in time is by one-sided finite differences. The alternative approach of solving on all time levels simultaneously is not considered practical [9].

Time stepping algorithms can be either explicit or implicit [1]. In explicit algorithms, the solution at each point in space depends only on the solutions at previous time levels, which are known. A high degree of parallelism can be achieved because the solution at each point on a time level can be calculated independently; however, explicit methods often suffer from severe restrictions on the size of the time step. In implicit algorithms, the solution at a point depends on the solutions at other points on the same time level, which are unknown. Although implicit methods do not suffer the same step size restriction as explicit methods, the degree of parallelism that can be achieved is reduced by the communication and synchronization requirements of solving simultaneously for each unknown at a time level [6].

Another method for the solution of time dependent PDE's is waveform relaxation, which was originally introduced as a numerical method for circuit simulation [4]. In this method, the space variables are discretized, and the time variable remains continuous. The resulting system of initial value problems is then solved using a line relaxation technique, such as Jacobi, Gauss-Seidel, or SOR. The degree of parallelism that can be achieved depends on the relaxation technique used; however, any relaxation technique requires a substantial amount of communication and memory [5].

Finally, we mention the windowed relaxation methods described in [7,8]. In these methods the spatial domain is divided among the processors, and within each subdomain, each processor computes iterates on several time levels (the window) before communicating the results to other processors. In a distributed memory machine, the effect of windowing is to decrease the number of messages and increase the size of the messages, thereby increasing

the processor efficiency.

In this paper, we introduce a method of parallelization for implicit time stepping algorithms. It is applicable to a wide range of problems (linear and nonlinear) and can be coupled with a wide range of existing algorithms, including finite element and finite difference algorithms. Further, the parallelism in our method is independent of any parallelism of the algorithm with which it is coupled. In Section 2, we present the parallel time stepping method. The method is analyzed in Section 3, and a performance model is developed. In Section 4, the method is demonstrated for both linear and nonlinear problems, and the performance is compared with the predictions of the model. In Section 5, we summarize the paper.

2 The Parallel Time Stepping Method

The parallel time stepping (PTS) method is a means by which parallelism can be introduced into a time stepping algorithm that uses an iterative method to find the solution at each time level. Specifically, while one or more processors are computing the solution on one time level, other processors can use intermediate solutions from this time level to improve the initial guesses for the solution on later time levels. The PTS method is very general in that it can be used with either linear or nonlinear PDE's, with any discretization of a PDE, and with any iterative method for the solution at a time level. The parallelism of the PTS method is independent of any parallelism available in the iterative method with which it is coupled; however, to simplify the presentation in this section, we assume that only one processor is assigned to each time line. An example including both space and time parallelism is included in Section 4.

We introduce the PTS method by considering the numerical solution of a linear, parabolic PDE that has been approximated by the sequence of linear systems

$$A_n u_n = f_n + B_n u_{n-1}, \quad n = 1, \dots, N, \quad (1)$$

where u_n and f_n are vectors in $\mathbf{R}^m$, A_n and B_n are $m \times m$ matrices, and u_0 is given. The subscript n denotes a time level, and the vector u_n denotes the approximate solution to the PDE at a discrete set of points on that time level. We note that the vector u_{n-1} must be known before we can compute the vector u_n .

Iterative methods, such as SOR and multigrid, are often used effectively in the solution of (1). If we let $u_n^{(k)} = Q_n(u_n^{(k-1)}, u_{n-1})$ denote the update step of such an algorithm, a serial time stepping method for the solution of (1) can be stated as follows.

Method 1 (serial time stepping) *Serial time stepping (STS) for the solution of (1) is given by*

```

for  $n = 1, \dots, N$  do
  set  $k = 0$ 
  set  $u_n^{(0)} = u_{n-1}$  ( $u_1^{(0)}$  is the initial condition)
  until convergence do
    compute  $k = k + 1$ 
    compute  $u_n^{(k)} = Q_n(u_n^{(k-1)}, u_{n-1})$ 
  end until
  set  $u_n = u_n^{(k)}$ 
end for

```

The number of iterations required for convergence at each time level is a function, in part, of the iteration function, Q_n , and the initial guess, $u_n^{(0)}$. If we assume that Q_n is the best iteration function available for the solution of this problem, the only improvement that can be made is the quality of the initial guess. To this end, let us suppose that the function $R_n(u_n^{(0)}, u_{n-1})$ can be used to refine the initial guess at time level t_n . Idle processors in a parallel computer might then use R_n to improve the initial guesses on time levels $t_{n+1}, \dots, t_N$ while the solution is being computed on time level t_n . For example, suppose we use three processors to solve on three time levels. Processor 1 solves the problem at the first time level by repeated evaluations of the iteration function Q_1 , and after each evaluation of Q_1 , sends the approximate solution to processor 2. Processor 2 uses this approximate solution at time level 1 to generate a new initial guess using the function R_2 . This initial guess is then sent to processor 3, which treats it as an approximate solution at time level 2 and generates a new initial guess for time level 3 using R_3 . After processor 1 has solved the problem on time level 1, processor 2 begins the solution process at time level 2 using its (improved) initial guess and the iteration function Q_2 while processor 3 continues to improve its initial guess using R_3 . Finally, after processor 2 has solved the problem at time level 2, processor 3 solves the problem at time level 3 using Q_3 .

Pseudocode for the PTS method is given below. The first loop in the pseudocode contains the evaluation of the function R_n and is executed until the solution on the previous time level has converged. The second loop corresponds to the loop in the serial time stepping method and is executed until the solution on the current time level has converged.

Method 2 (parallel time stepping) *If N processors are available for the solution of (1), then the parallel time stepping (PTS) method for processor n is given by*

```

set  $k = 0$ 
set  $u_n^{(0)}$  to the initial condition
if  $n \neq 1$  then
    until convergence on processor  $n - 1$  do
        compute  $k = k + 1$ 
        receive  $u_{n-1}^{(k-1)}$  from processor  $n - 1$ 
        compute  $u_n^{(k)} = R_n(u_n^{(k-1)}, u_{n-1}^{(k-1)})$ 
        send  $u_n^{(k)}$  to processor  $n + 1$ 
    end until
end if
set  $u_{n-1} = u_{n-1}^{(k)}$ 
set  $l = 0$ 
set  $u_n^{(0)} = u_n^{(k)}$ 
until convergence do
    compute  $l = l + 1$ 
    compute  $u_n^{(l)} = Q_n(u_n^{(l-1)}, u_{n-1})$ 
    send  $u_n^{(l)}$  to processor  $n + 1$ 
end until
set  $u_n = u_n^{(l)}$ 

```

There are many possible choices for the function R_n . For example, R_n might correspond to a multigrid algorithm in which a different combination of grids is used than for Q_n . (This would be useful because iterative methods eliminate different frequencies at different rates and different frequencies can propagate forward in time at different speeds.) One practical choice for the iteration function R_n is to set it equal to Q_n . There are several reasons for

this. First, we assume that Q_n was chosen because of desirable convergence properties for the problem to be solved. Second, because the $R_{n+1}, \dots, R_N$ are used concurrently with Q_n , setting $R_n = Q_n$ results in a load balanced algorithm. (Note that in many cases, the amount of work required for one application of the functions Q_n and R_n does not depend on n .) Third, the task of implementing the algorithm is simplified. Throughout the remainder of the paper, we will take $R_n = Q_n$.

In practice, we have P ($< N$) processors available for the solution of (1). In this case, processor p begins by computing the solution at time level t_p . When work has been completed on this level, it begins refining the initial guess on time level t_{P+p} and eventually computes the solution there. This process continues until the solution has been computed on each of the N time levels.

We note from the pseudocode above that processor n cannot begin work until a message has been received from processor $n - 1$, which occurs after processor $n - 1$ has completed one iteration. To formalize this, we introduce the concept of the delay at time level n , d_n , which we define to be the number of iterations that processor $n - 1$ completes before processor n starts. For the pseudocode listed above, $d_1 = 0$ and $d_n = 1$, $n = 2, \dots, N$. In the case of P ($< N$) processors, the delays, d_n , are unknown a priori for $n = P + 1, \dots, N$.

Even though we have developed the PTS method for a linear, parabolic PDE with a finite difference discretization, it is clear that the method can be used to parallelize any time stepping algorithm, including those for nonlinear problems, those for hyperbolic PDE's, and those that use finite element discretizations. The PTS method is demonstrated for a variety of problems in Section 4.

3 Analysis

The parallel time stepping method is very general. We have not specified the type of equation, the method of discretization, or the iterative solution algorithm. Thus, no one proof of convergence of the iteration $u_n^{(k)} = Q_n(u_n^{(k-1)}, u_{n-1})$, $k = 1, 2, \dots$, can be constructed. However, for a linear PDE, the convergence of the serial time stepping method (for a choice of discretization and iterative solution algorithm) implies the conver-

gence of the parallel time stepping method because the iteration defined by Q_n will converge for any initial guess.

We can develop a model for the behavior of the parallel time stepping method for linear PDE's. (Models can also be developed for nonlinear problems, but are highly problem dependent.) We assume that a linear PDE has been reduced to the form

$$A_n u_n = f_n + B_n u_{n-1}, \quad n = 1, \dots, N, \quad (2)$$

where u_n and f_n are vectors in $\mathbf{R}^m$, A_n and B_n are $m \times m$ matrices, u_0 is known, and A_n is nonsingular. We let

$$u_n^* = A_n^{-1}(f_n + B_n u_{n-1})$$

be the solution to (2) at time level n and define an iterative method for obtaining u_n^* by

$$u_n^{(k)} = Q_n(u_n^{(k-1)}, u_{n-1}). \quad (3)$$

To guarantee the convergence of $u_n^{(k)}$ to u_n^* , we require that Q_n satisfy the Lipschitz condition

$$\|Q_n(u, v) - Q_n(u^*, v)\| \leq \beta_n \|u - u^*\|, \quad \beta_n < 1,$$

for all u and v in $\mathbf{R}^m$ and some norm $\|\cdot\|$ on $\mathbf{R}^m$. These definitions form the traditional framework for the study of the STS method. To study the PTS method, we adopt the convention that $u_0^{(k)} = u_0$, $k = 1, \dots, \infty$, and define

$$u_n^{(k),*} = \begin{cases} A_n^{-1}(f_n + B_n u_{n-1}^{(k+d_n)}), & k = 1, \dots, I_{n-1} - d_n \\ A_n^{-1}(f_n + B_n u_{n-1}^{(I_{n-1})}), & k > I_{n-1} - d_n \end{cases},$$

where d_n is the delay defined in the previous section, and I_{n-1} is the number of iterations required for convergence of the iteration (3) to the solution of (2) at time level $n - 1$. In the notation of Method 2, I_n is the number of times R_n is evaluated plus the number of times Q_n is evaluated. The vector $u_n^{(k),*}$ is thus the solution to (2) with the true solution at time level $n - 1$ replaced by the most recent iterate, and we note that $u_n^{(k),*} = Q_n(u_n^{(k),*}, u_{n-1}^{(k+d_n)})$. Finally, we define $e_n^{(k)}$ by

$$e_n^{(k)} = \begin{cases} \|u_n^{(k)} - u_n^{(k),*}\|, & k = 1, \dots, I_n \\ 0, & k > I_n \end{cases}.$$

For k between 1 and I_n , $e_n^{(k)}$ is the norm of the difference between the iterate $u_n^{(k)}$ and the “apparent” true solution $u_n^{(k),*}$, which we refer to as the apparent error. For $k > I_n$, we set $e_n^{(k)} = 0$ to simplify error bound on later time levels.

We now derive a bound for $e_n^{(k)}$. Applying the Lipschitz condition on Q_n for $n = 1$ yields

$$e_1^{(k)} \leq \beta_1^k e_1^{(0)},$$

and for $n > 1$,

$$\begin{aligned} e_n^{(k)} &\leq \beta_n \|u_n^{(k-1)} - u_n^{(k),*}\| \\ &\leq \beta_n \left(\|u_n^{(k-1)} - u_n^{(k-1),*}\| + \|u_n^{(k-1),*} - u_n^{(k),*}\| \right) \\ &\leq \beta_n \left(e_n^{(k-1)} + \|A_n^{-1} B_n\| \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n)}\| \right) \\ &\leq \beta_n \left(e_n^{(k-1)} + (1 + \beta_{n-1}) \|A_n^{-1} B_n\| \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n),*}\| \right). \end{aligned}$$

Applying the above calculations recursively yields

$$e_n^{(k)} \leq \beta_n \left(e_n^{(k-1)} + \sum_{i=1}^{n-1} \left(e_i^{(k-1+\sum_{j=i}^{n-1} d_{j+1})} \prod_{j=i}^{n-1} (1 + \beta_j) \|A_{j+1}^{-1} B_{j+1}\| \right) \right). \quad (4)$$

We note from (4) and the definition of $e_n^{(k)}$ that if $d_n = I_{n-1}$, then we recover the traditional error bounds for the STS method. Using induction, we can also conclude from (4) that the PTS method converges (for linear problems).

We also note from (4) that the effect of the error on one time level can be magnified at all later time levels on which the computation is proceeding simultaneously. For many practical problems, this magnification factor is greater than one. Hence, our upper bound on the error allows the possibility that the PTS method on N processors requires more time to solve a problem than does the STS method on one processor.

The upper bound on the error (4) is not tight; however, the error can be approximated. Because, the iterates, $u_{n-1}^{(k)}$, asymptotically approach u_{n-1}^* along a vector lying in the subspace spanned by the eigenvectors corresponding to the largest eigenvalue, the distance between consecutive iterates asymptotically approaches $1 - \beta_{n-1}$. Replacing the term $1 + \beta_{n-1}$ in the upper bound with $1 - \beta_{n-1}$ yields

$$\begin{aligned} e_n^{(k)} &\leq \beta_n \left(e_n^{(k-1)} + \|A_n^{-1} B_n\| \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n)}\| \right) \\ &\approx \beta_n \left(e_n^{(k-1)} + (1 - \beta_{n-1}) \|A_n^{-1} B_n\| e_{n-1}^{(k-1+d_n)} \right). \end{aligned}$$

Thus, $e_n^{(k)}$ is approximated by $\sigma_n^{(k)}$, the solution to the recursion

$$\sigma_n^{(k)} = \begin{cases} \beta_n \sigma_n^{(k-1)} + \alpha_n \sigma_{n-1}^{(k-1+d_n)}, & \sigma_{n-1}^{(k-1+d_n)} \geq \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

for $k = 1, 2, \dots$ and $n = 1, 2, \dots$ with the initial conditions

$$\sigma_0^{(k)} = 0, \quad k = 0, 1, \dots,$$

$$\sigma_n^{(0)} = \|u_n^* - u_{n-1}^*\| + \gamma_n \sigma_{n-1}^{(d_n-1)}, \quad n = 1, 2, \dots,$$

where

$$\alpha_n = \beta_n(1 - \beta_{n-1})\|A_n^{-1}B_n\|,$$

$$\gamma_n = 1 + \|A_n^{-1}B_n\|,$$

and ϵ corresponds to the convergence criterion. This recursion can be solved in closed form; however, the this form does not yield additional information. Instead, we will evaluate the recursion relation numerically for specific cases. The approximations for the error will be used to predict speedups, which will be compared with experimental results in the next section.

The delays can be used to generalize the model to the case of P ($< N$) processors. For example, to model the one processor case (STS method), we can set d_n equal to I_n , the minimum k such that $\sigma_n^{(k)} = 0$. To model the P processor case, the delays on the first P time levels are arbitrary, and the delay at time n ($> P$) is chosen so that computation does not start until the solution on line $n - P$ has been completed. We note that the maximum number of processors that can be used without at least one processor being idle at all times must satisfy

$$P \leq I_n / \sum_{j=n+1}^{n+P} d_j. \quad (6)$$

for all n between 1 and $N - P$. This relation states that the processor assigned to time level n cannot complete its calculations before each of the remaining processors is assigned a time level and begins iterating.

The most common performance measure is speedup. It is normally defined as the time required for one processor to solve a problem divided by the time required for P processors to solve the same problem. The time to solve a

problem using either Method 1 or Method 2 is proportional to the “effective” number of evaluations of Q_n , that is, the number of evaluations of Q_n that are not overlapped with computations on previous time levels. The effective number of evaluations of Q_n is given by $\sum_{n=1}^N E_n$, where $E_n = I_n - I_{n-1} + d_n$. If we denote by $I_n(P)$ the number of iterations and by $E_n(P)$ the effective number of iterations required for convergence at time level n and by $d_n(P)$ the delay at time level n in the P processor case, $S(P)$, the speedup on P processors, is given by

$$\begin{aligned} S(P) &= \left(\sum_{n=1}^N I_n(1) \right) / \left(\sum_{n=1}^N E_n(P) \right) \\ &= \left(\sum_{n=1}^N I_n(1) \right) / \left(I_N(P) + \sum_{n=1}^N d_n(P) \right). \end{aligned} \quad (7)$$

For some problems, $I(1) = I_1(1) = \dots = I_N(1)$ are constant. In this case, there is a “steady state” solution of (5) in which $I(P) = I_1(P) = \dots = I_N(P)$ are constant, and $d(P) = I(P)/P$, and the “steady state” speedup is given by

$$SS(P) = \lim_{N \rightarrow \infty} S(P) = P \times I(1) / I(P). \quad (8)$$

The steady state speedup is an asymptotic value for the speedup (as the number of time levels at which the solution is desired increases). The “transient” nature of the speedup for a small number of time levels is due to the fact that none of the iterations at the first time level can be overlapped with iterations at previous time levels.

We now look at the effect of the parameters in the model on the performance of the PTS method. The delays, the terms $\|A_n^{-1}B_n\|$, and the convergence factors, β_n , have the largest effect, while the effect of the other parameters is minimal. The results presented in the remainder of this section are obtained by numerically evaluating the recursion relation, (5). The values of the parameters used in the evaluation of (5) are close to values seen in many applications, and the effects shown in the remainder of this section are observed over a range of values for the parameters.

The delay is a function of both hardware and software. It depends on the number of processors, the number of iterations that each processor requires for convergence, and the time to “start up” a processor on a new time level. In almost all cases, we want the minimum delay possible; nevertheless, it is

instructive to consider the effect of the delay on the number of iterations. As the delay, d_n , is decreased, the error introduced as the result of error at the previous time level is increased, and we expect that the number of iterations required for convergence, I_n , will increase. However, the numerical evaluation of (5) indicates that the “effective number of iterations,” $E_n = I_n - I_{n-1} + d_n$, will be reduced. This is shown in Figure 1.

Figure 1: The effect of the delay, d_2 , on the number of iterations, I_2 , and the effective number of iterations, E_2 . ($\beta_2 = .9$, $\alpha_2 = .09$, $\|u_2^* - u_1^*\| = .1$, $\epsilon = .00001$.) Note that $I_1 = 88$.

The terms $\|A_n^{-1}B_n\|$ are determined by the PDE and the method of discretization. We see from (5) that increasing the value of $\|A_n^{-1}B_n\|$ magnifies the effect of the error at the previous time level. The result is that the number of iterations required at each time level increases, the delays, d_n ($n > P$), increase, and the speedup decreases. Noting equation (8) and the effect of the delays on the number of iterations shown in Figure 1, we expect the decrease in speedup to be much more severe for a large number of processors. This effect is shown in Figure 2. There is a slight “staircase” nature to the curves, which is due to the fact that d_n and I_n are integers.

Figure 2: The effect of $\|A_n^{-1} B_n\|$ on the steady state speedup. ($\beta_n = .9$, $\|u_n^* - u_{n-1}^*\| = .1$, $\epsilon = .00001$, $P = 16$.)

The convergence factors β_n are determined by the iterative method chosen and affects both the number of iterations required for convergence and the magnification of the error on the previous time level. As the β_n approach one, errors from previous time levels are introduced faster than they can be eliminated. Thus, the effective number of iterations increases, and we expect the speedup to decrease. On the other hand, as β_n approaches zero, the number of iterations required for convergence decreases, and we see from equation (6) that the number of processors that can be used effectively decreases. Thus, we expect a decrease in the speedup. The overall effect of changing β_n on the steady state speedup is shown in Figure 3. We note that the speedup achieves its maximum on the interior of the interval $(0, 1)$, and that the PTS method performs best with relatively good iterative algorithms. As before, the staircase nature of the curves is due to the fact that d_n and I_n are integers.

Figure 3: The effect of β_n on the steady state speedup. ($\|A_n^{-1}B_n\| = 1$, $\|u_n^* - u_{n-1}^*\| = .1$, $\epsilon = .00001$, $P = 16$.)

4 Experimental Results

In this section, we present three examples of the PTS method. The first example is a parabolic PDE and an iterative method for which the parameters in the model, equation (5), can be calculated analytically. This allows direct comparison of the performance of the PTS method with the predictions of the model. The second example is a nonlinear, parabolic PDE with multi-grid as the underlying iterative method. This example demonstrates that the PTS method is applicable to a wide variety of problems. In the third example, we demonstrate that the PTS method can be effectively coupled with a parallel implementation of an iterative algorithm at each time level. All numerical results were obtained on the NCUBE/ten hypercube. For ease of programming, we assume that the number of processors to be used is a power of two, although this is not a requirement for the PTS method.

Example 1 The first example is the PDE

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial t} &= 3, & (x, t) \in (0, 1) \times (0, 5), \\ u(x, 0) &= 0, & x \in (0, 1), \\ u(0, t) &= 3t = u(1, t), & t \in (0, 5). \end{aligned} \tag{9}$$

We let $\Delta x = 1/64$ and $\Delta t = 5/200$ and replace (9) with the finite difference approximation

$$\begin{aligned} \frac{-u_{i+1,n} + 2u_{i,n} - u_{i-1,n}}{\Delta x^2} + \frac{u_{i,n} - u_{i,n-1}}{\Delta t} &= 3 \\ i &= 1, \dots, 63, \quad n = 1, \dots, 200, \\ u_{i,0} &= 0, \quad i = 1, \dots, 63, \\ u_{0,n} &= 3n\Delta t = u_{M,n}, \quad n = 1, \dots, 200. \end{aligned}$$

This yields a linear system of the form (1), which we solve with SSOR iteration with the near optimal relaxation parameter, $\omega = 1.8$. The parameters needed to evaluate equation (5) can be calculated analytically. They are

$$\beta_n = .83, \quad \|A_n^{-1}B_n\| = .80, \quad \|u_n^* - u_{n-1}^*\| = .075, \quad n = 1, \dots, 200.$$

For the convergence criterion, we let $\epsilon = 1 \times 10^{-6}$. Convergence can be checked explicitly because the true solution to (9) is known.

The problem was solved numerically using different numbers of processors. The results and the predictions of the model are shown in Table 1. Note that problem (9) satisfies the requirements necessary to compute the steady state speedup. As was stated in the previous section, the steady state speedup is an asymptotic value for the speedup as the number of time levels on which the solution is desired increases.

P	$\sum_{n=1}^{200} E_n$		$S(P)$		$SS(P)$	
1	12,000	(12,200)	1.00	(1.00)	1.00	(1.00)
2	6,118	(6,121)	1.96	(1.99)	2.00	(2.00)
4	3,168	(3,201)	3.79	(3.81)	3.90	(3.87)
8	1,847	(1,915)	6.49	(6.37)	6.53	(6.78)
16	1,234	(1,319)	9.72	(9.25)	9.50	(10.8)
32	872	(1,137)	13.7	(10.7)	13.9	(15.2)
64	634	(1,137)	18.7	(10.7)	19.1	(20.0)
128	489	(1,137)	24.5	(10.7)	27.6	(23.6)

Table 1: Comparison of the performance of the PTS method for Example 1 with the predictions of the model. The predictions are in parentheses.

We make two observations based on Table 1. The first is that with only 200 time steps, we cannot use more than 200 processors. The second observation is that the model is most accurate for $P \ll N = 200$. One reason is that the delays are larger, and small errors in modeling the delay have a smaller effect. (We recall that the delays are dependent, in part, on the hardware and must be modeled.) Another reason is that we used the approximation

$$\|Q_n(u, v) - Q_n(u_n^*, v)\| \approx \beta_n \|u - u_n^*\|, \quad \beta_n < 1,$$

which is most accurate for a large number of iterations (with β_n equal to the spectral radius of Q_n). For $P = 128$, there is a average of less than three effective iterations per timestep.

Example 2 As a second example, we choose Burgers' equation:

$$\nu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} - \frac{\partial u}{\partial t} = 0, \quad (x, t) \in (0, 1) \times (0, 5) \quad (10)$$

$$\begin{aligned}u(x, 0) &= \sin(\pi x), & x &\in (0, 1), \\u(0, t) &= 0 = u(1, t), & t &\in (0, 5).\end{aligned}$$

We let $\Delta x = 1/128$ and $\Delta t = 5/200$ and replace (10) with the finite difference approximation to get

$$\nu \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{\Delta x^2} - u_{i,n} \frac{u_{i,n} - u_{i-1,n}}{\Delta x} - \frac{u_{i,n} - u_{i,n-1}}{\Delta t} = f_{i,n}, \quad (11)$$

$$i = 1, \dots, 127, \quad n = 1, \dots, 200,$$

$$\begin{aligned}u_{i,0} &= \sin(i\pi\Delta x), & i &= 0, \dots, 128 \\u_{0,n} &= 0 = u_{128,n}, & n &= 1, \dots, 200.\end{aligned}$$

We note that upwind differencing has been used for the term $\partial u / \partial x$.

As an iterative method for the solution of (11), we choose multigrid iteration with a weighted Jacobi smoothing step and a weighting factor of $\omega = .95$. Because multigrid iteration requires a linear equation, we delay $u_{i,n}$ in the nonlinear term by one cycle. We consider the iterations to have converged if the residual is less than 1.0×10^{-5} . Table 2 shows the results obtained by running the PTS algorithm for this problem with $\nu = .01$ on different numbers of processors. There are no predicted results because the problem is nonlinear.

P	$\sum_{n=1}^{200} E_n$	$S(P)$
1	5,346	1.00
2	2,732	1.96
4	1,341	3.99
8	765	7.00
16	570	9.38
32	505	10.6
64	487	11.0
128	487	11.0

Table 2: The performance of the PTS algorithm for Example 2 with $\nu = .01$.

We note that the performance of the PTS algorithm for Example 2 is somewhat worse than that for Example 1. The reason for this is that fewer

iterations are required at each time step in Example 2. Thus, fewer processors can be used effectively. In general, for problems that have steady state solutions, the number of iterations decreases as n increases (as in Example 2), and as a result, fewer processors can be used effectively at later time levels.

Example 3 The third example is a linear, parabolic PDE that arises in the study of grain-boundary diffusion [3]. The dimensionless equation with parameters approximating the diffusion of chromium in gold is

$$0.01 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = \frac{\partial u}{\partial t}, \quad (x, y, t) \in (0, 1) \times (0, \frac{1}{2}) \times (0, 1), \quad (12)$$

with boundary conditions

$$\frac{\partial u}{\partial t}(x, 0, t) = \frac{\partial^2 u}{\partial x^2}(x, 0, t) + 0.1 \frac{\partial u}{\partial y}(x, 0, t), \quad x \in (0, 1), \quad t \in (0, 1),$$

$$\frac{\partial u}{\partial x}(x, \frac{1}{2}, t) = 0, \quad x \in (0, 1), \quad t \in (0, 1),$$

$$u(0, y, t) = 1, \quad y \in (0, \frac{1}{2}), \quad t \in (0, 1),$$

$$u(1, y, t) = 0, \quad y \in (0, \frac{1}{2}), \quad t \in (0, 1),$$

and initial conditions

$$u(x, y, 0) = 0, \quad x \in (0, 1], \quad y \in (0, \frac{1}{2}],$$

$$u(0, y, 0) = 1, \quad y \in (0, \frac{1}{2}),$$

$$u(x, 0, 0) = 1 - x, \quad x \in (0, 1).$$

We let $\Delta x = 1/32$, $\Delta y = 1/64$, and $\Delta t = 1/32$, and replace (12) with an implicit finite difference approximation (implicit Euler difference in time, central differences in space where possible, one-sided differences otherwise) to get a system of linear equations of the form (1). As a parallel iterative algorithm for the solution at each time level, we choose Jacobi iteration because of its inherent parallel nature and consider the iterations to have converged when the residual is less than 1.0×10^{-3} . Table 3 shows the run

$P_x \times P_y$	P_t				
	1	2	4	8	16
1×1	467. (1.00)	264. (1.77)	178. (2.62)	145. (3.22)	124. (3.77)
2×2	135. (3.46)	76.1 (6.13)	52.0 (8.98)	42.6 (11.0)	35.9 (13.0)
4×4	47.3 (9.87)	26.6 (17.6)	18.4 (25.4)	15.0 (31.1)	12.6 (37.1)
8×8	24.1 (19.4)	13.8 (33.8)	9.44 (49.5)	7.73 (60.4)	6.64 (70.3)

Table 3: The performance of the PTS algorithm for Example 3. P_x and P_y are the numbers of processors in the x and y direction respectively, and P_t is the number of time levels on which iterations are carried out simultaneously. The total number of processors used is $P_x \times P_y \times P_t$. Run times are in seconds, and speedups are given in parentheses.

times and speedups for runs with various combinations of processors in the space and time directions.

Even though Jacobi iteration is considered a highly parallel algorithm, a problem size of 32×32 is small and communication overhead is significant. We see from Table 3 that if only a small number of processors are available for the solution of (12), then they are most effectively used by the parallel Jacobi algorithm to solve at one time level. However, if a large number of processors are available, then they are most effectively used when the Jacobi iteration is coupled with the PTS method. This effect would be more pronounced if we had chosen an iterative method with a less efficient parallel implementation, such as multigrid or SOR.

We can also see from Table 3 that the total speed up is approximately equal to the speedup obtained in the space variables times the speedup obtained in the time variable.

5 Summary

In this paper, we have presented a technique by which parallelism in the time direction can be introduced into implicit time stepping algorithms. The attraction of the technique is that it can be coupled with a wide variety of algorithms and that the parallelism introduced in the time direction is

independent of any parallelism in space. Thus, the number of processors that can be efficiently applied to the solution of a time dependent PDE is increased by at least an order of magnitude.

We also presented a performance model of the method for linear problems. Based on this model, we were able to predict the effect of algorithm parameters, such as the convergence rate and the number of processors used, on the speedup. In Section 4, this model was found to be in good agreement with the actual performance of the method.

Finally, we demonstrated the PTS method for linear and nonlinear problems and for three common iterative methods. We found that the method was very effective for a small number of processors and remained effective while the number of processors was less than the number of time levels on which the solution was desired and less than the number of iterations required for convergence on a time level. We also demonstrated that the PTS method can be effectively coupled with parallel iterative algorithms for the solution at each time level.

Acknowledgment. The author would like to thank E. F. Brickell and D. E. Amos of Sandia National Laboratories for their help in the analysis of equation (5).

References

- [1] W. F. Ames. *Numerical Methods for Partial Differential Equations*. Academic Press, Inc., New York, 1977.
- [2] G. J. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MA, 1983.
- [3] P. H. Holloway, D. E. Amos, and G. C. Nelson. Analysis of grain-boundary diffusion in thin films: chromium in gold. *J. Appl. Phys.*, 47(9):3769–3775, 1976.
- [4] E. Lelarasmee, A. Ruheli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for the time domain analysis of large scale integrated circuits. *IEEE Trans. Computer-Aided Design of ICAS*, CAD-1(3):131–145, 1982.
- [5] A. R. Newton and A. L. Sangiovanni-Vincentelli. Relaxation-based electrical simulation. *SIAM J. Sci. Stat. Comp.*, 4(3):485–524, 1983.
- [6] J. M. Ortega and R. G. Voigt. *Solution of Partial Differential Equations on Vector and Parallel Computers*. SIAM, Philadelphia, PA, 1985.
- [7] J. H. Saltz and V. K. Naik. Towards developing robust algorithms for solving partial differential equations on MIMD machines. *Par. Comp.*, 6(1):19–44, 1988.
- [8] J. H. Saltz, V. K. Naik, and D. M. Nicol. Reduction of the effects of the communication delays in scientific algorithms on message passing MIMD architectures. *SIAM Journal of Scientific And Statistical Computing*, 8(1):s118–s134, 1987.
- [9] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice Hall, Inc., Englewood Cliffs, N.J., 1973.

Internal Distribution

E. H. Barsis (5)	1400
W. J. Camp (5)	1420
D. B. Holdridge	1421
R. C. Allen	1422
D. E. Amos	1422
L. S. Baca	1422
D. D. Cline	1422
D. E. Womble (15)	1422
A. J. Cleary	1422
R. S. Tuminaro	1422
E. F. Brickell	1423
C. A. Phillips	1423
R. E. Benner	1424
G. M. Pollock	1424
M. P. Sears	1424
J. N. Shadid	1424
C. T. Vaughan	1424
S. J. Plimpton	1424
S. A. Landenberger (5)	3141
C. L. Ward (8)	3141-1
W. I. Klein (3)	3151
J. A. Wackerly	8524