Asynchronous and Corrected-Asynchronous Numerical Solutions of Parabolic PDEs on MIMD Multiprocessors *†

appeared in Numerical Algorithms, Vol. 6, No. III-IV, pp. 275-296, 1994

Dganit Amitai
Amir Averbuch
Samuel Itzikowitz ‡
Eli Turkel §

School of Mathematical Sciences
Sackler Faculty of Exact Sciences
Tel-Aviv University
Tel-Aviv 69978, Israel

Abstract

A major problem in achieving significant speed-up on parallel machines is the overhead involved with synchronizing the concurrent processes. Removing the synchronization constraint has the potential of

*This research was partially supported by a grant from The Basic Research Foundation administrated by The Israel Academy of Sciences and Humanities

†A reduced version of the paper was presented at the Fourth SIAM Conference on Parallel Processing for Scientific Computing, Dec 11-13, 1989, Chicago, USA.

‡The work by this author was supported by research grant 337 of the Israeli National Council for Research and Development.

§This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18107 while the author was in residence at the Institute for Computer Applications in Sciences and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

speeding up the computation. We present asynchronous (AS) and corrected-asynchronous (CA) finite difference schemes for the multi-dimensional heat equation. Although our discussion concentrates on the Euler scheme for the solution of the heat equation, it has the potential of being extended to other schemes and other parabolic PDEs. These schemes are analyzed and implemented on the shared-memory multi-user *Sequent* Balance machine. Numerical results for one and two dimensional problems are presented. It is shown experimentally that synchronization penalty can be about 50% of run time: in most cases, the asynchronous scheme runs twice as fast as the parallel synchronous scheme. In general, the efficiency of the parallel schemes increases with processor load, with the time-level, and with the problem dimension. The efficiency of the AS may reach 90% and over, but it provides accurate results only for steady-state values. The CA, on the other hand, is less efficient but provides more accurate results for intermediate (non steady-state) values.

Key-Words:
Asynchronous finite difference schemes, parallel finite difference schemes, parallel MIMD and multiuser machines, parallel steady-state and non steady-state parabolic solutions, synchronization overhead.

1 Introduction

As parallel machines become more popular, most algorithms used on parallel machines still rely heavily on synchronizing the concurrent processes. There is an inherent inefficiency in the synchronization requirement, which is two-fold. First, a fast processor is delayed until the slowest processor finishes. Thus, the pace of the algorithm is dictated by the slowest processor. There are various reasons why certain processors will be ahead of the others, even when they are physically configured at the same speed.

- Random noise
  Any unpredictable perturbation that can cause a momentary delay.
- Multi-user environment
  In such an environment many tasks are simultaneously assigned to each processor. Since a single processor can be used for any of those tasks

at any particular time, we cannot predict how much will be devoted for the computation of our algorithm.

- Master/Slave
  In such an environment one processor, the *master*, is doing additional tasks to those performed by the other processors for example, i/o operations, scheduling or load balancing tasks. Therefore, it may be slow in performing our algorithm.

- Load Balancing
  In many problems it may be logical to divide the problem unequally among the processors due to different boundary conditions or approximation schemes that are used. In general the partition of nodes among processors is dictated by numerical and physical considerations rather than just by computer architecture considerations. This extends the duration of an iteration for those processors that are assigned to do more work and therefore those would be slower than the others.

Second, there is a delay period associated with the synchronization mechanism itself whether it is setting the semaphores in a shared memory environment or waiting on a message to arrive in a message-passing environment, or any other possible implementation (i.e. when possible, setting a time bound on the duration of an iteration, so that the next iteration starts only after this time bound is exhausted). No matter what implementation is used, a slow communication channel slows the progress of the entire computation. Moreover, in some situations synchronization may cause contentions over communication resources and memory access, that require careful implementation using additional mechanisms such as *locking*.

A particular case where tasks are to be repeated is that of iterative computation. Numerical properties of iterative solutions to PDEs are usually based on the assumption that the iterations are *synchronized*. This is equivalent to assuming that the algorithm is governed by a global clock so that the start of each iteration is simultaneous for all processors. In implementing a synchronous algorithm in an inherently asynchronous architecture a *synchronization mechanism* is used in order to guarantee the correct execution. This considerably degrades the efficiency of those algorithms. An asynchronous algorithm can potentially reduce the synchronization penalty since each processor can execute more iterations when it is not constrained to wait for the most recent results of the computation in other processors. In addition, asynchronous algorithms eliminate the programming efforts involved in setting up and debugging the synchronization mechanism and also simplifies the task management.

The usage of *asynchronous iterations* for an iterative solution of systems of linear equations, is due to [6]. Asynchronous Iterative Methods for Multiprocessors are also discussed in [4], and are used for the solution of ordinary differential equations by [11]. In this paper, we present an asynchronous iterative methods for the solution of partial differential equations. These methods are based on Euler explicit finite difference schemes. In Section 2 we present the parabolic PDE which will serve as our model problem, the corresponding finite difference scheme and its asynchronous parallel modification. Section 3 analyzes our asynchronous scheme and determine the conditions under which the asynchronous iterations work. To compensate for the inaccuracy of non steady-state values of this asynchronous scheme, we propose and discuss in Section 4 the corrected-asynchronous scheme, that while still being asynchronous it performs some extra extrapolation calculations. Numerical results are presented in Section 5. Finally, our results are summarized in Section 6.

2 The Problem and its Asynchronous Solution

We demonstrate our approach for asynchronous solution on the multi dimensional heat equation. The same approach can be extended and generalized to other types of problems. For our model problem we consider the simplest parabolic equation, the heat equation in d-variables,

$$\frac{\partial u}{\partial t} = \sum_{s=1}^{d} a_s^2 \frac{\partial^2 u}{\partial {x_s}^2} \tag{1}$$

in a rectangular domain, with accompanying initial and boundary conditions, where a_s^2 $(1 \le s \le d)$ are constant positive coefficients. For our model problem we consider Dirichlet boundary condition. However, other conditions are equally applicable. A one dimensional model problem is,

$$\frac{\partial u}{\partial t} = a_1^2 \frac{\partial^2 u}{\partial x^2} \quad \text{for} \quad 0 < x < 1 \quad 0 < t \le T$$

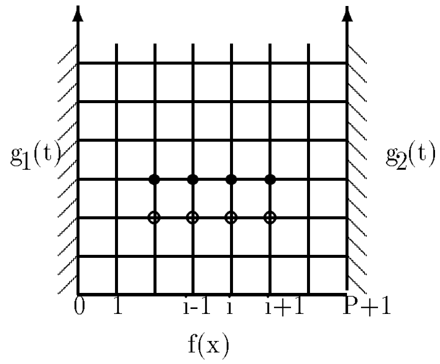

Figure 1: Model problem - Domain discretization

with initial condition,

$$u(x,0) = f(x) \qquad 0 \leq x \leq 1$$

and Dirichlet boundary conditions,

$$u(0,t) = g_1(t) \qquad 0 < t \leq T$$

$$u(1,t) = g_2(t) \qquad 0 < t \leq T$$

where a_1^2 is a constant positive coefficient.

After discretizing the domain (see Fig. 1) we approximate Eq. (1) at grid-point $(\vec{x}_i, t_n)$ by the forward Euler finite difference approximation:

$$v_i^{n+1} = v_i^n + \sum\nolimits_{s=1}^{d} r_s \delta_s^2 v_i^n \tag{2}$$

where, $r_s = \frac{a_s^2 \Delta t}{(\Delta x_s)^2} \qquad 1 \leq s \leq d$ are held as constants; i $(1 \leq i \leq p)$ denotes index of the spatial coordinate $\vec{x}$ of a particular grid point; d denotes the dimension of the problem; and the operator δ_s^2 denotes a central difference in the direction of s.

The one dimensional version of this scheme is,

$$v_i^{n+1} = v_i^n + r(v_{i-1}^n - 2v_i^n + v_{i+1}^n)$$

where, $r = \frac{a^2 \Delta t}{(\Delta x)^2}$ is held constant, and the values of v_i^0 $(1 \leq i \leq p)$, v_0^n, and v_{p+1}^n are determined by the initial and boundary conditions, respectively.

It is well known that the scheme of Eq. (2) is stable if

$$0 < \sum_s r_s \leq \frac{1}{2} \tag{3}$$

Although the following discussion concentrates on the Euler scheme for the solution of the heat equation, it has the potential of being extended to other schemes and other equations. In this paper we only consider the simple scheme (2). Extensions to other explicit schemes are straightforward. We emphasize in this study the analysis of asynchronous schemes. Comparisons for ADI schemes will be performed in a work in preparation.

2.1 Asynchronous (AS) Model

We now present the model of *asynchronous iterations* to the numerical problem described. The definition of *chaotic iteration* was presented in [6]. The formal definition of *asynchronous iteration* presented below is a modified version of what is discussed in [4], [7], and [11].

Definition 1 *Let F_i be a computational task to be done on the i^{th} component of a given vector $\vec{u}$, taking as its arguments components of $\vec{u}$ in the neighborhood of i. Asynchronous iterations $(F, \vec{u}, J, A)$ corresponding to these tasks and starting with a given vector, $\vec{u}^0$, are a sequence of vectors defined recursively by:*

$$u_i^{n+1} = \begin{cases} u_i^n & \text{if} \quad i \notin J_n \\ F_i(u_1^{n_i+d(1,i,n)}, u_2^{n+d(2,i,n)}, \ldots, u_L^{n+d(L,i,n)}) & \text{if} \quad i \in J_n \end{cases} \tag{4}$$

where, n is a **global** *value of an iteration level, $\vec{u} = (u_1, \ldots, u_L)$, $F = (F_1, \ldots, F_L)$, $J = \{J_n | n = 1, 2, \ldots\}$ is a sequence of non-empty subsets of $\{1, \ldots, L\}$, $A = \{A_1, \ldots, A_L\}$, and A_i is a sequence of elements in $\mathbb{Z}^L$, $A_i = \{(d(1,i,n), \ldots, d(L,i,n)) | n = 1, 2, \ldots\} \quad \forall i = 1, \ldots, L$.*

No assumptions are made on the relations between the calculations of the grid points, except a non-starvation condition which guarantees that the i^{th} grid point is updated an infinite number of times, i.e i occurs infinitely often in the sets J_n. This means that no point is abandoned forever, and consequently, no processor is held forever executing the same iteration. Synchronous iterations are obtained from this model when $d(k,i,n) = 0 \quad \forall k,i,n$.

A single or several grid points may be assigned to each processor of a parallel MIMD machine with p asynchronous processors. For simplicity also assume that values stored by each processor are available to the rest by means of a shared memory. Yet, with some minor modifications this model is equally applicable to message-passing architectures, where in fact, the overhead of synchronization is more significant.

Let us now take the **local** point of view and consider the i^{th} grid point which is assigned, perhaps with some of its neighbors, to a specific processor. We present the following modified asynchronous difference equation for the multi-dimensional heat equation:

$$u_i^{n_i+1} = u_i^{n_i} + \sum_{s=1}^{d} r_s \hat{\delta}_s^2 u_i^{n_i} \tag{5}$$

Here, n_i denotes the last completed iteration at the i^{th} grid point and $\hat{\delta}_s^2 u_i^{n_i}$ is a central difference of the value of u_i at the n_i^{th} iteration, using the most recent available values of the neighboring points. In the s^{th} coordinate direction the neighboring values are denoted by $u_{i-1_s}^{n_i+\alpha_{i_s}^{n_i}}$ and $u_{i+1_s}^{n_i+\beta_{i_s}^{n_i}}$, respectively.

The values of $\alpha_{i_s}^{n_i}$ and $\beta_{i_s}^{n_i}$ corresponds to the iteration - delay or - haste of the neighboring grid point along the s-coordinate axis, relative to the i^{th} grid point, when evaluating its $(n_i+1)^{th}$ iteration.

For example, in one dimension we obtain

$$u_i^{n_i+1} = u_i^{n_i} + r(u_{i-1}^{n_i+\alpha_i^{n_i}} - 2u_i^{n_i} + u_{i+1}^{n_i+\beta_i^{n_i}})$$

where, $r = \frac{a^2 \Delta t}{(\Delta x)^2}$.

For our analysis we require α and β to be bounded, i.e. no node falls infinitely behind its neighbors. We note that for each iteration, α and β are functions of i, the number of the processor, and not of x. Hence as we add more intermediate points, the difference between the adjacent iteration levels does not approach zero.

3 Analysis of the Asynchronous Scheme

Lemma 1 *When α and β are bounded, the asynchronous finite difference approximation Eq. (5) is consistent with the following heat equation:*

$$\frac{\partial u}{\partial t} = K(\vec{x}, t)\nabla \cdot [\nabla u] \tag{6}$$

where, $K(\vec{x}, t) = \frac{1}{1 - \sum_{s=1}^{d} r_s[\alpha_s(\vec{x}, t) + \beta_s(\vec{x}, t)]}$, *and* $r_s = \frac{a_s^2 \Delta t}{(\Delta x_s)^2}$ *are constants.*

Proof: Taylor series expansion of Eq. (5) yields,

$$\begin{aligned} \underbrace{u_i^{n_i+1} = u_i^{n_i} + \sum_{s=1}^{d} r_s \delta_s^2 u_i^{n_i}}_{original \quad scheme} \quad &+ \underbrace{[\sum_{s=1}^{d} r_s(\alpha_s^{n_i} + \beta_s^{n_i})](u_i^{n_i+1} - u_i^{n_i})}_{perturbation} \\ &+ O[(\Delta t)^2 + \textstyle\sum_{s=1}^{d} (\Delta t)(\Delta x_s)] \end{aligned} \tag{7}$$

Thus,

$$[1 - \sum_{s=1}^{d} r_s(\alpha_s^{n_i} + \beta_s^{n_i})] \frac{u_i^{n_i+1} - u_i^{n_i}}{\Delta t} = \sum_{s=1}^{d} \frac{\delta_s^2 u_i^{n_i}}{(\Delta x_s)^2}$$

approximates Eq. (6) with truncation error of $O[(\Delta t) + \sum_{s=1}^{d} (\Delta x_s)]$. ∎

The coefficient, K, in Eq. (6) depends on α_s and β_s and thus can become negative. It is well-known that the multi-dimensional heat equation Eq. (6) is *well-posed* for a positive coefficient, $K > 0$, and can be ill-posed for a negative coefficient. Thus, a sufficient condition for well-posedness is $1 - \sum_s r_s(\alpha_s + \beta_s) > 0$. Consequently, for the special case where $r_s = r \ \forall s$ and $r \leq \frac{1}{2d}$ we obtain, $\sum_s(\alpha_s + \beta_s) < 2d$. However, this leads to a severe restriction on α and β so the scheme is not completely asynchronous. A weaker condition for the well-posedness of Eq. (6) can be set, requiring positiveness only in some average sense.

Lemma 2 *A sufficient condition for the well-posedness of Eq. (6) for large times when*

$$K_1(t) \leq K(\vec{x}, t) \leq K_2(t) \qquad \forall \vec{x}$$

is

$$\int_0^t K_1(\xi) d\xi > 0 \qquad \text{for} \quad t \quad \text{large}$$

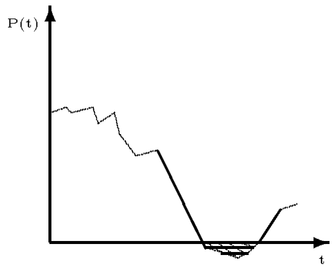

Figure 2: Well-posedness

Proof: Assume $K(\vec{x},t)$ satisfies, $K_1(t) \leq K(\vec{x},t) \leq K_2(t) \quad \forall \vec{x}$. Using separation of variables we obtain a general solution to the equation $v_t = K_1(t)\nabla \cdot [\nabla v]$:

$$v(\vec{x},t) = \sum_{n=1}^{\infty} A_n e^{-\lambda_n P(t)} \varphi_n(\vec{x})$$

where, $P(t) = \int_0^t K_1(\xi)d\xi$ and $\nabla \cdot [\nabla \varphi_n] + \lambda \varphi_n = 0$. $\varphi_n(\vec{x})$ are the appropriate eigenfunctions of the steady-state equation with eigenvalue $\lambda_n > 0$. Accordingly, $v_t = K_2(t)\nabla \cdot [\nabla v]$ has a solution

$$v(\vec{x},t) = \sum_{n=1}^{\infty} B_n e^{-\lambda_n \int_0^t K_2(\xi)d\xi} \varphi_n(\vec{x})$$

For the one-dimensional case we obtain, $\lambda_n = n^2 > 0$; $\varphi_n(x) = sin(n\pi x)$ and for two dimensions , $\lambda_n = n^2 + m^2 > 0$; $\varphi_n(x,y) = \sin(n\pi x)\sin(m\pi y)$ For large times only the first mode $\varphi_1(\vec{x})$ is important. In this case the equation is reducible to an ordinary differential equation and hence

$$e^{-\lambda \int_0^t K_2(\xi)d\xi} \varphi(\vec{x}) \leq u(\vec{x},t) \leq e^{-\lambda \int_0^t K_1(\xi)dt\xi} \varphi(\vec{x})$$

Thus, sufficient conditions for u to converge to a steady-state are:

1. $P(t) > -\delta \quad \forall t$ for some δ
2. $A_n = o(e^{-\lambda_n \delta}) \quad n \to \infty$
3. $\lim_{t\to\infty} P(t) = \infty$

A possible way of achieving convergence is restricting $K > 0$ at the beginning. This ensures that $P(t)$ is initially positive (see Fig. 2). If cond. 2 is not true and $P(t)$ is negative for some t, then

$$\nabla \cdot [\nabla v] = -\sum_{n=1}^{\infty} \lambda^2 A_n e^{\lambda_n P(t)} \varphi_n(\vec{x})$$

may not converge. To enforce cond. 2 we can start synchronously and only later let the processors run asynchronously. Another possibility in case $P(t)$ becomes negative, $P(t) > -\delta \quad \forall t$, is to filter the initial conditions, to damp the high frequencies

$$u(\vec{x}, 0) = f(\vec{x}) = \sum \tilde{A}_n \varphi_n(\vec{x})$$

and $\tilde{A}_n$ satisfies cond. 2, e.g $\tilde{A}_n = A_n e^{-xn}$ for $n > N$. ∎

Lemma 3 *The asynchronous finite difference scheme Eq. (5) is stable for*

$$0 < \sum_{s=1}^{d} r_s \leq \frac{1}{2}$$

Proof: Assume that the ${n_i}^{th}$ iteration of the i^{th} grid point had been completed and its next iteration is currently being performed (see Fig. 3). Since (3) is valid, then the absolute values of the coefficients of u values on the right hand side of Eq. (5) sum to 1.

$$|u_i^{n_i+1}| \leq \max\{|u_i^{n_i}|, |u_{i-1_s}^{n_i+\alpha_{i_s}^{n_i}}|, |u_{i+1_s}^{n_i+\beta_{i_s}^{n_i}}| \quad |1 \leq s \leq d\}$$

Using the last inequality recursively by backtracking the origins of each relevant grid point, we finally reach the initial values, thus

$$|u_i^{n_i+1}| \leq \max_j |u_j^0|$$

∎

By lemma 1, lemma 3, and the Lax Equivalence Theorem the scheme is convergent in the maximum norm. (We also note that this proof applies to any scheme for which all the coefficients are between 0 and 1. For parabolic type of equations, higher order schemes can be constructed with this property.) Hence,

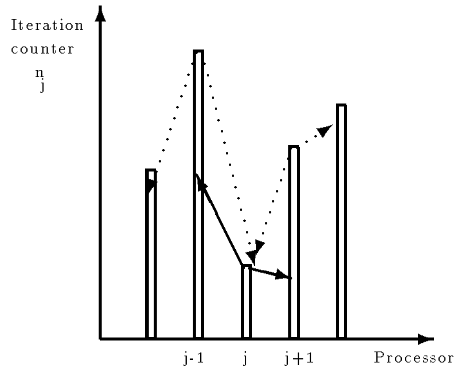

Figure 3: Stability analysis - processors status

Lemma 4 *The asynchronous finite difference scheme is convergent to Eq. (6), under the same restrictions as Lemma 3.*

This approach for proving the convergence of the AS scheme was separately derived and used by the third author in the paper [1], and it was presented in the [10].

Assume now that the solution of the asynchronous scheme in Eq. (5) had been completely evaluated up to a time level $T = N\Delta t$, and it is now interpolated into a smooth function $w(\vec{x}, t)$ over the entire region under discussion. We denote by v the discrete solution of the synchronous solution of Eq. (2) and assume that both w and v are satisfying the required boundary and initial conditions. Let ε_i^n be the difference between them at an *interior* grid point P_i for a **global** value of n;

$$\varepsilon_i^n = w_i^n - v_i^n \tag{8}$$

and let,

$$\vec{\varepsilon}^{(n)} = \begin{pmatrix} \varepsilon_1^n \\ \vdots \\ \varepsilon_I^n \end{pmatrix}$$

be the corresponding difference vector, in some specific order, associated with all interior spatial grid points P_i $(1 \leq i \leq I)$ at the same **global** time level

$t = n\Delta t$ $(n < N)$. In addition, let

$$\gamma_i^n = \frac{\sum_{s=1}^d r_s}{1 - \sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)} \tag{9}$$

where $\alpha_{i_s}^n$ and $\beta_{i_s}^n$ are the corresponding delays or hastes $d(k,i,n)$ in Eq. (4) of the neighbors of the i^{th} grid point along the s-coordinate axis, while w_i^{n+1} had been evaluated. Then we can prove the following lemma which gives bound on ε_i^n, given a specific grid spacing.

Lemma 5 *If α and β are bounded by M and*

$$0 < \gamma_i^n \leq \frac{1}{2} \quad \forall i, n \tag{10}$$

then for $n\Delta t \leq T$

$$||\vec{\varepsilon}^{(n)}||_\infty \leq M \sum_{k=0}^{n-1} \max_i |v_i^{k+1} - v_i^k| + T \cdot O[(\Delta t) + \sum_s (\Delta x_s)] \tag{11}$$

Remark 1 *The last inequality should be read as follows: there exists a constant R (which is, in fact, the bound on the second-order derivative of $w(\vec{x}, t)$) such that*

$$||\vec{\varepsilon}^{(n+1)}|| \leq M \sum_{k=0}^{n} \max_i |v_i^{k+1} - v_i^k| + R \cdot T[(\Delta t) + \sum_s (\Delta x_s)]$$

for all sufficiently small Δt and Δx_s $(1 \leq s \leq d)$. Note that R depends on $w(\vec{x}, t)$, which in turn depends on the mesh spacing.

Proof: Using Taylor Theorem of the Mean and neglecting terms of $O[(\Delta t)^2 + \sum_s (\Delta x_s)(\Delta t)]$ we obtain from Eqs. (5) and (8),

$$\begin{aligned} [1 - \textstyle\sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)]\varepsilon_i^{n+1} &= \textstyle\sum_{s=1}^d r_s(\varepsilon_{i-1_s}^n + \varepsilon_{i+1_s}^n) + \\ &+ [1 - 2\textstyle\sum_{s=1}^d r_s - \sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)]\varepsilon_i^n \\ &+ [\textstyle\sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)](v_i^{n+1} - v_i^n) \end{aligned} \tag{12}$$

in which $\varepsilon_{i-1_s}^n$ and $\varepsilon_{i+1_s}^n$ denote the difference values at the neighbors of the i^{th} grid point along the s-coordinate axis. In a matrix form one obtains,

$$\vec{\varepsilon}^{(n+1)} = A^{(n)} \vec{\varepsilon}^{(n)} + \vec{g}^{(n)} \tag{13}$$

In Eq. (13) $A^{(n)}$ is a $(2d+1)$-diagonal matrix with main diagonal terms of the form

$$\frac{1-2\sum_{s=1}^{d} r_s - \sum_{s=1}^{d} r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}{1-\sum_{s=1}^{d} r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}$$

and d additional diagonals on each side with terms of the form

$$\frac{r_s}{1-\sum_{s=1}^{d} r_s(\alpha_{i_s}^n + \beta_{i_s}^n)} \qquad 1 \leq s \leq d$$

while $\vec{g}^{(n)}$ is a vector whose components are given by

$$g_i^n = \frac{\sum_{s=1}^{d} r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}{1-\sum_{s=1}^{d} r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}(v_i^{n+1} - v_i^n) \tag{14}$$

Following is an example of a 9×9 matrix, corresponding in some specific order to a 3×3 interior grid, for the two dimensional problem:

$$\begin{pmatrix}
\frac{1-\xi_1}{1-\nu_1} & \frac{r_2}{1-\nu_1} & & \frac{r_1}{1-\nu_1} & & & & & \\
\frac{r_2}{1-\nu_2} & \frac{1-\xi_2}{1-\nu_2} & \frac{r_2}{1-\nu_2} & & \frac{r_1}{1-\nu_2} & & & & \\
 & \frac{r_2}{1-\nu_3} & \frac{1-\xi_3}{1-\nu_3} & & & \frac{r_1}{1-\nu_3} & & & \\
\frac{r_1}{1-\nu_4} & & & \frac{1-\xi_4}{1-\nu_4} & \frac{r_2}{1-\nu_4} & & \frac{r_1}{1-\nu_4} & & \\
 & \frac{r_1}{1-\nu_5} & & \frac{r_2}{1-\nu_5} & \frac{1-\xi_5}{1-\nu_5} & \frac{r_2}{1-\nu_5} & & \frac{r_1}{1-\nu_5} & \\
 & & \frac{r_1}{1-\nu_6} & & \frac{r_2}{1-\nu_6} & \frac{1-\xi_6}{1-\nu_6} & & & \frac{r_1}{1-\nu_6} \\
 & & & \frac{r_1}{1-\nu_7} & & & \frac{1-\xi_7}{1-\nu_7} & \frac{r_2}{1-\nu_7} & \\
 & & & & \frac{r_1}{1-\nu_8} & & \frac{r_2}{1-\nu_8} & \frac{1-\xi_8}{1-\nu_8} & \frac{r_2}{1-\nu_8} \\
 & & & & & \frac{r_1}{1-\nu_9} & & \frac{r_2}{1-\nu_9} & \frac{1-\xi_9}{1-\nu_9}
\end{pmatrix}$$

in which $\nu_i = r_1(\alpha_{i_1}^n + \beta_{i_1}^n) + r_2(\alpha_{i_2}^n + \beta_{i_2}^n)$, $\xi_i = 2(r_1+r_2)+\nu_i$, and

$$\vec{\varepsilon}^{(n)} = \begin{pmatrix} \varepsilon_1^n \\ \varepsilon_2^n \\ \varepsilon_3^n \\ \varepsilon_4^n \\ \varepsilon_5^n \\ \vdots \\ \varepsilon_8^n \\ \varepsilon_9^n \end{pmatrix} = \begin{pmatrix} \varepsilon_{11}^n \\ \varepsilon_{12}^n \\ \varepsilon_{13}^n \\ \varepsilon_{21}^n \\ \varepsilon_{22}^n \\ \vdots \\ \varepsilon_{32}^n \\ \varepsilon_{33}^n \end{pmatrix}$$

where ε_{ij}^n is the error of the grid point at the i^{th} row and the j^{th} column at the n^{th} time level.

It follows from Eq. (10) that all the elements of $A^{(n)}$ are positive and that $\|A^{(n)}\|_\infty = 1$. Since $\vec{\varepsilon}^{(0)} = \vec{0}$, we obtain from Eq. (13) that

$$\|\vec{\varepsilon}^{(n+1)}\|_\infty \leq \sum_{k=0}^{n} \|\vec{g}^{(k)}\|_\infty \tag{15}$$

In addition, $\frac{1}{1-\sum_{s=1}^d r_s(\alpha_{i_s}^n+\beta_{i_s}^n)} = \frac{\gamma_i^n}{\sum_{s=1}^d r_s}$ so that,

$$|\frac{\sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}{1-\sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)}| \leq \frac{\gamma_i^n}{\sum_{s=1}^d r_s}|\sum_{s=1}^d r_s(\alpha_{i_s}^n + \beta_{i_s}^n)| \leq 2M\gamma_i^n$$

Then (11) follows from the last inequality, from Eq. (10) and from Eq. (15). ∎

Note from Eq. (12) that in the steady-state case when $v_i^{n+1} - v_i^n \to 0$, $\|\varepsilon^{(n)}\|_\infty$ does not increase as $n \to \infty$, neglecting terms of $O[(\Delta t)^2 + \sum_s(\Delta x_s)(\Delta t)]$.

4 Corrected-Asynchronous (CA) Scheme

We have shown that the asynchronous iterations are consistent with a PDE that is different than the original one. Consequently, time accuracy is lost. This suggests that if we apply a correction during each iteration we can improve our approximation, without requiring explicit synchronization. For each processor we now require an extra variable that will serve as an iteration counter for that processor. The correction we apply is an extrapolation based on those variables.

Hence, we construct a modified equation whose asynchronous approximation is time consistent with the original Eq. (1) by subtracting the perturbation term of Eq. (7),

$$u_i^{n_i+1} = u_i^{n_i} + \sum_{s=1}^d r_s\hat{\delta}_s^2 u_i^{n_i} - (u_i^{n_i+1} - u_i^{n_i})\sum_{s=1}^d r_s(\alpha_{i_s}^{n_i} + \beta_{i_s}^{n_i})$$

where, $\hat{\delta}_s^2$ denotes a central difference in the space coordinates, with each point in its own completed time level. Thus, our Corrected-Asynchronous

(CA) scheme is:

$$u_i^{n_i+1} = u_i^{n_i} + \frac{1}{1+\sum_{s=1}^{d} r_s(\alpha_{i_s}^{n_i} + \beta_{i_s}^{n_i})} \sum_{s=1}^{d} r_s \hat{\delta}_s^2 u_i^{n_i} \tag{16}$$

provided that,

$$1+\sum_{s=1}^{d} r_s(\alpha_{i_s}^{n_i} + \beta_{i_s}^{n_i}) \neq 0 \tag{17}$$

Lemma 6 *If $\alpha_{i_s}^{n_i}$ and $\beta_{i_s}^{n_i}$ are bounded $\forall n, i, s$ then the Corrected-Asynchronous approximation Eq. (16) is consistent with Eq. (1).*

Proof: Using Taylor Theorem of the Mean we have,

$$\hat{\delta}_s^2 u_i^{n_i+1} = \delta_s^2 u_i^{n_i} + (\alpha_{i_s}^{n_i} + \beta_{i_s}^{n_i})(u_i^{n_i+1} - u_i^{n_i}) + O(\Delta x_s)(\Delta t) + O(\Delta t^2)$$

Substituting in Eq. (16) we obtain,

$$u_i^{n_i+1} = u_i^{n_i} + \sum_{s=1}^{d} r_s \delta_s^2 u_i^{n_i} + \sum_{s=1}^{d} O(\Delta x_s) O(\Delta t) + O(\Delta t^2)$$

or,

$$\frac{u_i^{n_i+1} - u_i^{n_i}}{\Delta t} = \sum_{s=1}^{d} \frac{\delta_s^2 u_i^{n_i}}{(\Delta x_s)^2} + \sum_{s=1}^{d} O(\Delta x_s) + O(\Delta t)$$

Hence the scheme of Eq. (16) is consistent with Eq. (1). ∎

Lemma 7 *If*

$$0 < \frac{\sum_s r_s}{1+\sum_s r_s(\alpha_{i_s}^{n_i} + \beta_{i_s}^{n_i})} \leq \frac{1}{2} \tag{18}$$

then the CA scheme of Eq. (16) is stable.

Proof: Analogous to the proof of lemma 3. ∎

By the Lax Equivalence Theorem, lemma 6 and lemma 7 imply that the CA scheme is convergent, provided that condition (18) is valid.

Condition (18) can be interpreted either as restrictions on α and β, or as a bound on r_s. Practically, the bounds on α and β are determined by various factors as discussed earlier. Consequently, they impose the above restriction on r_s.

Finally, similar to lemma 5, we now show that at a given point of a specific mesh spacing, the difference between the solutions of Eq. (16) and Eq. (2) is bounded by $O[(\Delta t) + \sum_s (\Delta x_s)]$, provided that condition (3) is valid.

Lemma 8 *Let*

$$\delta_i^n = \tilde{w}_i^n - v_i^n$$

where $\tilde{w}_i^n$ *is the smooth interpolation of the CA scheme, Eq. (16), solution and* v_i^n *is the solution of the synchronous scheme Eq. (2) both with the same initial/boundary conditions. In addition, let*

$$\vec{\delta}^{(n)} = \begin{pmatrix} \delta_1^n \\ \vdots \\ \delta_I^n \end{pmatrix}$$

be the corresponding difference vector, in some specific order, with all interior spatial grid points P_i $(1 \leq i \leq I)$ *at the same* **global** *time* $t = n\Delta t$ $(n < N)$. *If (3) is valid then for* $n\Delta t \leq T$

$$\|\vec{\delta}^{(n)}\|_\infty \leq T \cdot O[(\Delta t) + \sum_s (\Delta x_s)] \tag{19}$$

provided that (17) holds.

Proof: Along the same lines of the proof of lemma 5 we can show from Eq. (7) that $\tilde{w}_i^n$ satisfies Eq. (2) with an error of $O[(\Delta t)^2 + \sum_s (\Delta x_s)(\Delta t)]$. Hence, if (3) is valid then

$$\|\vec{\delta}^{(n+1)}\|_\infty \leq \|\vec{\delta}^{(n)}\|_\infty + O[(\Delta t)^2 + \sum_s (\Delta x_s)(\Delta t)]$$

Since $\|\vec{\delta}^{(0)}\|_\infty = 0$, by using the inequality recursively, we obtain (19).
∎

5 Numerical Results

We have implemented the above algorithms on the shared-memory multi-user *Sequent Balance* [15]. The *Sequent* systems are commercial multiprocessing environment, that incorporate multiple identical processors and a single common memory. Their computing power is based on multiple general-purpose 32-bit microprocessors which are all identical. Each processor can execute both user and kernel code. All processors share a single pool of memory, in addition to their own local memory, to enhance resource sharing and communication among different processes. Processors, memory modules, and i/o controllers are all plugged into a high-speed bus. Thus, the *Sequent* systems fit our model of an asynchronous multi-processor with the ability to exchange data between its processors via shared memory.

The *Sequent* systems run the *DYNIX* operating system, a version of UNIX that supports a multi-user environment. Therefore the overhead of a single operation cannot be predicted. For example, task creation on another processor involves allocation of a free processor, allocation of memory and free entries in system tables, re-mapping, and finally context switching. The duration of allocations of both processors and memory is unpredictable as it is effected by numerous factors, although context switching itself takes only a few hundred machine cycles. As such, the *Sequent* is an example of a machine where no a priori assumptions can be made regarding relative speeds of processing.

We investigated a serial (SR) implementation of Eq. (2), a parallel synchronous (SY) version of Eq. (2), an asynchronous (AS) implementation Eq. (5), and corrected-asynchronous (CA) implementation Eq. (16).

In our synchronous (SY) version we delay all processors until the last one finishes the current iteration using a *barrier*. In this implementation every processor increments a counter upon finishing the current iteration. While checking the counter it can determine whether it is the last to finish. If it is, it changes a special hardware managed variable to inform the other processors that the iteration is finished. If it is not, it waits for a signal from the last processor. For other possible implementations see [8] and [9].

Note also that our task was simplified since each processor updates only his own specific values, although it may read some other values. Therefore, there is no requirement for simultaneous access to update global data. A special mechanism known as *locking* can be used in order to prevent such access in cases where the algorithm permits it.

The following parameters were considered:

- Δx, Δy - represent the discretization lengths along the x-axis and the y-axis, respectively.
- Δt - represents the discretization length along the t-axis.
- T - The time level to be reached for the given problem. We seek the solutions for $u(x,t)$ where $t \leq T = N\Delta t$.

- p - The number of processing elements used for the calculation.
- L - The number of total grid points in the considered domain.
- τ - The clock time elapsed before a specific algorithm completed the calculation of the given problem.
- e - The maximal error magnitude
- E - The maximal absolute relative error of the numerical result (relative to the analytic solution).
- *scheme* - Three types of parallel schemes were used: Eq. (2) (SY - Synchronous), Eq. (5) (AS - Asynchronous), Eq. (7) (CA - Corrected Asynchronous), and the serial (SR) version of Eq. (2).

We examined both a one dimensional problem, as well as a two dimensional problem. The one dimensional problem considered was

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

with the initial condition,

$$u(x, 0) = \sin(\pi x)$$

and the Dirichlet boundary conditions,

$$u(0, t) = u(1, t) = 0.$$

The calculated results were compared with the analytic result given by

$$u(x, t) = \sin(\pi x)e^{-\pi^2 t}.$$

For the two-dimensional problem we examined the following non-homogeneous problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + F$$

where,

$$F = 5\pi^2 \sin(\pi x) \sin(2\pi y)$$

with the initial condition,

$$u(x, y, 0) = 0$$

and the boundary conditions:

$$u(0, y, t) = 0 \quad u(1, y, t) = 0 \quad u(x, 0, t) = 0 \quad u(x, 1, t) = 0$$

The calculated results were compared with the analytic result given, for the steady-state, by

$$u_\infty = \sin(\pi x)\sin(2\pi y).$$

we also considered the homogeneous non steady-state problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

with the initial condition,

$$u(x, y, 0) = \cos(\pi x)\cos(\pi y)$$

and the following Dirichlet boundary conditions:

$$\begin{aligned} u(0, y, t) &= \cos(\pi y)e^{-2\pi^2 t} \\ u(1, y, t) &= -\cos(\pi y)e^{-2\pi^2 t} \\ u(x, 0, t) &= \cos(\pi x)e^{-2\pi^2 t} \\ u(x, 1, t) &= -\cos(\pi x)e^{-2\pi^2 t}. \end{aligned}$$

The calculated results were compared with the analytic solution given by

$$u(x, y, t) = \cos(\pi x)\cos(\pi y)e^{-2\pi^2 t}$$

Performance results are shown in Table 1 for the one dimensional problem with one grid point per processor, and in Tables 2 and 3 for the one dimensional problem with several points per processor, all for the steady-state problem. In Table 4 we present results for the two-dimensional non-homogeneous steady-state problem. Non steady-state results are shown in Table 5 for the one dimensional problem with several points per processor, and in Table 6 for the two-dimensional problem.

Table 1 presents the efficiency, which is given by:

$$\text{Efficiency} = \frac{\tau_{SR}(1, l)}{p \cdot \tau_{scheme}(p, l)}$$

T = 2 (e = 10^{-8})				T = 3 (e = 10^{-9})				T = 4 (e = 10^{-10})			
p	SY	AS	CA	p	SY	AS	CA	p	SY	AS	CA
10	0.08	0.1	0.09	10	0.11	0.15	0.13	10	0.13	0.19	0.15
11	0.1	0.12	0.11	11	0.12	0.16	0.14	11	0.14	0.2	0.16
12	0.09	0.12	0.1	12	0.11	0.18	0.15	12	0.13	0.21	0.17
13	0.09	0.12	0.11	13	0.12	0.18	0.15	13	0.15	0.22	0.18
14	0.09	0.13	0.12	14	0.12	0.19	0.16	14	0.15	0.23	0.18
15	0.1	0.14	0.12	15	0.12	0.21	0.16	15	0.15	0.26	0.19
16	0.1	0.15	0.12	16	0.12	0.2	0.16	16	0.12	0.25	0.2

Table 1: Efficiency for the steady-state one dimensional problem with one grid point per processor; $L = p, \Delta x = \frac{1}{L}, \frac{\Delta t}{(\Delta x)^2} = 0.5$

where $\tau_{SR}(1, L)$ is the run time of the serial version, solving with its single processor a problem of L grid points, and $\tau_{scheme}(p, L)$ is the run time of the above parallel schemes, solving the same problem with p processors.

We observe from Table 1 that for several processors (more than 10) the efficiency is 10% - 25 %. We emphasize, however, that much better preformance is obtained when several grid points are assigned to each processor, as shown in Table 2. In this case, the efficiency of the AS scheme reaches about 90% and more, for small number of processors, while the efficiency of the CA scheme is about 50%. It decreases slightly for the CA, and more significantly for the AS, as the number of processors increases. For large number of processors, the efficiency of CA is slightly higher than that of the SY. In most cases, as T increases, the efficiency of all the parallel algorithms slightly improves. In any case, the AS is the most efficient scheme to compute a fixed number of time steps. The synchronization penalty is indicated clearly by Table 3, which shows the run time of the AS and CA schemes relative to the SY run time. As expected, the AS scheme proves to be the fastest scheme. In most cases it is almost twice as fast as the SY scheme. The CA scheme is faster than the SY only for large number of processors, and even then, as was mentioned above, its efficiency is about the same as that of the SY. In general, as the number of processors increases, there are more independent entities to synchronize, and there is a better chance for one to delay the rest.

p	T = 2 (e = 10^{-9})			T = 3 (e = 10^{-13})			T = 4 (e = 10^{-17})		
	SY	AS	CA	SY	AS	CA	SY	AS	CA
2	0.78	0.96	0.53	0.79	0.96	0.53	0.79	0.96	0.53
3	0.76	0.95	0.52	0.77	0.96	0.52	0.77	0.96	0.52
4	0.74	0.95	0.52	0.75	0.96	0.52	0.74	0.96	0.52
6	0.66	0.91	0.51	0.68	0.93	0.51	0.66	0.93	0.51
8	0.6	0.87	0.49	0.62	0.89	0.5	0.62	0.9	0.5
12	0.49	0.77	0.49	0.5	0.8	0.49	0.5	0.82	0.51
16	0.37	0.67	0.41	0.4	0.73	0.44	0.4	0.75	0.44

Table 2: Efficiency for the steady-state one dimensional problem with $\frac{L}{p}$ points per processor; $L = 48, \Delta x = 0.02128$, $\frac{\Delta t}{(\Delta x)^2} = 0.5$

Yet, the effect of this increase is not dramatic if all processors are hardware identical and are synchronized per iteration and therefore they are executing more or less the same number of iterations. Note that, the overall system load has a direct impact on the execution of sophisticated system tasks such as synchronization services, in multi-users environments such as the *Sequent* system. Thus, we may see significant delays as the system is loaded with other tasks. This increases the run time of the SY scheme, as the number of processors increases and thereby decreases the relative run time of AS and CA, as observed in Table 3. However, these delays also cause loss of accuracy of the AS and the CA schemes.

Results for the non-homogeneous two-dimensional problem are shown in Table 4 to perform in a similar manner. In this case, we have a non-trivial steady-state. The iterations are calculated until $\sqrt{\frac{\sum(u-u_A)^2_{ij}}{L}} < 10^{-2}$. Again the AS is much faster than the SY and the CA is at best as fast as the SY.

The efficiency and the accuracy of **intermediate** (non steady-state) values are shown in Table 5. In general, for relatively small times T, the asynchronous scheme provides the least accurate results. Therefore, the asynchronous scheme should be applied **only** to steady state problems. Note that unlike synchronous iterations that steadily converge, the convergence of asynchronous iterations may not be monotone, thus affecting the stopping

p	T = 2		T = 3		T = 4	
	AS	CA	AS	CA	AS	CA
2	0.82	1.49	0.82	1.5	0.82	1.5
3	0.8	1.46	0.8	1.46	0.8	1.46
4	0.77	1.43	0.77	1.43	0.77	1.42
6	0.72	1.31	0.73	1.33	0.71	1.29
8	0.69	1.23	0.69	1.24	0.69	1.24
12	0.63	0.99	0.65	1.0	0.61	0.99
16	0.55	0.89	0.54	0.89	0.55	0.9

Table 3: Run time relative to SY for the steady-state one dimensional problem with $\frac{L}{p}$ points per processor; all parameters are as in Table 2

criteria. The CA scheme is a compromise of the two. It improves the accuracy lost in the asynchronous version, by doing some extra extrapolation calculation, at the cost of increasing the calculation time. The CA scheme is faster than the SY scheme only for large number of processors (see also Table 6), when the synchronization overhead of SY is significant. Its efficiency, however, decreases as the number of processors increases. Then, the overhead of initiating another processor becomes more significant relative to the work it actually performs. processor becomes more significant than the work it actually performs. Note that although CA is significantly more accurate than AS, it is still less accurate than the SY scheme, because of the lower order of its truncation error ($O(\Delta x_s)$ instead of $O({\Delta x_s}^2)$).

Two dimensional results are shown in Table 6. In this case, the efficiency of all schemes proves to be much better than the one dimensional case (in some cases, it is close to and even higher than one). The run time improvement for the AS and CA schemes relative to SY, is significant when the communication is significant relative to computation. With a small number of processors the synchronization penalty is small in comparison to the extra computation needed by the CA scheme (as compared to the AS and SY schemes). However, as the number of processors increases and there is less work per processor, the run time of the CA is close to that of AS.

Although the intermediate time-level results of the CA scheme are signif-

p	pts in blk	Efficiency			Rel. Run Time	
		SY	AS	CA	AS	CA
2	8×16	0.66	1.0	0.60	0.59	1.11
4	4×16	0.38	0.645	0.35	0.58	1.09
8	2×16	0.22	0.38	0.205	0.56	1.07
16	1×16	0.11	0.20	0.10	0.55	1.03

Table 4: Efficiency and run time relative to SY for the 2-dim. steady-state non-homogeneous problem; $L = 256$, $\Delta x = \Delta y = 0.0667$, $\frac{\Delta t}{(\Delta x)^2} + \frac{\Delta t}{(\Delta y)^2} = 0.5$; Calculated until $\sqrt{\frac{\sum (u_{ij} - u_A)^2}{L}} < 10^{-2}$

icantly more accurate than those of the AS, they are yet much less accurate than those of the SY scheme. This is due to the dimensional additivity of the truncation error which, as indicated earlier, is of lower order than that of the synchronous scheme. Hence, in multi-dimensional problems, one should either use a very fine grid, or else higher order schemes.

6 Summary

This paper presents asynchronous (AS) and corrected-asynchronous (CA) finite difference schemes (based on the Euler explicit scheme) for the multi-dimensional heat equation, to be implemented on parallel MIMD multiprocessors. Although we consider only the heat equation, our analysis can be easily modified and extended to other parabolic partial differential equations, and other finite difference schemes. Our schemes are analyzed and implemented on the shared-memory multi-user *Sequent* Balance machine. They are compared with the corresponding serial (SR) scheme and the parallel synchronous (SY) scheme. In general, the efficiency of the parallel schemes increases as more mesh points are assigned to each processor, as the time-level increases and as the problem dimension increases. It is proved that the AS scheme converges to the solution of a differential equation other than the original one. However, it provides accurate steady-state results and its

p	Efficiency			E			e		
	SY	AS	CA	SY	AS	CA	SY	AS	CA
2	0.78	0.96	0.5	0.00367	0.9	0.02	0.000026	0.007	0.00008
3	0.74	0.94	0.49	0.00367	0.9	0.05	0.000026	0.007	0.00023
4	0.69	0.91	0.47	0.00367	0.9	0.05	0.000026	0.007	0.00025
6	0.59	0.82	0.45	0.00367	0.9	0.05	0.000026	0.007	0.0002
8	0.5	0.74	0.42	0.00367	0.9	0.04	0.000026	0.007	0.00013
12	0.36	0.56	0.35	0.00367	0.9	0.07	0.000026	0.007	0.00018
16	0.28	0.42	0.29	0.00367	0.9	0.04	0.000026	0.007	0.00012

Table 5: Efficiency and accuracy for the one dimensional problem and with $\frac{L}{p}$ grid points per processor; $L = 48, \Delta x = 0.02128$, $\frac{\Delta t}{(\Delta x)^2} = 0.5, T = 0.5$

efficiency may reach 90% and over. AS may be almost twice as fast as the parallel synchronous (SY) scheme. It is proved that unlike the AS scheme, the CA scheme does converge to the solution of the original heat equation, under certain requirements. Its efficiency, however, is only about 50% in the best cases considered here. Nevertheless, CA offers more accurate results for the intermediate (non steady-state) time-levels. Yet, its accuracy is less than the SY scheme, especially in the multi-dimensional case, due to the loss of order in the truncation error in each spatial coordinate. Hence, for intermediate time-level results one should either use a very fine spatial net, or else

p	pts in blk	Efficiency			E			e		
		SY	AS	CA	SY	AS	CA	SY	AS	CA
2	8×16	0.95	1.05	0.51	0.07	0.99	0.17	0.0000036	0.000051	0.0000030
4	4×16	0.79	0.90	0.45	0.07	0.99	0.5	0.0000036	0.000051	0.000018
8	2×16	0.67	0.81	0.42	0.07	0.97	0.16	0.0000036	0.000049	0.0000028
16	1×16	0.45	0.60	0.37	0.07	0.99	0.17	0.0000036	0.000051	0.0000030

Table 6: Efficiency and accuracy for the 2-dim. problem; $L = 256, \Delta x = \Delta y = 0.0667$, $\frac{\Delta t}{(\Delta x)^2} + \frac{\Delta t}{(\Delta y)^2} = 0.5, T = 0.5$

use higher order schemes.

References

[1] Dganit Amitai, Amir Averbuch, Moshe Israeli, and Samuel Itzikowitz. Time-stabilizing method for the numerical solution of parabolic PDEs. in preparation.

[2] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30(3):449–456, July 1983.

[3] R. Barlow and D. Evans. Synchronous and Asynchronous Iterative Parallel Algorithms for Linear Systems. *Comput. J.*, 25:56–60, 1982.

[4] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM*, 25:226–244, 1978.

[5] D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Math. Prog.*, 27:107–120, 1983.

[6] D. Chazan and W. Miranker. Chaotic relaxations. *Journal of Linear Algebra Applications*, 2:199–222, 1969.

[7] P. Dubois and F. Briggs. Performance of Synchronized Iterative Processes in Multiprocessor Systems. *IEEE Trans. Soft. Eng.*, SE-8:419–431, 1982.

[8] Françoise André, Daniel Herman and Jean-Pierre Verjus Translated by J. Howlett. *Synchronization of Parallel Programs.* The MIT Press Cambridge, Mass., 1985.

[9] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[10] Moshe Israeli. Self-Stabilizing Asynchronous Schemes for Solving Parabolic and Hyperbolic Problems. In *Conference of the Israeli Union of Mathematics*, May 1990.

[11] D. Mitra. Asynchronous relaxations for the numerical solution of differential equations by parallel processors. *SIAM J. Sci. Stat. Comput.*, 8:43–53, 1987.

[12] James M. Ortega and Robert G. Voigt. Solution of parallel differential equations on vector and parallel computers. *SIAM Review*, 27(2):149–239, June 1985.

[13] D. Reed and M. Patrick. A Model of Asynchronous Iterative Algorithms for Solving Large Sparse Linear Systems. In *Proceedings 1984 International Conference on Parallel Computing*, pages 402–409, 1984.

[14] Daniel A. Reed, Loyce M. Adams, and Merrell L. Patrick. Stencils and Problem Partitioning: Their Influence on the Performance of Multiple Processor Systems. ICASE Report 86-24, NASA, May 1986.

[15] Sequent Systems. *The Sequent Guide to Parallel Programming*, 1987.