On Parallel Asynchronous High-Order Solutions of Parabolic PDEs

appeared in Numerical Algorithms, Vol. 12, pp. 159-192, 1996

Dganit Amitai [†]
Amir Averbuch [†]
Moshe Israeli [‡] [*]
Samuel Itzikowitz [†]

[†] Department of Computer Science
School of Mathematical Sciences
Tel Aviv University, Tel Aviv 69978, Israel
[‡] Faculty of Computer Science
Technion
Haifa 32000, Israel

Abstract

In achieving significant speed-up on parallel machines, a major obstacle is the overhead associated with synchronizing the concurrent processes. This paper presents high-order *parallel asynchronous* schemes, which are schemes that are specifically designed to minimize the associated synchronization overhead of a parallel machine in solving Parabolic PDEs. They are asynchronous in the sense that each processor is allowed to advance at its own speed. Thus, these schemes are suitable for single (or multi) user shared memory or (message passing) MIMD multiprocessors. Our approach is demonstrated for the solution of the multidimensional heat equation, of which we present a spatial second-order Parametric Asynchronous Finite-Difference (PAFD) scheme. The well-known synchronous schemes are obtained as its special cases. This is a generalization and expansion of the results in [5] and [7]. The consistency, stability and convergence of this scheme are investigated in detail. Numerical tests show that although PAFD provides the desired order of accuracy, its efficiency is inadequate when performed on each grid point.

In an alternative approach that uses domain decomposition, the problem domain is divided among the processors. Each processor computes its subdomain mostly independently, while the PAFD scheme provides the solutions at the subdomains' boundaries. We use high-order finite-difference implicit scheme within each subdomain and determine the values at subdomains' boundaries by the PAFD scheme. Moreover, in order to

[*]This research was supported by the fund for promotion of research at the Technion.

allow larger time-step, we use remote neighbors' values rather than those of the immediate neighbors. Numerical tests show that this approach provides high efficiency and in the case which uses remote neighbors' values an almost linear speedup is achieved. Schemes similar to the PAFD can be developed for other types of equations [3].

Key-Words:
Parallel asynchronous and synchronous finite difference methods, parabolic finite difference approximations with constant coefficients, synchronization overhead, time-stabilizing finite difference methods.

AMS(MOS) subject classifications: 65W05, 65N05, 65N10.

1 Introduction

Parallel architectures are *not* transparent to the programmer. In order to run *efficiently* on a parallel computer not only the code has to be altered, but also a variety of other issues need to be addressed at the algorithm level in order to minimize the overhead of global synchronization and global communication. Extensive work has been done on mapping existing methods to fit parallel architectures. Yet, most of this work assumes some correlation among the processors, achieved by synchronization mechanisms. In particular, numerical *iterative* solutions of PDEs are usually based on the assumption that the time-steps are *synchronized.* This is equivalent to assuming that the algorithm is governed by a global clock so that the start of each time-step is simultaneous for all processors (see Fig. 1 (a)). The overhead associated with such synchronization degrades the speedup expected from a parallel machine. Asynchronous algorithms have the potential of eliminating such overhead, as each processor can execute more time-steps without the need to wait for all other processors.

A pioneering work in the use of *asynchronous iterations* for iterative solutions of systems of linear equations is discussed in [10]. Asynchronous iterative methods for the solution of linear systems on *multiprocessors* are also discussed in [8], and are used for the solution of ordinary differential equations by [15]. A detailed discussion can be found in [9].

In [1], [2] and [7], asynchronous finite-difference schemes are developed for the steady-state and non steady-state multidimensional heat equation. While these schemes are based on constant time-increments, those presented in [4] and [5] are asynchronous with adaptive time-increments. In addition, they have a *time-stabilizing property*: the derivatives of their time-level with respect to the spatial coordinates are bounded as the number of time-steps increases, and they vanish as the spatial mesh-size approaches zero. Therefore, unlike the asynchronous schemes of [2], those with the time-stabilizing property provide accurate results even when there is a fixed delay among the processors, i.e when certain processors are always slower than the others. Thus, they are suitable for multiprocessors with non-identical processors or unbalanced workload, provided that the differences among their speeds/loads are not too large. However, these adaptive schemes are only of first-order accuracy.

The object of this paper is to develop higher order parallel *asynchronous* schemes, thus, generalizing the results of [1], [2], [4], [5], and [7]. In particular, we seek an explicit scheme in which the need for *global* synchronization is eliminated. Therefore, each processor can run

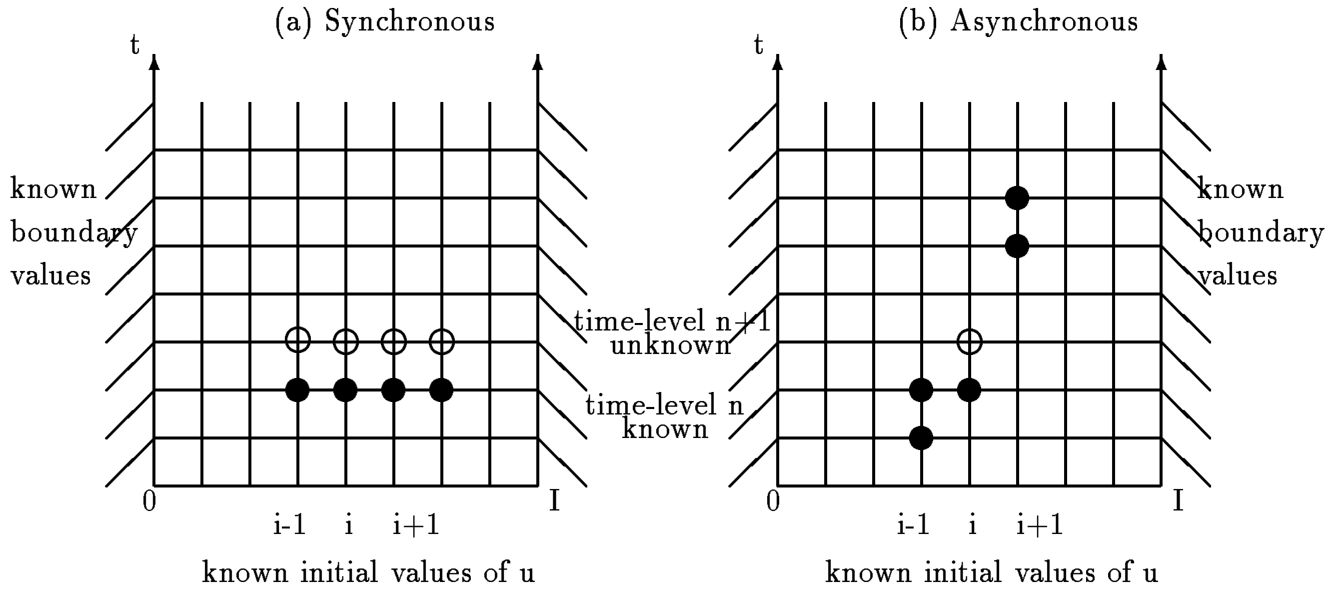


Figure 1: An asynchronous (vs. synchronous) advance on a grid

at its own speed (e.g. in a multi-user or a master/slave environments), or alternatively, no a priori assumptions are made on the processors' load (e.g. allowing distribution of unequal load).

Our approach is introduced and analyzed for a simple problem, which should serve only as a model for other problems and more sophisticated schemes [3]. Specifically, we consider the multidimensional heat equation,

$$\frac{\partial u}{\partial t} - \sum_{l=1}^{d} a_l^2 \frac{\partial^2 u}{\partial {x_l}^2} = F(\vec{x}, t) \tag{1}$$

in a bounded domain Ω, where $\vec{x} = (x_1, x_2, \ldots, x_d)$, along with the appropriate boundary and initial conditions.

We derive a Parametric Asynchronous Finite-Difference (PAFD) scheme which is a second-order (in space) asynchronous scheme. In addition, we reduce the truncation error, and ease constraints on the time-step. However, as shall be shown, this scheme involves the search for intervals' intersection points which degrades its performance substantially. Moreover, being based on an explicit scheme, the time-step is restricted in comparison with implicit schemes.

As an alternative approach, based on domain decomposition strategy, we divide the domain among the processors. Each processor computes the points within its subdomain independently and determines the solution values at subdomains' boundaries by using the PAFD scheme. The explicit solution on subdomains' boundaries along with implicit solution within a subdomain enable us to take large time-steps and thereby increasing the efficiency. Moreover, our procedure is inherently asynchronous in the sense that every processor can proceed without strict dependencies on its neighbors. This combination turns out to provide much more efficient scheme, and in some cases tested, an almost linear speedup.

The paper is organized as follows: in Sec. 2 we present our PAFD scheme and examine its consistency, stability and convergence whose meaning in this asynchronous case is explained in detail. First, we present the case of calculating every grid point according to the PAFD scheme. Then, an hybrid approach is analyzed using implicit scheme and the PAFD. The performance is illustrated by numerical examples in Sec. 3. Finally, our results and conclusions are summarized in Sec. 4.

2 Parametric Asynchronous Finite Difference (PAFD) Scheme

An asynchronous approach to the numerical solution of parabolic PDEs was proposed in [1] and [2] for steady-state problems, and in [7] for non steady-state problems. These schemes were modified in [4] and [5] to accommodate variable and adaptive time-steps increments. However, these are first order approximations since their truncation error is only $O(\Delta x_l)$ $(1 \leq l \leq d)$. Here we maintain the same *local* approach as in [5] and [7] in the sense that each processor performs its own work at its own speed, without an explicit synchronization, thus eliminating the requirement for synchronized simultaneous time-step of all processors.

Our motivation is to reduce the truncation error of these schemes, obtaining asynchronous schemes of higher order of accuracy. For this purpose, we maintain a backlog of previous values. Specifically, we use two known (previous) time-levels to calculate the next time-level. This is done by using the generalized two time-level (six-points) finite difference approximation. This general d-dimensional *synchronous* scheme for the grid point $(\vec{x}_{\mathbf{i}}, t_n)$ where $\mathbf{i} = (i_1, i_2, \ldots, i_d)$ and $t_n = n\Delta t$ can be written as (see [13]):

$$\begin{aligned}\frac{u_{\mathbf{i}}^{n+1} - u_{\mathbf{i}}^{n}}{\Delta t} = \; & \sum_l \frac{a_l^2}{(\Delta x_l)^2} \Big[\theta_{1_l}(u_{\mathbf{i}+\mathbf{1}_l}^{n+1} - u_{\mathbf{i}}^{n+1}) - \theta_{2_l}(u_{\mathbf{i}}^{n+1} - u_{\mathbf{i}-\mathbf{1}_l}^{n+1}) + \\ & \theta'_{1_l}(u_{\mathbf{i}+\mathbf{1}_l}^{n} - u_{\mathbf{i}}^{n}) - \theta'_{2_l}(u_{\mathbf{i}}^{n} - u_{\mathbf{i}-\mathbf{1}_l}^{n})\Big] + F_{\mathbf{i}}^{n}\end{aligned} \tag{2}$$

In Eq. (2) $0 \leq \theta_{j_l},\ \theta'_{j_l} \leq 1$ $j = 1, 2$ are arbitrary parameters, $u_{\mathbf{i}}^n$ and $F_{\mathbf{i}}^n$ denote the values of u and F at grid point $(\vec{x}_{\mathbf{i}}, t_n)$ respectively, $u_{\mathbf{i}}^{n+1}$ is the value of u at $(\vec{x}_{\mathbf{i}}, t_{n+1})$, $\mathbf{1}_l = (0, 0, \ldots, \underbrace{1}_{l-th\ component}, \ldots, 0, 0)$ while $\mathbf{i} - \mathbf{1}_l$ and $\mathbf{i} + \mathbf{1}_l$ are the neighbors of the point $\mathbf{i}$ in the positive and the negative direction along the $l-$axis, respectively. In the *parallel* implementation of Eq. (2), we first divide the domain into subdomains and each processor is assigned the task of processing the grid points in its subdomain.

In a *synchronous case*, the $(n+1)^{th}$ time level of a grid point is calculated only after the calculation of the n^{th} time level for all other grid points had been completed. The solution values are associated with the same time level for all grid points, and can be represented by a single global counter, n. However, for the *asynchronous* approach we use the two most recent available values at the neighboring grid points, regardless of their associated time levels (see Fig 1 (b)).

Specifically, we consider a model of two arrays of data, each with twice as many components as the number of grid points. One array, denoted by $\vec{u}$ contains the two most recent solution values of the scheme as calculated by the processors for all spatial grid points. The other array, denoted by $\vec{t}$, contains the corresponding two most recent time-levels as calculated for all grid points.

A processor is allowed to change only grid points values of its own subdomain, and to read grid points values of neighboring subdomains assigned to other processors. Therefore, the arrays are *global* in the sense that access is allowed and information is shared among all processors. However, the arrays are *locally* accessed in the sense that there is no simultaneous access and each processor reads/writes values at its own rate. In the shared memory machine model we thus have two global arrays whose values must be accessed (read/write) together. For a message-passing type of architecture, relevant array-components will reside in the local memories of each processor. In any case, an iteration counter, denoted by $n_{\mathbf{i}}$, is maintained for each grid point $\mathbf{i}$. Its value is incremented each time the processor had completed a computation of that grid point.

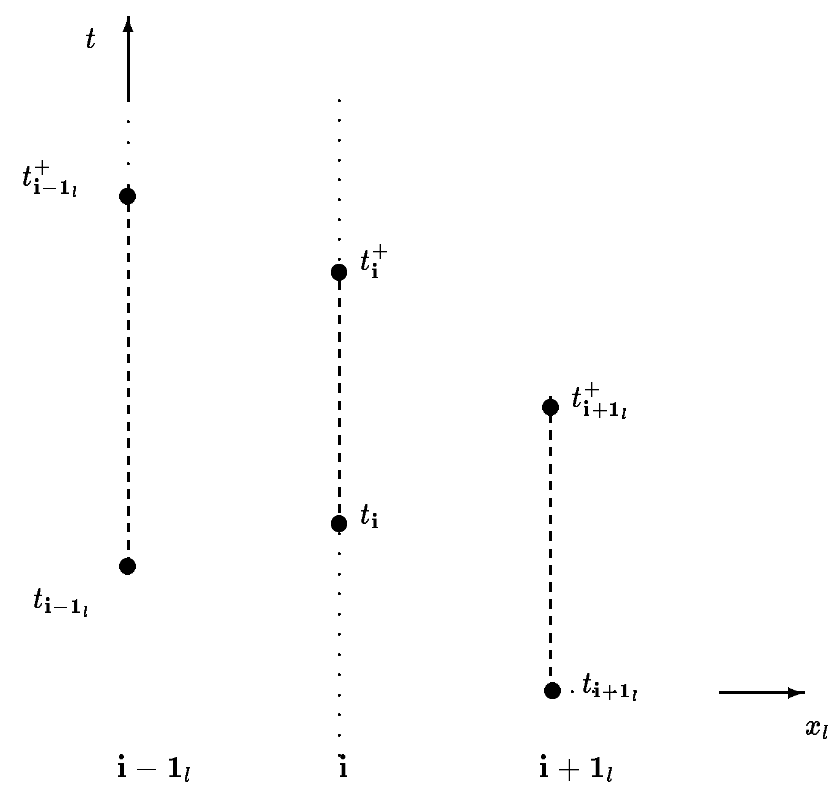


Figure 2: Six points asynchronous scheme when $t^+_{\mathbf{i}+\mathbf{1}_l} > t_{\mathbf{i}+\mathbf{1}_l}$ and $t^+_{\mathbf{i}-\mathbf{1}_l} > t_{\mathbf{i}-\mathbf{1}_l}$.

Consider now the asynchronous case and suppose that the solution at the grid point $\mathbf{i}$ for $t = t_{\mathbf{i}}$ had just been completed and the solution for this grid point is currently being evaluated for the next time-level $t = t^+_{\mathbf{i}}$. Let $t^+_{\mathbf{i}+\mathbf{1}_l}$ and $t_{\mathbf{i}+\mathbf{1}_l}$ denote the time-levels of the last and the previous available solutions at grid point $\mathbf{i}+\mathbf{1}_l$. Note that in the general case, $t_{\mathbf{i}+\mathbf{1}_l}$ is not necessarily less than $t^+_{\mathbf{i}+\mathbf{1}_l}$. Similarly, let $t^+_{\mathbf{i}-\mathbf{1}_l}$ and $t_{\mathbf{i}-\mathbf{1}_l}$ be the time-levels of the last and previous available solutions at grid point $\mathbf{i}-\mathbf{1}_l$ (see Fig. 2). For convenience we shall use the following short notations:

$$\begin{array}{ll} u_{\mathbf{i}} \triangleq u(\vec{x}_{\mathbf{i}}, t_{\mathbf{i}}) & u_{\mathbf{i}}^{+} \triangleq u(\vec{x}_{\mathbf{i}}, t_{\mathbf{i}}^{+}) \\ u_{-l} \triangleq u(\vec{x}_{\mathbf{i}-\mathbf{1}_l}, t_{\mathbf{i}-\mathbf{1}_l}) & u_{-l}^{+} \triangleq u(\vec{x}_{\mathbf{i}-\mathbf{1}_l}, t_{\mathbf{i}-\mathbf{1}_l}^{+}) \\ u_{+l} \triangleq u(\vec{x}_{\mathbf{i}+\mathbf{1}_l}, t_{\mathbf{i}+\mathbf{1}_l}) & u_{+l}^{+} \triangleq u(\vec{x}_{\mathbf{i}+\mathbf{1}_l}, t_{\mathbf{i}+\mathbf{1}_l}^{+}) \end{array}$$

With the above notations Eq. (2) to be performed asynchronously at each grid point $\mathbf{i}$, can be rewritten as

$$\begin{array}{rl} \frac{u_{\mathbf{i}}^{+}-u_{\mathbf{i}}}{t_{\mathbf{i}}^{+}-t_{\mathbf{i}}} = & \sum_l \frac{a_l^2}{(\Delta x_l)^2}\Big[\theta_{1_l}(u_{+l}^{+}-u_{\mathbf{i}}^{+}) - \theta_{2_l}(u_{\mathbf{i}}^{+}-u_{-l}^{+}) + \\ & \theta'_{1_l}(u_{+l}-u_{\mathbf{i}}) - \theta'_{2_l}(u_{\mathbf{i}}-u_{-l})\Big] + F_{\mathbf{i}} \end{array} \tag{3}$$

where $F_{\mathbf{i}} = F(\vec{x}_{\mathbf{i}}, t_{\mathbf{i}})$. Note that Eq. (3) is an explicit scheme in the sense that $u_{\mathbf{i}}^{+}$ is evaluated in terms of already *known* values at neighboring grid points. Unfortunately, however, Eq. (3) is inconsistent with the original differential equation. In order to obtain a consistent scheme, we modify Eq. (3) and present the following scheme:

$$K\frac{u_{\mathbf{i}}^{+}-u_{\mathbf{i}}}{t_{\mathbf{i}}^{+}-t_{\mathbf{i}}} = \sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}\Big[\theta_{1_l}(u_{+l}^{+}-u_{\mathbf{i}}^{+}) - \theta_{2_l}(u_{\mathbf{i}}^{+}-u_{-l}^{+}) + \theta'_{1_l}(u_{+l}-u_{\mathbf{i}}) - \theta'_{2_l}(u_{\mathbf{i}}-u_{-l})\Big] + F_{\mathbf{i}} \tag{4a}$$

in which

$$K = 1 + \sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}\Big[\theta_{1_l}(t_{+l}^{+}-t_{\mathbf{i}}^{+}) + \theta_{2_l}(t_{-l}^{+}-t_{\mathbf{i}}^{+}) + \theta'_{1_l}(t_{+l}-t_{\mathbf{i}}) + \theta'_{2_l}(t_{-l}-t_{\mathbf{i}})\Big] \tag{4b}$$

As shall be shown below, this asynchronous scheme is consistent with Eq. (1) and may provide a second-order spatial approximation, with appropriate choice of θ_{j_l},θ'_{j_l} $j = 1, 2$. Furthermore, under appropriate restrictions, this scheme is stable and "convergent" in a sense to be explained later.

2.1 Consistency

We prove the following:

Lemma 1 (**Consistency**): *The finite difference scheme of Eqs. (4a) and (4b) is consistent with Eq. (1) if :*

1. $\theta_{1_l} + \theta_{2_l} + \theta'_{1_l} + \theta'_{2_l} = 2 \quad 1 \le l \le d$
2. $\theta_{1_l} + \theta'_{1_l} = \theta_{2_l} + \theta'_{2_l} = 1 \quad 1 \le l \le d$
3. $t_{\pm l}^{+} - t_{\mathbf{i}} = O(\Delta x_l^{1+\alpha})$ *and similarly* $t_{\pm l} - t_{\mathbf{i}} = O(\Delta x_l^{1+\alpha}) = t_{\mathbf{i}}^{+} - t_{\mathbf{i}}$ *where* $\alpha > 0$.
4. K *and* $\frac{1}{K}$ *are both* $O(1)$

Proof: Let u be the solution of Eq. (1). By expanding to Taylor's series with remainder, we have the following:

$$\begin{array}{l}
\theta_{1_l}(u^+_{+l} - u^+_{\mathrm{i}}) - \theta_{2_l}(u^+_{\mathrm{i}} - u^+_{-l}) + \theta'_{1_l}(u_{+l} - u_{\mathrm{i}}) - \theta'_{2_l}(u_{\mathrm{i}} - u_{-l}) = \\
= \left[\theta_{1_l}(t^+_{+l} - t^+_{\mathrm{i}}) + \theta_{2_l}(t^+_{-l} - t^+_{\mathrm{i}}) + \theta'_{1_l}(t_{+l} - t_{\mathrm{i}}) + \theta'_{2_l}(t_{-l} - t_{\mathrm{i}})\right](u_t)^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+(\theta_{1_l} - \theta_{2_l} + \theta'_{1_l} - \theta'_{2_l})(\Delta x_l)(u_{x_l})^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+\{\frac{\theta_{1_l}}{2}\left[(t^+_{+l} - t_{\mathrm{i}})^2 - (t^+_{\mathrm{i}} - t_{\mathrm{i}})^2\right] + \frac{\theta_{2_l}}{2}\left[(t^+_{\mathrm{i}} - t_{\mathrm{i}})^2 - (t^+_{-l} - t_{\mathrm{i}})^2\right] + \\
+\left[\frac{\theta'_{1_l}}{2}(t_{+l} - t_{\mathrm{i}})^2 + \frac{\theta'_{2_l}}{2}(t_{-l} - t_{\mathrm{i}})^2\right]\}(u_{tt})^{t_{\mathrm{i}}}_{\mathrm{i}} \\
+(\Delta x_l)\left[\theta_{1_l}(t^+_{+l} - t_{\mathrm{i}}) - \theta_{2_l}(t^+_{-l} - t_{\mathrm{i}}) + \theta'_{1_l}(t_{+l} - t_{\mathrm{i}}) - \theta'_{2_l}(t_{-l} - t_{\mathrm{i}})\right](u_{x_lt})^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+\frac{(\Delta x_l)^2}{2}(\theta_{1_l} + \theta_{2_l} + \theta'_{1_l} + \theta'_{2_l})(u_{x_lx_l})^{t_{\mathrm{i}}}_{\mathrm{i}} \\
+\{\frac{\theta_{1_l}}{6}\left[(t^+_{+l} - t_{\mathrm{i}})^3 - (t^+_{\mathrm{i}} - t_{\mathrm{i}})^3\right] - \frac{\theta_{2_l}}{6}\left[(t^+_{\mathrm{i}} - t_{\mathrm{i}})^3 - (t^+_{-l} - t_{\mathrm{i}})^3\right] + \\
+\frac{\theta'_{1_l}}{6}(t_{+l} - t_{\mathrm{i}})^3 + \frac{\theta'_{2_l}}{6}(t_{-l} - t_{\mathrm{i}})^3\}(u_{3t})^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+\frac{(\Delta x_l)}{2}\left[\theta_{1_l}(t^+_{+l} - t_{\mathrm{i}})^2 - \theta_{2_l}(t_{-l} - t_{\mathrm{i}})^2 + \theta'_{1_l}(t_{+l} - t_{\mathrm{i}})^2 - \theta'_{2_l}(t_{-l} - t_{\mathrm{i}})^2\right](u_{x_ltt})^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+\frac{(\Delta x_l)^2}{2}\left[\theta_{1_l}(t^+_{+l} - t_{\mathrm{i}}) + \theta_{2_l}(t^+_{-l} - t_{\mathrm{i}}) + \theta'_{1_l}(t_{+l} - t_{\mathrm{i}}) + \theta'_{2_l}(t_{-l} - t_{\mathrm{i}})\right](u_{x_lx_lt})^{t_{\mathrm{i}}}_{\mathrm{i}} + \\
+\frac{(\Delta x_l)^3}{6}(\theta_{1_l} - \theta_{2_l} + \theta'_{1_l} - \theta'_{2_l})(u_{3x_l})^{t_{\mathrm{i}}}_{\mathrm{i}} + \sum_{k=0}^{4}\left[O(t^+_{+l} - t_{\mathrm{i}})^k + O(t^+_{-l} - t_{\mathrm{i}})^k\right](\Delta x_l)^{4-k}
\end{array}$$

In addition $\frac{u^+_{\mathrm{i}} - u_{\mathrm{i}}}{t^+_{\mathrm{i}} - t_{\mathrm{i}}} = (u_t)^{t_{\mathrm{i}}}_{\mathrm{i}} + O(t^+_{\mathrm{i}} - t_{\mathrm{i}})$. Therefore, under the above assumptions:

$$\begin{array}{l}
\frac{u^+_{\mathrm{i}} - u_{\mathrm{i}}}{t^+_{\mathrm{i}} - t_{\mathrm{i}}} - \frac{1}{K}\sum\frac{a_l^2}{(\Delta x_l)^2}\left[\theta_{1_l}(u^+_{+l} - u^+_{\mathrm{i}}) - \theta_{2_l}(u^+_{\mathrm{i}} - u^+_{-l}) + \theta'_{1_l}(u_{+l} - u_{\mathrm{i}}) - \theta'_{2_l}(u_{\mathrm{i}} - u_{-l})\right] - \frac{F_{\mathrm{i}}}{K} = \\
= \frac{1}{K}\left[(u_t)^{t_{\mathrm{i}}}_{\mathrm{i}} - \sum a_l^2(u_{x_lx_l})^{t_{\mathrm{i}}}_{\mathrm{i}} - F_{\mathrm{i}}\right] + \frac{1}{K}\sum O(\Delta x_l^\beta) + O(t^+_{\mathrm{i}} - t_{\mathrm{i}}) = \\
= \sum O(\Delta x_l^\beta)
\end{array} \tag{5}$$

where $\beta = \min(\alpha, 2)$.

Hence, the scheme of Eqs. (4a)-(4b) is consistent with Eq. (1) and the truncation error is $O\left[\max_{1\leq l\leq d}(\Delta x_l)^\beta\right]$. ∎

Thus the requirements for second-order accuracy are as follows:

Lemma 2 *If the requirements of Lemma 1 are satisfied with $\alpha = 1$ and*

$$\theta_{1_l}t^+_{+l} + \theta'_{1_l}t_{+l} = \theta_{2_l}t^+_{-l} + \theta'_{2_l}t_{-l} \triangleq t_{I_l} \tag{6}$$

for $1 \leq l \leq d$, then the truncation error of (4a)-(4b) is $O[\max_{1\leq l\leq d}(\Delta x_l)^2]$.

Proof: Follows from Taylor's expansion in the proof of Lemma 1. ∎

Note that with the second condition of Lemma 1, the geometrical interpretation of Eq. (6) is that the two intervals $[t_{+l}, t^+_{+l}]$ (or $[t^+_{+l}, t_{+l}]$ if $t^+_{+l} < t_{+l}$) and $[t_{-l}, t^+_{-l}]$ (or $[t^+_{-l}, t_{-l}]$ if $t^+_{-l} < t_{-l}$) intersect at some point t_{I_l}.

When there is an intersection interval we choose the *maximal* intersection point (see Fig. 3). Therefore, θ_{jl}, θ'_{jl} $j = 1, 2$ $1 \leq l \leq d$ are determined as follows:

1. If $t_{I_l} = t^+_{+l}$

$$\text{then} \qquad \theta_{1_l} = 1 \qquad \text{and} \qquad \theta_{2_l} = \frac{t^+_{+l} - t_{-l}}{t^+_{-l} - t_{-l}} \tag{7a}$$

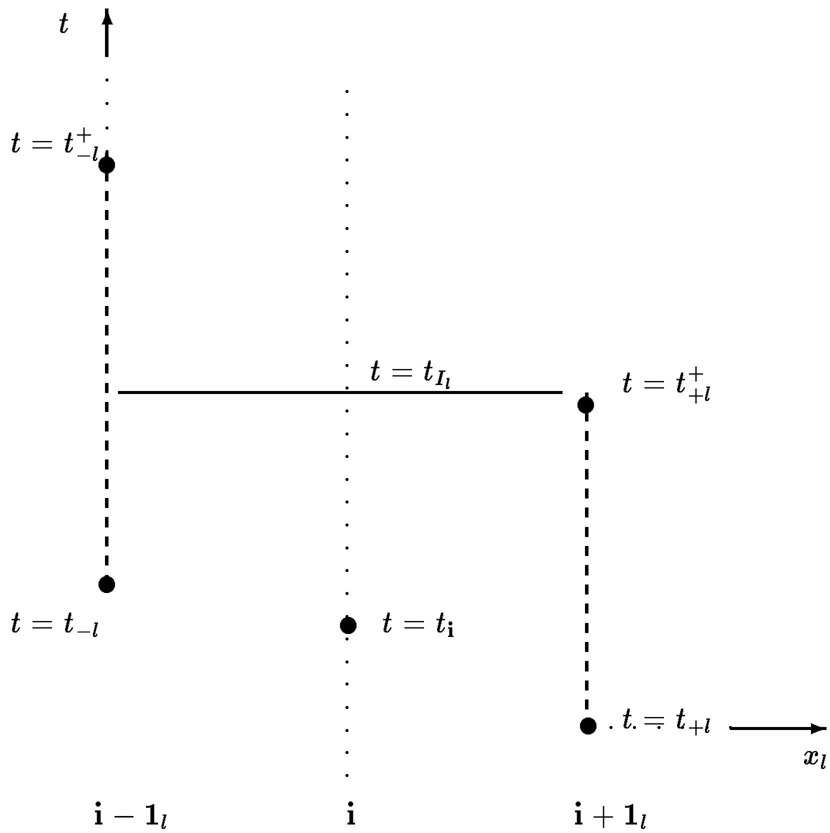


Figure 3: For evaluating the solution at $t = t_{\rm i}^+$, $t = t_{I_l}$ is taken as the highest intersection point

2. If $t_{I_l} = t^+_{-l}$

$$\text{then} \qquad \theta_{2_l} = 1 \qquad \text{and} \qquad \theta_{1_l} = \frac{t^+_{-l} - t_{+l}}{t^+_{+l} - t_{+l}} \tag{7b}$$

3. If $t_{I_l} = t_{+l}$

$$\text{then} \qquad \theta_{1_l} = 0 \qquad \text{and} \qquad \theta_{2_l} = \frac{t_{+l} - t_{-l}}{t^+_{-l} - t_{-l}} \tag{7c}$$

4. If $t_{I_l} = t_{-l}$

$$\text{then} \qquad \theta_{2_l} = 0 \qquad \text{and} \qquad \theta_{1_l} = \frac{t_{-l} - t_{+l}}{t^+_{+l} - t_{+l}} \tag{7d}$$

In all cases, $\theta'_{1_l} = 1 - \theta_{1_l}$ and $\theta'_{2_l} = 1 - \theta_{2_l}$.

2.2 Stability

We now proceed and examine the stability of our scheme of Eqs. (4a)-(4b). For this purpose, assume that some perturbations occurred in the initial values at all grid points.

The scheme (4a)-(4b) is stable if and only if the effect of these perturbations decreases as the iteration-number increases.

Lemma 3 *(**Stability**): The scheme (4a)-(4b) is stable if*

1. $\theta_{j_l},\ \theta'_{j_l} \geq 0 \ \ j = 1, 2, \ \ 1 \leq l \leq d$
2. $\theta_{1_l} + \theta_{2_l} + \theta'_{1_l} + \theta'_{2_l} = 2 \ \ \ 1 \leq l \leq d$
3. $\theta_{1_l} - \theta_{2_l} + \theta'_{1_l} - \theta'_{2_l} = 0 \ \ \ 1 \leq l \leq d$
4. $$\Delta t_{\mathbf{i}} = \frac{Kc}{\sum(\frac{a_l}{\Delta x_l})^2[2 - c(\theta_{1_l} + \theta_{2_l})]} \tag{8}$$

   in which K is given by Eq. (4b), $0 \leq c \leq 1$, and $\Delta t_{\mathbf{i}} = t^+_{\mathbf{i}} - t_{\mathbf{i}}$.

Proof: Assuming $K \neq 0$ we rewrite Eq. (4a) in the form:

$$\begin{aligned} u^+_{\mathbf{i}} \ &= \left[1 + \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 (\theta_{1_l} + \theta_{2_l})\right]^{-1} \left\{\left[1 - \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 (\theta'_{1_l} + \theta'_{2_l})\right] u_{\mathbf{i}} + \right. \\ &\left. + \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 \left[\theta_{1_l} u^+_{+l} + \theta_{2_l} u^+_{-l} + \theta'_{1_l} u_{+l} + \theta'_{2_l} u_{-l}\right]\right\} + \tfrac{\Delta t_{\mathbf{i}}}{K} F_{\mathbf{i}} \end{aligned} \tag{9}$$

Let $e = \tilde{u} - v$ where $\tilde{u}$ and v are the perturbed and exact solutions of (4a), respectively. Then

$$\begin{aligned} e^+_{\mathbf{i}} \ &= \left[1 + \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 (\theta_{1_l} + \theta_{2_l})\right]^{-1} \left\{\left[1 - \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 (\theta'_{1_l} + \theta'_{2_l})\right] e_{\mathbf{i}} \right. \\ &\left. + \tfrac{\Delta t_{\mathbf{i}}}{K} \sum \left(\tfrac{a_l}{\Delta x_l}\right)^2 \left[\theta_{1_l} e^+_{+l} + \theta_{2_l} e^+_{-l} + \theta'_{1_l} e_{+l} + \theta'_{2_l} e_{-l}\right]\right\} \end{aligned} \tag{10}$$

Under the above assumptions, the coefficients of $e_{\mathbf{i}},\ e^+_{+l},\ e^+_{-l},\ e_{+l},\ e_{-l}$ are all positive, and the sum of their absolute values is one. Therefore,

$$|e^+_{\mathbf{i}}| \leq \max_{1 \leq l \leq d} \left\{|e_{\mathbf{i}}|,\ |e^+_{\pm l}|,\ e_{\pm l}|\right\} \tag{11}$$

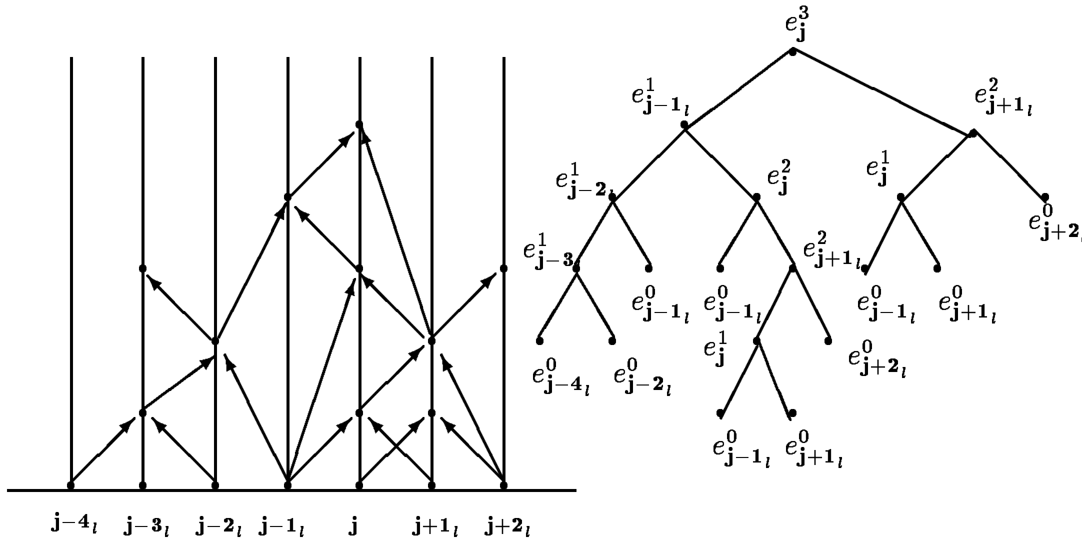
This equation is satisfied, whenever a value is updated. We may apply the last inequality recursively, each time following the "time-parents" of the grid point which maximizes $|e|$, until the initial values are reached (see Fig. 4). Then we obtain that

$$|e^+_{\mathbf{i}}| \leq \max_{\mathbf{p}} |e^0_{\mathbf{p}}| \tag{12}$$

in which $e^0_{\mathbf{p}}$ is the initial perturbation at grid point $\mathbf{p}$. Hence, the scheme (4a)-(4b) is stable.
∎

By substituting Eq. (8) into Eq. (4a), our scheme takes the following form:

$$t^+_{\mathbf{i}} = \frac{c}{2\sum(\frac{a_l}{\Delta x_l})^2} + (1-c)t_{\mathbf{i}} + \frac{c}{\sum_s(\frac{a_s}{\Delta x_s})^2} \sum (\frac{a_l}{\Delta x_l})^2 t_{I_l} \tag{13a}$$



(a) (b)

Figure 4: (a) Mapping of the values required for the computation of $u^3_{\mathbf{j}}$ (b) The associated binary tree ($c = 1$) for grid point $\mathbf{j}$ in direction l

where

$$t_{I_l} = \frac{1}{2}(\theta_{1_l} t^+_{+l} + \theta'_{1_l} t_{+l} + \theta_{2_l} t^+_{-l} + \theta'_{2_l} t_{-l}) \quad , \tag{13b}$$

and

$$u^+_{\mathbf{i}} = (1-c)u_{\mathbf{i}} + \frac{c}{\sum(\frac{a_s}{\Delta x_s})^2} \sum_l (\frac{a_l}{\Delta x_l})^2 u_{I_l} + \frac{c}{2\sum(\frac{a_l}{\Delta x_l})^2} F_{\mathbf{i}} \tag{13c}$$

where

$$u_{I_l} = \frac{1}{2}(\theta_{1_l} u^+_{+l} + \theta'_{1_l} u_{+l} + \theta_{2_l} u^+_{-l} + \theta'_{2_l} u_{-l}) \tag{13d}$$

Note that t_{I_l} and u_{I_l} represent the weighted average of neighbors' time-levels and solution values, respectively, in the $l-$axis direction. Equations (13a)-(13d) together with the requirements of Lemma 2 represent our spatial second-order **P**arametric **A**synchronous **F**inite **D**ifference (PAFD) scheme. It collapses into the synchronous case when the difference between any two time-levels on the RHS of Eq. (13b) is held constant. For example, if $\theta_{1_l} = \theta_{2_l} = 0$ ($\Rightarrow \theta'_{1_l} = \theta'_{2_l} = 1$) and $t_{+l} = t_{-l} = t_{\mathbf{i}}$ then we obtain the Euler explicit scheme. Recall that Eq. (6) is used to ensure second-order spatial accuracy in the asynchronous case, and the intersection point t_{I_l} is taken as the maximal intersection point (see Fig. 3) along the $l-$axis. Therefore, t_{I_l} is always a time-level of one of the neighbors of grid point $\mathbf{i}$ along the $l-$axis.

The PAFD scheme is an extension of the asynchronous scheme with adaptive time-steps and time-stabilizing property (TS) of [5]. That scheme can be derived from Eqs. (4a)-(4b) with $\theta_1 = \theta_2 = 0$ ($\Longrightarrow \theta'_1 = \theta'_2 = 1$). However, Eq. (6) is not fulfilled and the TS scheme is only of first-order spatial accuracy.

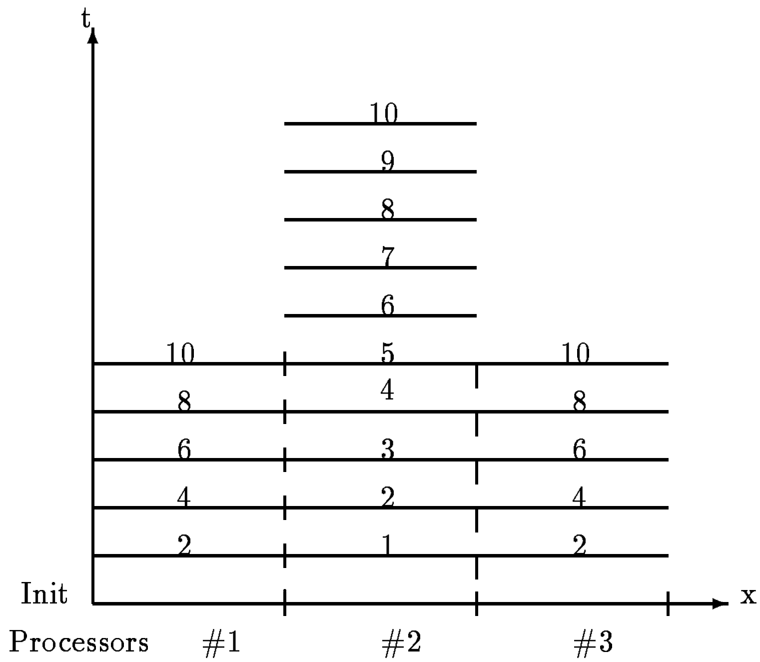


Figure 5: A profile of the processors advances, for an asynchronous scheme of constant and unrestricted time-steps. The numbers indicate "ticks" of a virtual external clock.

Following is a simple example that emphasizes the difference between constant time-step schemes and the PAFD *adaptive* time-steps. For this example, we consider a multiprocessor of three processors with the second processor running twice as fast as its neighbors. Had we run asynchronously with constant time-steps, without any restrictions on the processors' advance, we would have an ever increasing gap between the second processor and its neighbors (as illustrated in Fig. 5).

However, the PAFD scheme is based on adaptive time-steps which are calculated based on the relative positions of the neighboring grid points. These calculations implicitly reflect restrictions on the maximum allowed size for advance. In fact, the next time level of the second processor is limited by the advance of its neighbors, as is indicated by Eqs. (13a)-(13b). Therefore, in this example, assuming for simplicity that $c = 1$, we have the second processor at most "one step" ($\frac{1}{2}(\Delta x)^2$)) ahead of his neighbors. Fig. 6 is a graphical presentation of the processors' behavior, for either Algo. 1 with one point per processor or Algo. 2 with one block per processor.

Practically, our second order PAFD scheme works for grid point $\mathbf{i}$ as follows:

Algorithm 1

1. *For a boundary point compute the appropriate boundary conditions.*

2. *For every internal grid point* $\mathbf{i}$ *do:*

   (a) *Given the time-levels* $t_{\pm l}$ *and* $t^+_{\pm l}$ *along the* $l-$*axis* $(1 \leq l \leq d)$ *of the corresponding neighbors of grid point* $\mathbf{i}$*, choose* t_{I_l} *as the maximal intersection point of* $[t_{-l}, t^+_{-l}]$

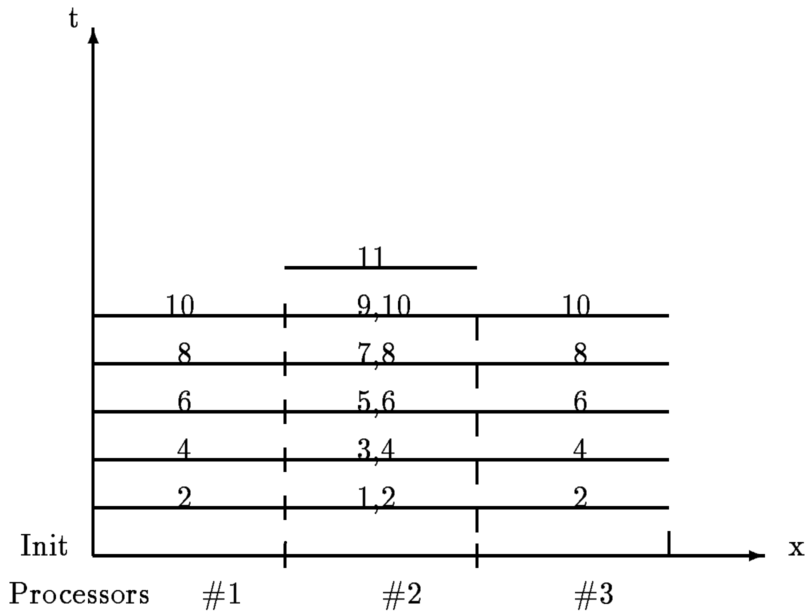


Figure 6: A profile of the processors advances, following our Algorithm 1 with $c = 1$. The numbers indicate "ticks" of a virtual external clock.

(or $[t^+_{-l}, t_{-l}]$ if $t^+_{-l} < t_{-l}$) and $[t_{+l}, t^+_{+l}]$ (or $[t^+_{+l}, t_{+l}]$ if $t^+_{+l} < t_{+l}$), if exists. If there is no intersection point, wait until one is found.

(b) Given t_{I_l} determine θ_{j_l} and θ'_{j_l} for $j = 1, 2$ and $1 \leq l \leq d$ according to Eqs. (7a) - (7d).

(c) Choose the next time-level $t^+_{\mathbf{i}}$ according to Eqs. (13a) and (13b).

(d) Evaluate $u^+_{\mathbf{i}}$, the approximated solution at $t = t^+_{\mathbf{i}}$, according to Eqs. (13c) and (13d).

3. Repeat the above steps until the required time is reached for all grid points.

Remark 1 *On a parallel computer, it is possible to divide the domain into subdomains and assign a subdomain to a processor. In this case, each grid point except the boundary points will be calculated according to the above algorithm.*

Remark 2 *Theoretically, there is a possibility that no intersection point will be found in step 2 (a) and thus the computation will be blocked. However, this has rarely happened in the tests we present in the Results section.*

2.3 Convergence

We consider now the convergence of (13a)-(13d), assuming that the requirements of Lemma 2 and Lemma 3 are satisfied. First, we wish to discuss the *meaning* of convergence in our case where the time-steps of each grid point are determined adaptively (and in fact, randomly) by Eqs. (13a) and (13b). This discussion will derive a direct proof of convergence.

In this regard, we recall that due to our specific choice of t_{I_l} (i.e maximal intersection point), it is always a time-level of some grid point. Therefore, we may apply Eq. (13a) recursively, until the time-level $t = 0$ is reached so that it can not be applied repeatedly any more. We then obtain that

$$t_{\mathbf{i}}^{n_{\mathbf{i}}} = B_{\mathbf{i}}^{n_{\mathbf{i}}} \frac{c}{2\sum(\frac{a_l}{\Delta x_l})^2} \tag{14}$$

In Eq. (14) $t_{\mathbf{i}}^{n_{\mathbf{i}}}$ is the time-level of grid point $\mathbf{i}$ in its $n_{\mathbf{i}}$-th iteration and $B_{\mathbf{i}}^{n_{\mathbf{i}}}$ is a constant which depends upon the "roots-trace" of $t_{\mathbf{i}}^{n_{\mathbf{i}}}$ as determined by the recursive implementation of Eq. (13a). The value of $B_{\mathbf{i}}^{n_{\mathbf{i}}}$ depends also upon the ratios $\frac{\Delta x_k}{\Delta x_l}$ $(1 \leq k, l \leq d)$ but it is independent of Δx_l $(1 \leq l \leq d)$ themselves. Recall that we assume the requirements of both Lemma 2 and Lemma 3 and in particular, $t_{\pm l} - t_{\mathbf{i}} = O(\Delta x_l^2)$ from which follows by Eq. (13a) that $\Delta t_{\mathbf{i}} = t_{\mathbf{i}}^{+} - t_{\mathbf{i}} = O[\max_{1 \leq l \leq d}(\Delta x_l)^2]$. Therefore, with Eqs. (13a) and (13b), the requirement $t_{\mathbf{i}}^{+} - t_{\mathbf{i}} = O(\Delta x_l)^2$ is superfluous, and if $\Delta x_l \to 0$ for $1 \leq l \leq d$ then $\Delta t_{\mathbf{i}} \to 0$.

Let $\epsilon_{\mathbf{i}} = u_{\mathbf{i}} - v_{\mathbf{i}}$ where $u_{\mathbf{i}}$ is the solution of (13a)-(13d) and $v_{\mathbf{i}}$ is the solution of Eq. (1) with the given initial/boundary conditions, at grid point $\mathbf{i}$ and time-level $t = t_{\mathbf{i}}^{n_{\mathbf{i}}}$. We say that our asynchronous scheme (13a)-(13d) is *convergent* if for any fixed $\mathbf{i}$, $\epsilon_{\mathbf{i}} \to 0$ as $\Delta x_l \to 0$ $(1 \leq l \leq d)$ such that $t_{\mathbf{i}}^{n_{\mathbf{i}}}$ of Eq. (14) is held constant. This can be achieved, for example, in the following way. For a fixed grid point $\mathbf{i}$, and for some initial specified Δx_l $(1 \leq l \leq d)$, let $T_{\mathbf{i}}^{m}$ be an increasing monotonic sequence of problem-times at which the solution is evaluated by some multiprocessor, such that $T_{\mathbf{i}}^{m} \to \infty$ as $m \to \infty$ and $T_{\mathbf{i}}^{m} > t_{\mathbf{i}}^{n_{\mathbf{i}}}$ $\forall m$, where $t_{\mathbf{i}}^{n_{\mathbf{i}}}$ is the time value that has to be held constant when $\Delta x_l \to 0$ $(1 \leq l \leq d)$. It follows from Eq. (14), when $t_{\mathbf{i}}^{n_{\mathbf{i}}}$ is replaced by $T_{\mathbf{i}}^{m}$, then $T_{\mathbf{i}}^{m}$ are increasing monotonic functions of Δx_l $(1 \leq l \leq d)$. Therefore, for each m, Δx_l can be reduced to Δx_l^m by a corresponding factor as to ensure that $\mathbf{i}$ is always a grid point and that $T_{\mathbf{i}}^{m} = t_{\mathbf{i}}^{n_{\mathbf{i}}}$ $\forall m$. We thereby obtain a decreasing sequence Δx_l^m as $m \to \infty$ $(1 \leq l \leq d)$, and a corresponding sequence $B_{\mathbf{i}_m}^{n_{\mathbf{i}}} \to \infty$ as $m \to \infty$, such that when substituted in Eq. (14), for any m, they all yield the same time-level $t = t_{\mathbf{i}}^{n_{\mathbf{i}}}$ of the same spatial point.

With this process, the first and last grid points along each spatial axis may not fall exactly on the original boundaries. When this happens, the boundaries may be shifted to the positive or negative direction, as the case may be, to match exactly the first and last grid points along the corresponding spatial axis. Then, conditions at the new boundaries may be defined, using a second-order interpolation, which as shall be shown shortly, is the local error of our scheme.

With this convergence meaning we can prove the following:

Lemma 4 (**Convergence**)*: Assume that for* $1 \leq l \leq d$ *:*

1. $\theta_{j_l},\ \theta'_{j_l} \geq 0 \ \ j = 1, 2$
2. $\theta_{1_l} + \theta_{2_l} + \theta'_{1_l} + \theta'_{2_l} = 2$
3. $\theta_{1_l} - \theta_{2_l} + \theta'_{1_l} - \theta'_{2_l} = 0$
4. $t_{\pm l}^{+} - t_{\mathbf{i}} = O(\Delta x_l^2) = t_{\pm l} - t_{\mathbf{i}}$

5. $\theta_{1_l} t^+_{+l} + \theta'_{1_l} t_{+l} = \theta_{2_l} t^+_{-l} + \theta'_{2_l} t_{-l}$

6. $\frac{1}{K}$ *and* K *are both* $O(1)$

Then our scheme (13a)-(13d) is convergent with an error of $O\left[\max_{1\leq l\leq d}(\Delta x_l)^2\right]$.

Proof: By Taylor series with remainder, under the above assumptions (see Eq. (5)) we get:

$$v^+_{\mathbf{i}} = (1-c)v_{\mathbf{i}} + \frac{c}{\sum(\frac{a_s}{\Delta x_s})^2}\sum(\frac{a_l}{\Delta x_l})^2 v_{I_l} + \frac{c}{2\sum(\frac{a_l}{\Delta x_l})^2}F_{\mathbf{i}} + \frac{c}{2\sum(\frac{a_l}{\Delta x_l})^2}\left[K(\Delta t_{\mathbf{i}}) + \sum O(\Delta x_l)^2\right] \tag{15}$$

By subtracting Eq. (15) from Eq. (13c) and applying the triangular inequality, since $K = O(1)$ we obtain:

$$|\epsilon^+_{\mathbf{i}}| \leq (1-c)|\epsilon_{\mathbf{i}}| + \frac{c}{\sum(\frac{a_s}{\Delta x_s})^2}\sum(\frac{a_l}{\Delta x_l})^2|\epsilon|_{I_l} + \frac{cM}{2\sum(\frac{a_l}{\Delta x_l})^2}\left[(\Delta t_{\mathbf{i}}) + \sum(\Delta x_l)^2\right]$$

for some constant positive number M.

As for $t^{n_{\mathbf{i}}}_{\mathbf{i}}$, since $\epsilon^0_{\mathbf{i}} = 0 \ \ \forall \mathbf{i}$, we now get by applying the last inequality recursively,

$$|\epsilon^{n_{\mathbf{i}}}_{\mathbf{i}}| \leq B^{n_{\mathbf{i}}}_{\mathbf{i}}\frac{c}{2}\frac{M}{\sum(\frac{a_l}{\Delta x_l})^2}\left[(\Delta t_{\mathbf{i}}) + \sum(\Delta x_l)^2\right] \tag{16}$$

where $B^{n_{\mathbf{i}}}_{\mathbf{i}}$ is the *same* as in Eq. (14). Since $(\Delta t_{\mathbf{i}}) = O(\Delta x^2_l)\ \ 1 \leq l \leq d$, we obtain from Eq. (14) and Eq. (16) that

$$|\epsilon^{n_{\mathbf{i}}}_{\mathbf{i}}| \leq t^{n_{\mathbf{i}}}_{\mathbf{i}} A \max_{1\leq l\leq d}\left[(\Delta x_l)^2\right] \tag{17}$$

where A is a constant which is independent of (Δx_l), $\ 1 \leq l \leq d$. The last inequality completes the proof of convergence. ∎

2.4 Hybrid Approach

The proofs of the preceding subsections address the case where every grid point is computed according to the PAFD scheme, thus allowing an asynchronous computation of each point. As shall be shown in the results section (see Table 2), this approach imposes an overhead for calculating the intersection point, t_{I_l}, and thus resulting slower performance. However, in practice, the number of processors is limited and many grid points are assigned per processor. Therefore, coarse grain computation is more useful, as within a processor the calculations for all points are synchronized. Thus, the assumption of asynchronity within a processor is wasteful, and we can assume synchronization within a processor, and asynchronization among the processors. It follows that we need an asynchronous scheme (e.g. PAFD) only on the interface points between processors.

In this case, we consider a parallel asynchronous solution where parallelism is achieved by decomposition of the problem domain into subdomains. When the spatial domain is decomposed, each processor is assigned the task of processing a subdomain. Such decomposition reduces the problem into several sub-problems that can be solved *independently* by each processor, with minimal synchronization restrictions imposed. This is done by using within each subdomain any implicit finite difference scheme. Whereas the solution values on the subdomains' boundaries are determined by using the PAFD scheme. For each processor to be able to compute asynchronously of the others, we require the computation to be based on values stored at that processor and on values at neighboring processors, with minimal restrictions imposed on their related time-levels. Thus, we use for the calculation of the subdomains' boundaries the PAFD explicit scheme which is *asynchronous* in the sense that such calculation requires communicating values of neighboring processors but without restricting their relative speed of processing. Since, an implicit scheme can be used within a subdomain, it does not impose any restrictions on the time-step in addition to those of the PAFD scheme.

Such approach is based on a *domain decomposition* (DD) strategy, except that we do not require synchronization of the processors as is usually assumed. In general Domain Decomposition techniques facilitate the numerical solution of PDEs on parallel computers. Their main idea is to divide the whole computational domain into smaller subdomains, and the original problem into subproblems to be solved independently in each subdomain. The division and the collection of the solutions at each subdomain, are done in such way to provide an accurate global solution. After decomposition the problem is solved in each subdomain independently, to a great extent, except for a synchronized time step, necessary in order to obtain the global solution. The synchronization of the time steps is crucial as it introduces overheads and may severely reduce the performance gained by parallelism.

Domain Decomposition is used for various motivations such as to simplify problems in complicated geometries. When used for parallelization, each subdomain is allocated to different processor. Obviously, the main objective is to perform as much work as possible within each subdomain to minimize communication and synchronization overhead. An implicit scheme can be utilized within each subdomain. Yet, special treatment is needed at the subdomains' boundaries to assure global solution at minimum overhead. Our approach differs of other domain decomposition implementations because minimal constraints are imposed on the processing coordination, thus, enabling an implementation on parallel machines in which some processors have persistent speed differences (e.g. multiprocessors with non-identical processors or unbalanced workload) provided that the speed/load differences are not too large.

In practice, a DD approach to the PAFD scheme is implemented as follows:

Algorithm 2

1. *Divide the domain into subdomains and assign each processor the task of processing a subdomain.*

2. *For every subdomain:*

   (a) *Determine* Δt_k *according to Eq. (7) , for* k *at subdomain's boundaries (if the subdomain boundary is a domain boundary use the given boundary conditions). For every subdomain* s, $\Delta t_s = \min\{\Delta t_k\}$, $t_k^+ = t_k + \Delta t_s$, *and* c *is determined accordingly.*

   (b) *Determine* u_k^+ *according to Eqs. (14a) - (14d) using the values determined above.*

(c) Use an implicit scheme (Eq. (2) with $\theta_1 = \theta_2 = 1$ and $\theta_1^{'} = \theta_2{}' = 0$) with Δt_s.

3. *Repeat Step 2 for every subdomain until the time-level of the subdomain is equal or higher than the required time, T.*

An analysis of the numerical properties of such a procedure is given in the following lemma:

Lemma 5 (**Hybrid**)*: Using Backward-Euler scheme inside subdomain and PAFD scheme on subdomains' boundaries (Algorithm 2) results a scheme which is stable, consistent with Eq. (1), and convergent with an error of* $O[\max_{1\le l\le d}(\Delta x_l)^2]$.

Proof: Backward-Euler (Eq. (2) with fixed values of $\theta_1 = \theta_2 = 1$ and $\theta_1^{'} = \theta_2{}' = 0$) used within subdomain is indeed the PAFD scheme of Eqs. (14a) - (14d) performed on each internal grid point, i, with the following fixed parameters values of $c = \frac{1}{2}$, $\theta_1 = \theta_2 = 1$ and $\theta_1^{'} = \theta_2{}' = 0$. On subdomain boundaries PAFD is performed with θ_l determined by Eqs. (7a) - (7d). Thus, in fact, PAFD is performed on each internal point of the domain (with appropriate values of c, θ_l, and $\theta_l^{'}$; Moreover, the requirements for Lemmas 3, 4, and 5 are held. Therefore, we conclude that by those lemmas the scheme is stable, consistent with Eq. (1), and convergent with an error of $O[\max_{1\le l\le d}(\Delta x_l)^2]$. ∎

In order to illustrate the numerical properties of such an hybrid approach we present the following computer simulations. In particular, we present a case where the PAFD is performed with different spatial discretization ($\Delta x = kh = H$) than that of the implicit scheme ($\Delta x = h$). [12] showed that, while the stability restriction is $\Delta t \le \frac{H^2}{2}$, in certain cases the order of accuracy is $O(h^2 + H^3)$.

All simulations were run on a Sun Sparc 2 uniprocessor, simulating the behavior of a multiprocessor. The problem we solve is,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad 0 \le x \le 1 \tag{18}$$

with the initial and boundary conditions:

$$u(x,0) = x^2 \qquad u(0,t) = 2t \qquad\qquad u(1,t) = 1 + 2t$$

whose analytic solution is $x^2 + 2t$. The choice of this problem was made so that the PAFD scheme whose order of accuracy is of $O(\Delta t + \Delta x^2)$ will produce no approximation errors beyond those caused by asynchronization and round-off errors.

We simulated the behavior of the following different schemes (summarized in Table 1) to be compared with our approach:

- **SER-E**: *serial* Euler explicit scheme (Forward-Euler)
- **SER-I**: *serial* Euler implicit scheme (Backward-Euler)
- **EU-I**: *parallel* scheme based on domain decomposition. Within each subdomain we use Euler implicit (Backward-Euler) finite difference scheme. The solution values on the subdomains' boundaries are determined by using the Euler explicit (Forward-Euler) scheme.

- **PAFD-I**: *parallel asynchronous* scheme based on domain decomposition. Within each subdomain we use Euler implicit (Backward-Euler) finite difference scheme. The solution values on the subdomains' boundaries are determined by using the PAFD scheme. Note, that in this case an asynchronous parallel processing is simulated.

- **EU(k)-I**: *parallel asynchronous* scheme similar to EU-I except that whereas with EU-I the same spatial discretization ($\Delta x = h$) is used within subdomain and on subdomains' boundaries, in this case a larger discretization is used on subdomains' boundaries ($\Delta x = kh$).

- **PAFD(k)-I**: *parallel asynchronous* scheme similar to PAFD-I except that while whereas with PAFD-I the same spatial discretization ($\Delta x = h$) is used within subdomain and on subdomains' boundaries, in this case a larger discretization is used on subdomains' boundaries ($\Delta x = kh$). Note, that in this case an asynchronous parallel processing is simulated.

Scheme	Parallel	Asynchronous processing	Scheme within subdomain	Scheme on subdomains' boundary
SER-E	no	no	Forward-Euler	Not Applicable
SER-I	no	no	Backward-Euler	Not Applicable
EU-I	yes	no	Backward-Euler	Forward-Euler ($\Delta x = h$)
PAFD-I	yes	yes	Backward-Euler	PAFD ($\Delta x = h$)
EU(k)-I	yes	no	Backward-Euler	Forward-Euler ($\Delta x = kh$)
PAFD(k)-I	yes	yes	Backward-Euler	PAFD ($\Delta x = kh$)

Table 1: Summary of schemes simulated

All these schemes are second-order approximations in space, and first-order approximations in time. All simulations were run for the sample problem of Eq. (18) for $T = 0.01$, unless stated otherwise. In simulating a parallel processing the problem domain was divided at $x = 0.5$, and each half was processed separately. In simulating an asynchronous parallel processing (for PAFD-I and PAFD(k)-I) one half ($0 \leq x \leq 0.5$) was processed at a rate about 10% slower than the other half.

First, we present for SER-I, EU-I, EU(3)-I, PAFD-I and PAFD(3)-I their relative error as Δx gets smaller, see Figures 7 to 11. When considering the fixed interval $0 \leq t \leq T$ with a sequence of finite difference solutions for successively finer networks, the computation is defined as unstable if as $\Delta x \to 0$ the finite difference solutions at $t = T$ can become unbounded. In our case, for all the parallel schemes simulated the relative errors were bounded by those of SER-I. We note that a combination of explicit and implicit schemes has a potential for smaller errors than those of the original schemes due to error compensation [17].

Following is a comparison of the absolute errors found when simulating parallel synchronous processing of EU-I and EU(5)-I compared with SER-E, SER-I (see Fig. 12 for

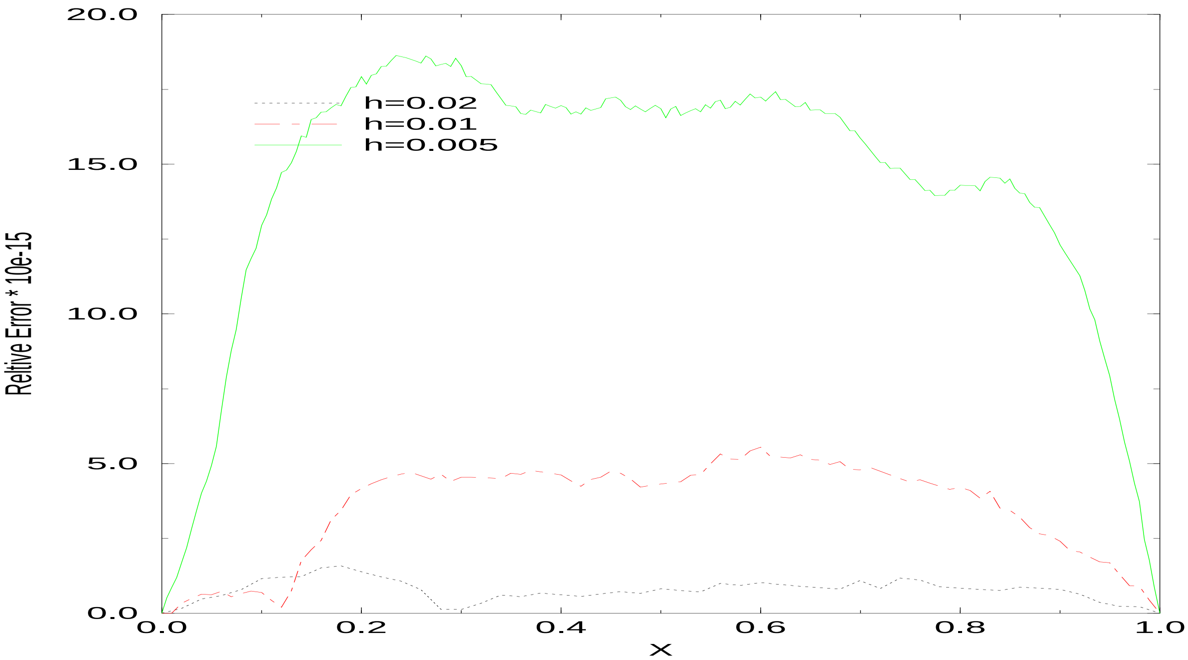


Figure 7: Relative errors found in *simulation* on Sun Sparc 2 of SER-I at $T = 0.01$, for decreasing values of Δx.

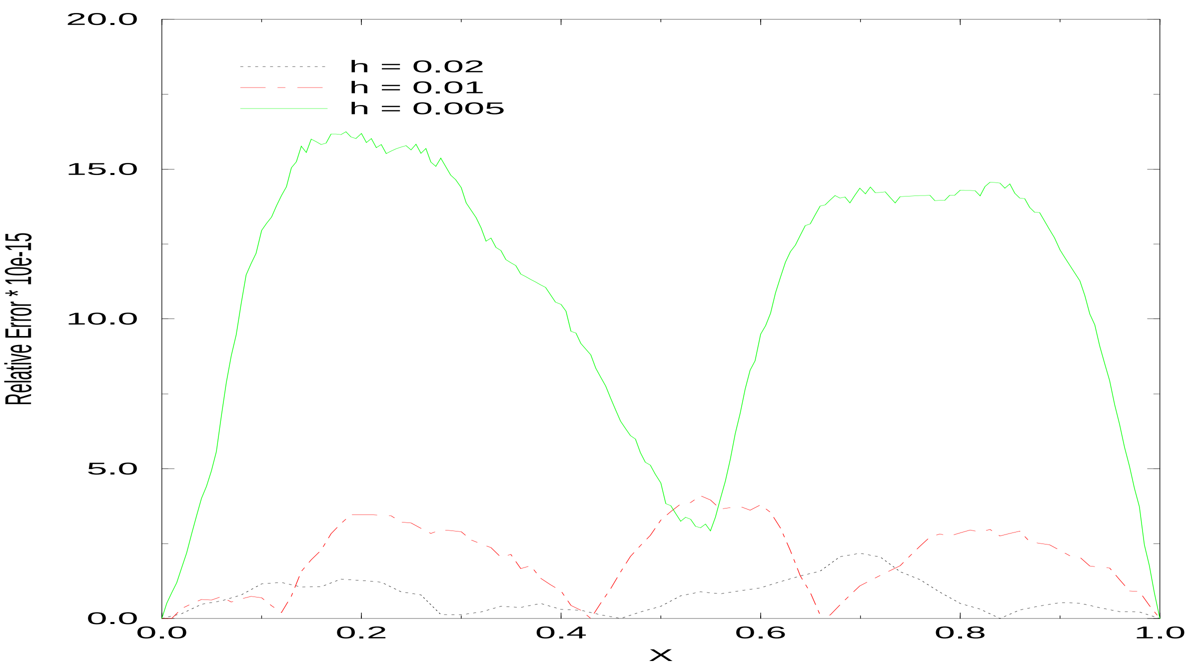


Figure 8: Relative errors found in *simulation* of a synchronous parallel processing on Sun Sparc 2 of EU-I at $T = 0.01$, for decreasing values of Δx.

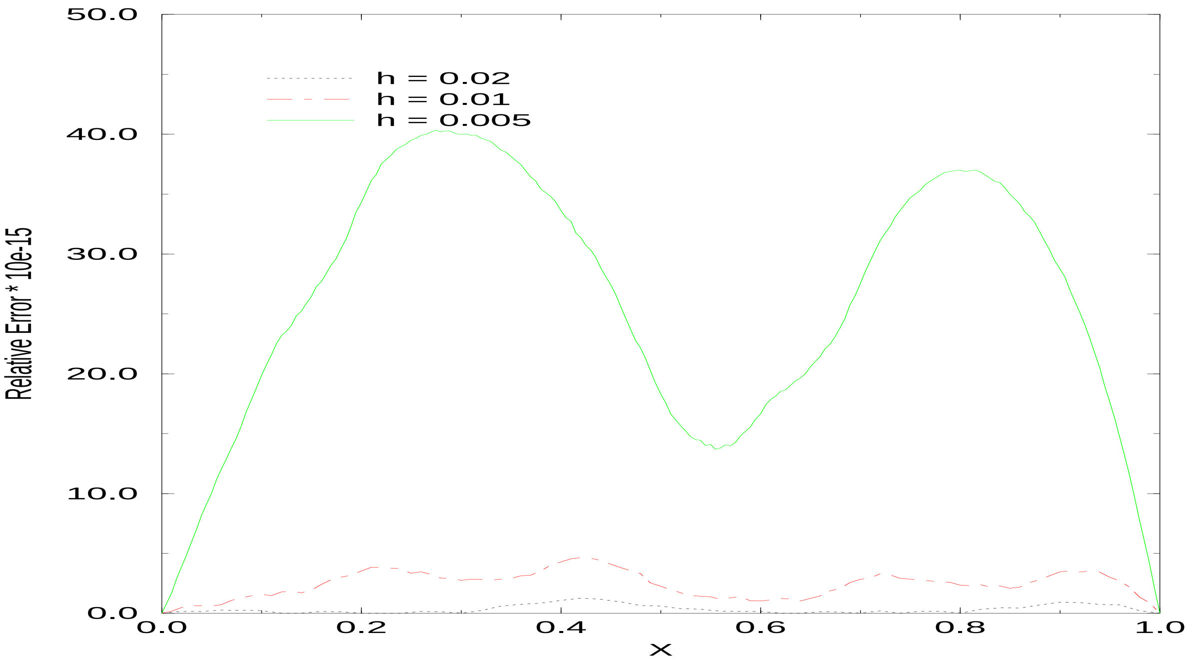


Figure 9: Relative errors found in *simulation* of a synchronous parallel processing on Sun Sparc 2 of EU(5)-I at $T = 0.01$, for decreasing values of Δx.

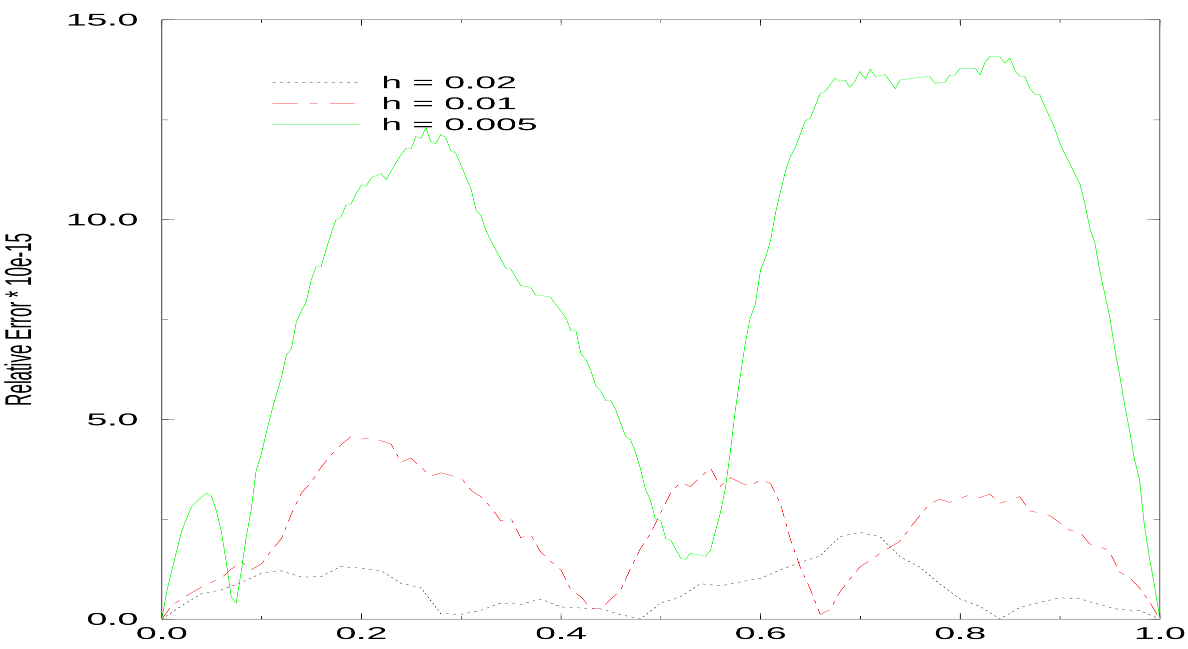


Figure 10: Relative errors found in *simulation* of an *asynchronous* parallel processing on Sun Sparc 2 of PAFD-I at $T = 0.01$, for decreasing values of Δx.

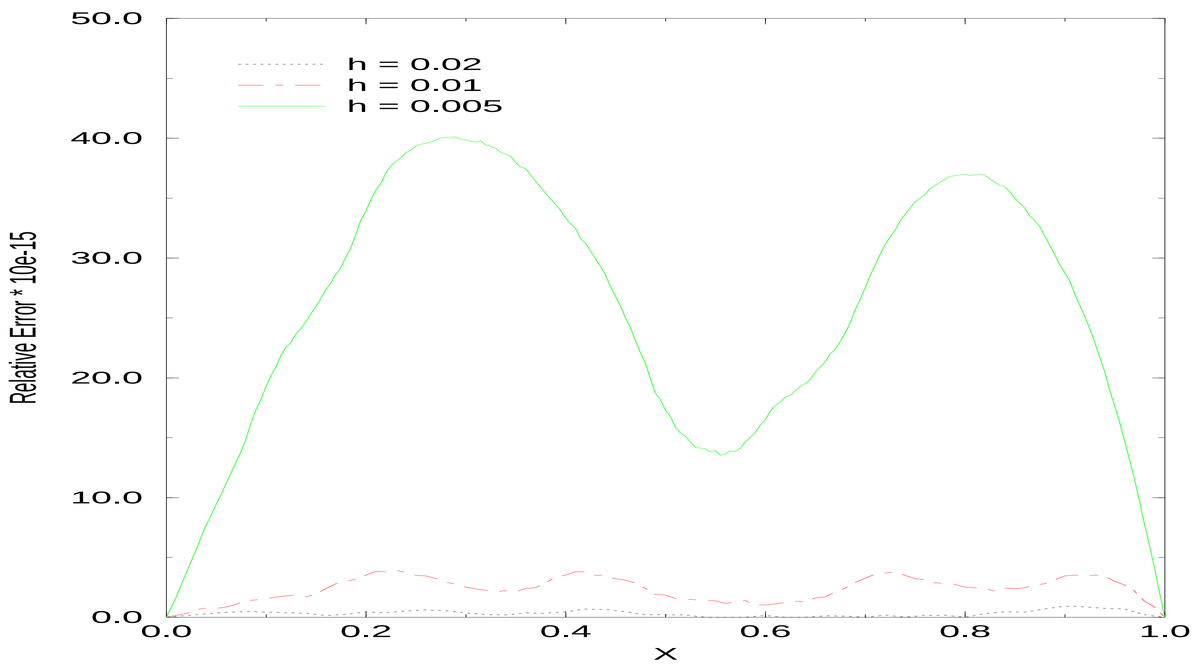


Figure 11: Relative errors found in *simulation* of an *asynchronous* parallel processing on Sun Sparc 2 of PAFD(5)-I at $T = 0.01$, for decreasing values of Δx (Δx = h = 0.02, 0.01, and 0.005) .

$T = 0.01$ and Fig. 13 for $T = 0.1$). We note that here decomposing the domain and independently processing each subdomain produced smaller errors.

Next are the results for simulating a parallel *asynchronous* processing having the right hand side updated at a rate about 40% slower than that of the left. Only PAFD-I and PAFD(k)-I can run in his environment, all other schemes are synchronous in nature and require the faster processor to wait for the slower. We present the results for PAFD-I, PAFD(2)-I and PAFD(5)-I. The profile of the absolute errors found at each grid point are shown in Fig. 14 for $T = 0.01$ and in Fig. 15 for $T = 0.1$. We note that even though the synchronization constraint is removed the errors produced are similar to those produced in a synchronous environment by EU-I and EU(k)-I.

Using a k-th remote neighbor, both in the synchronous case of EU(k)-I and the asynchronous case of PAFD(k)-I, produced the smallest errors. This can be explained by the following simulations showing a decay of a perturbation introduced to the initial conditions. The smallest errors and fastest decay are achieved $k = 5$. Greater values introduce greater errors.

To gain a better insight we have introduced a sinusoidal perturbation at initial conditions, and took snapshots of the errors found after a certain number of time-steps. As expected, a decay of the initial perturbation can be clearly seen in all simulations (Figs. 16 to 24) even in those simulating an asynchronous environment for PAFD-I and PAFD(k)-I. Yet, the decay is much faster in those schemes using $k > 1$, but up to the limit of $k = 5$, for this particular example. Also, when the difference in the updating rates of neighboring subdomains gets large (50% and more) a perturbation appears on the subdomain boundary and the scheme is unusable.

Fig. 16 shows the decay for SER-E. Fig. 17 shows the decay for SER-I. Fig. 18 shows

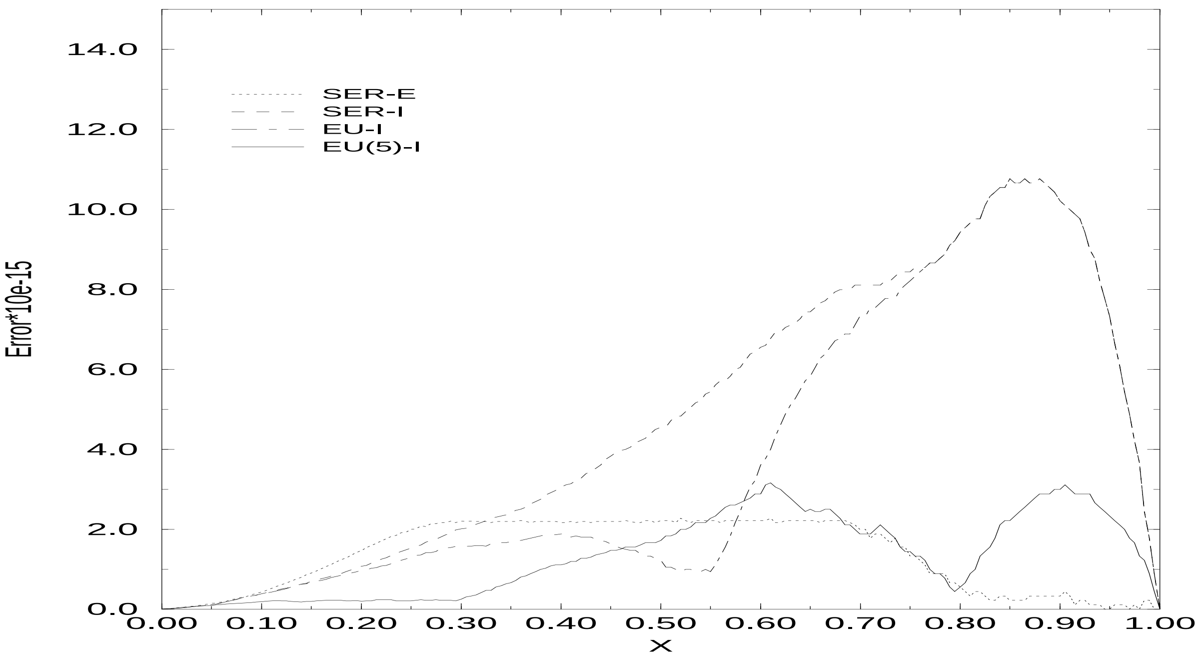


Figure 12: Error found in *simulation* on Sun Sparc 2 of two processors running the various schemes *synchronously* at $T = 0.01$.

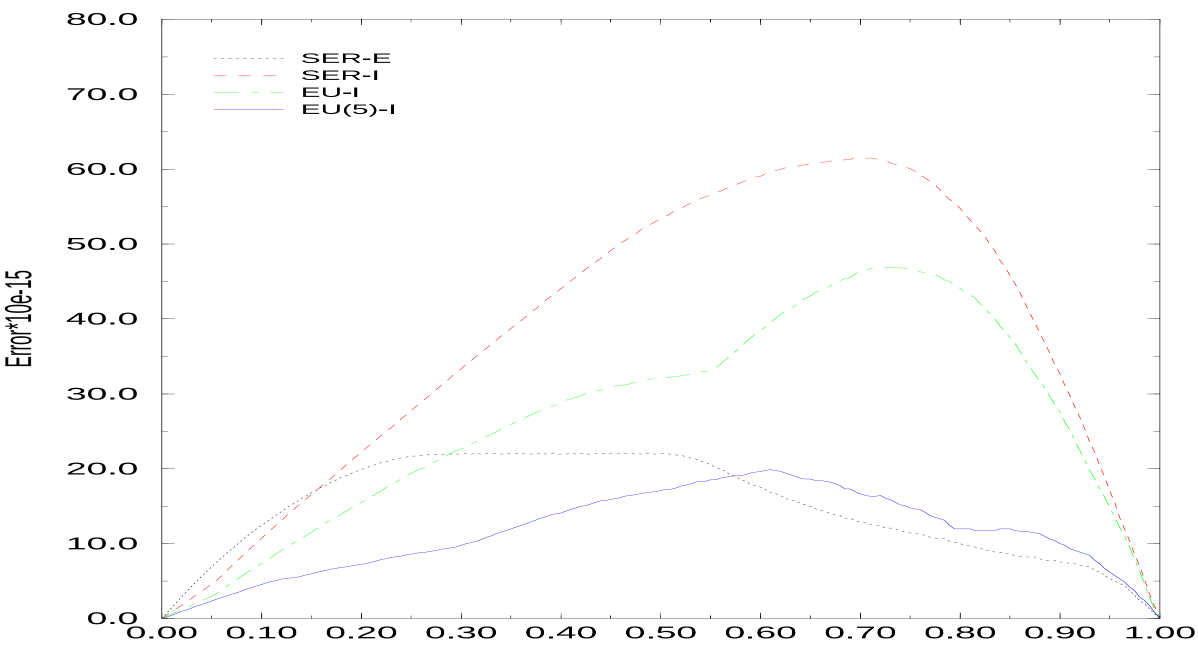


Figure 13: Error found in *simulation* on Sun Sparc 2 of two processors running the various schemes *synchronously* at $T = 0.1$.

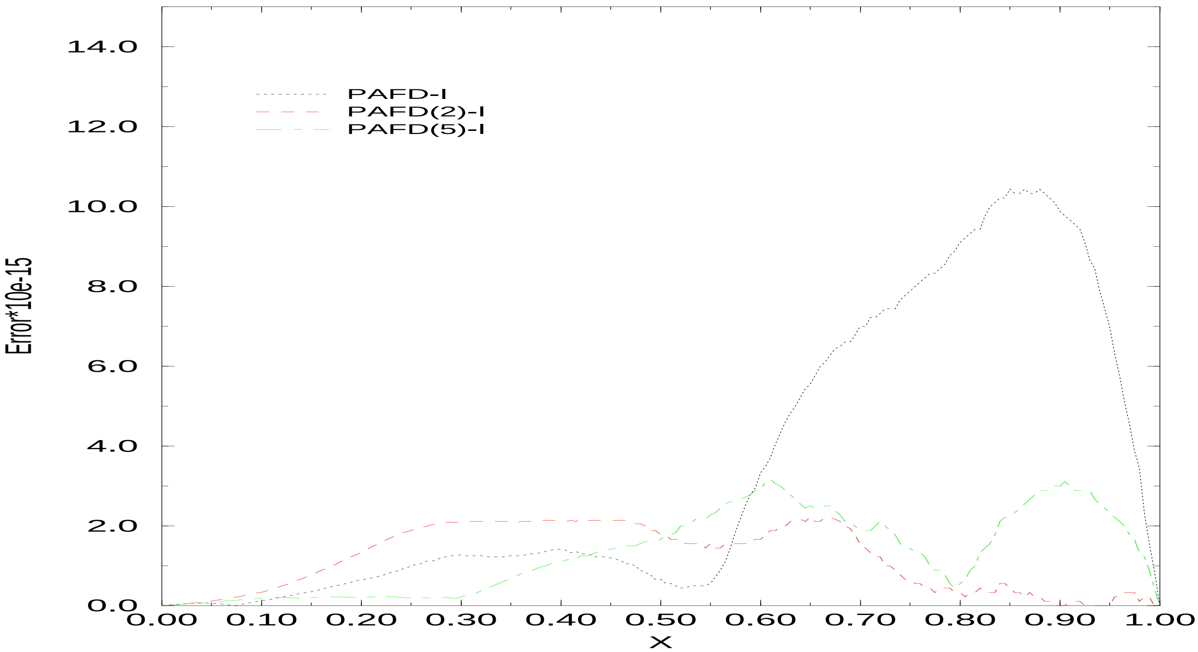


Figure 14: Error found in simulation of two processors (subdomains' boundary at $x = 0.5$) for PAFD-I running *asynchronously*, the right hand side updated at a rate about 50% slower than the left. Results at $T = 0.01$.

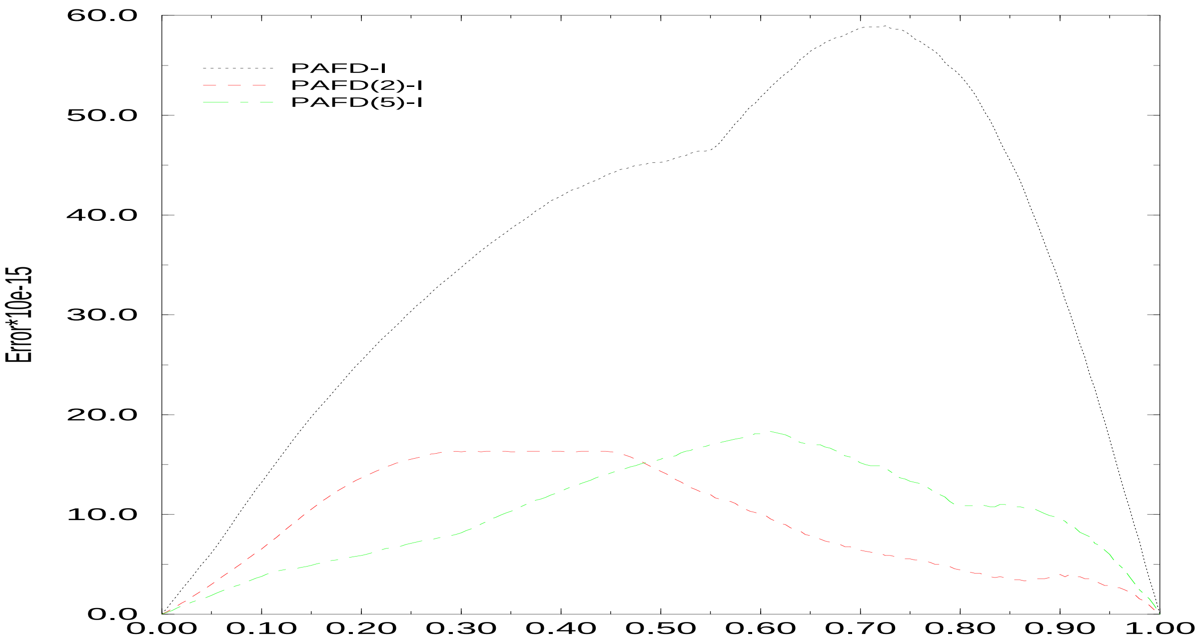


Figure 15: Error found in simulation similar to above figure but at $T = 0.1$.

the decay for EU-I diving the domain into two subdomains at $x = 0.5$. Fig. 19 shows the decay for EU(2)-I diving the domain into two subdomains at $x = 0.5$. Fig. 20 shows the decay for PAFD-I diving the domain into two subdomains at $x = 0.5$. This time we simulate an *asynchronous* environment with the right hand side updated at a rate about 10% slower than the left. Fig. 21 shows the same scheme and parameters as in Fig. 20, except that the right side is updated at a rate about 50% slower than the left. Fig. 22 shows the decay for PAFD(2)-I. The right hand side is updated at a rate about 50% slower than the left. Fig. 23 shows the decay for PAFD(5)-I. The right side is updated at a rate about 10% slower than the left. Finally, Fig. 24 shows the decay for PAFD(5)-I This time the right side is updated at a rate about 50% slower than the left.

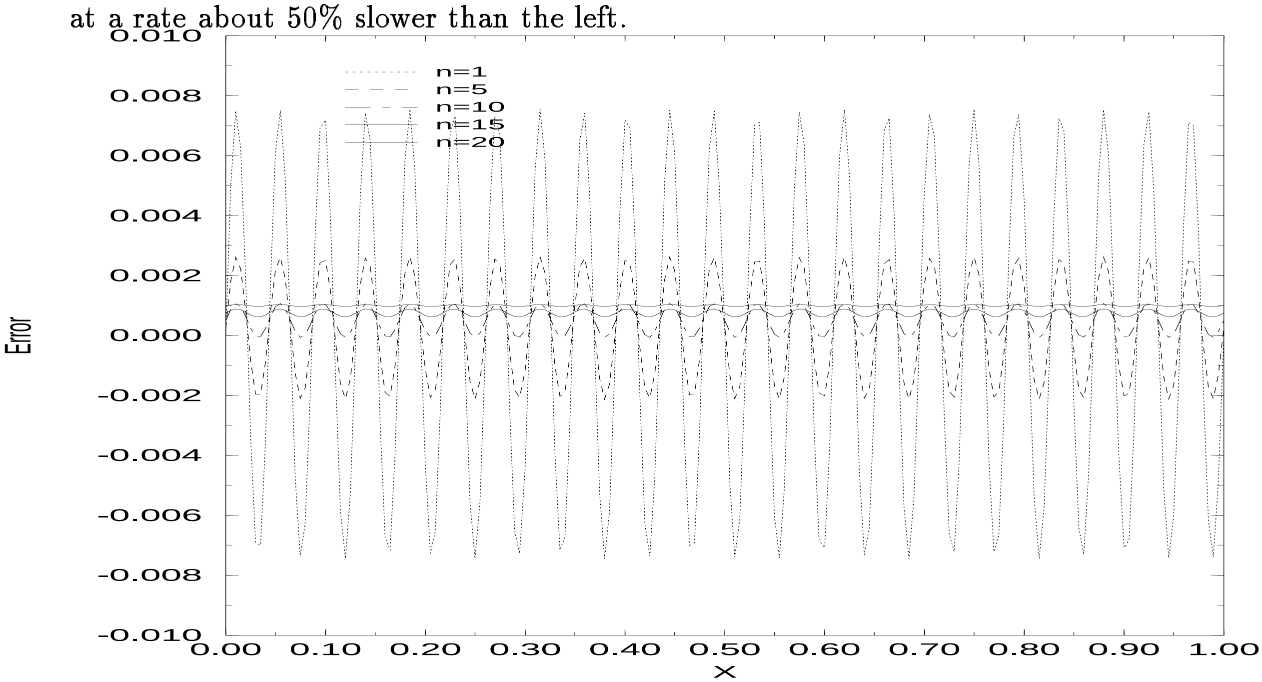


Figure 16: Error found in simulation of SER-E after introducing sinusoidal perturbation at initial conditions. The snapshot is taken after n=1,5,10,15 and 20 time-steps.

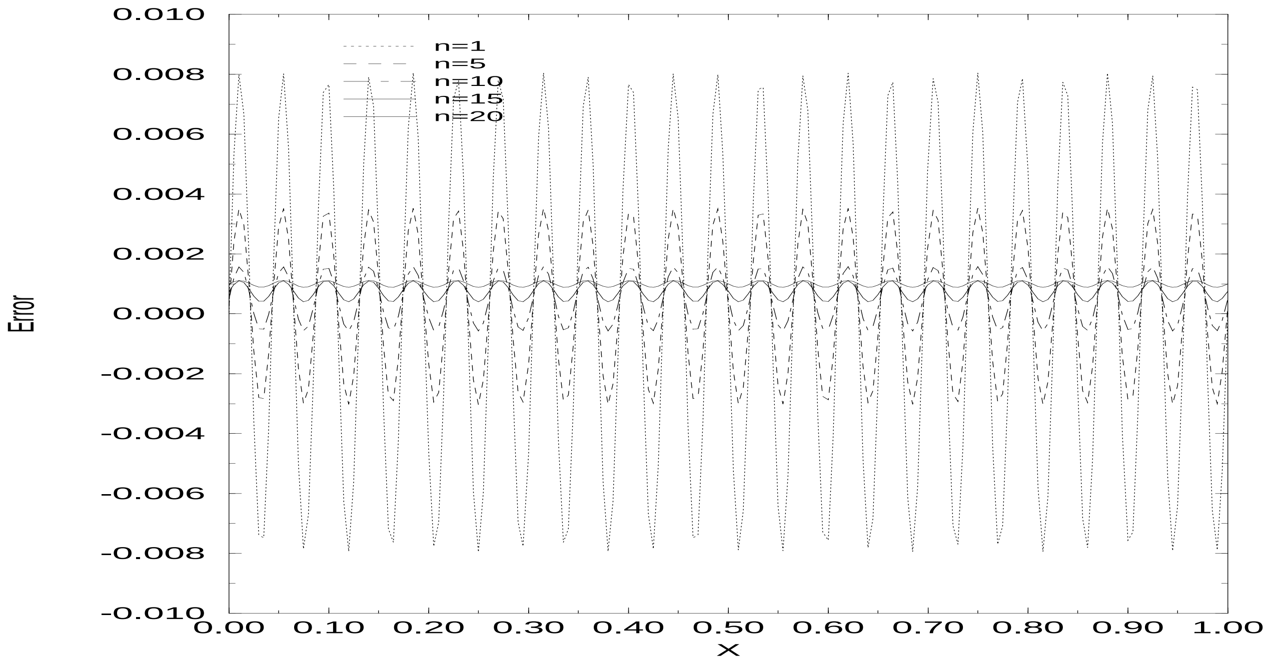


Figure 17: Error found in simulation of SER-I after introducing sinusoidal perturbation at initial conditions. The snapshot is taken after n=1,5,10,15 and 20 time-steps.

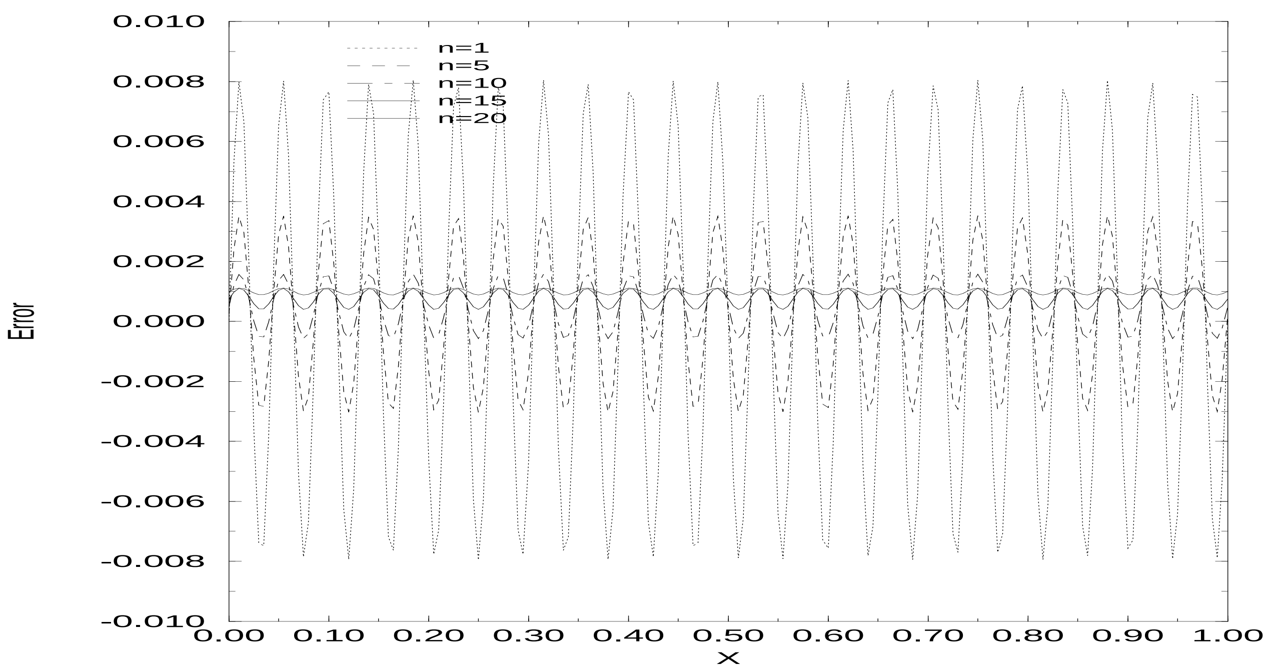


Figure 18: Error found in simulation of cay for EU-I diving the domain into two subdomains at $x = 0.5$, after introducing sinusoidal perturbation at initial conditions. The snapshot is taken after n=1,5,10,15 and 20 time-steps.

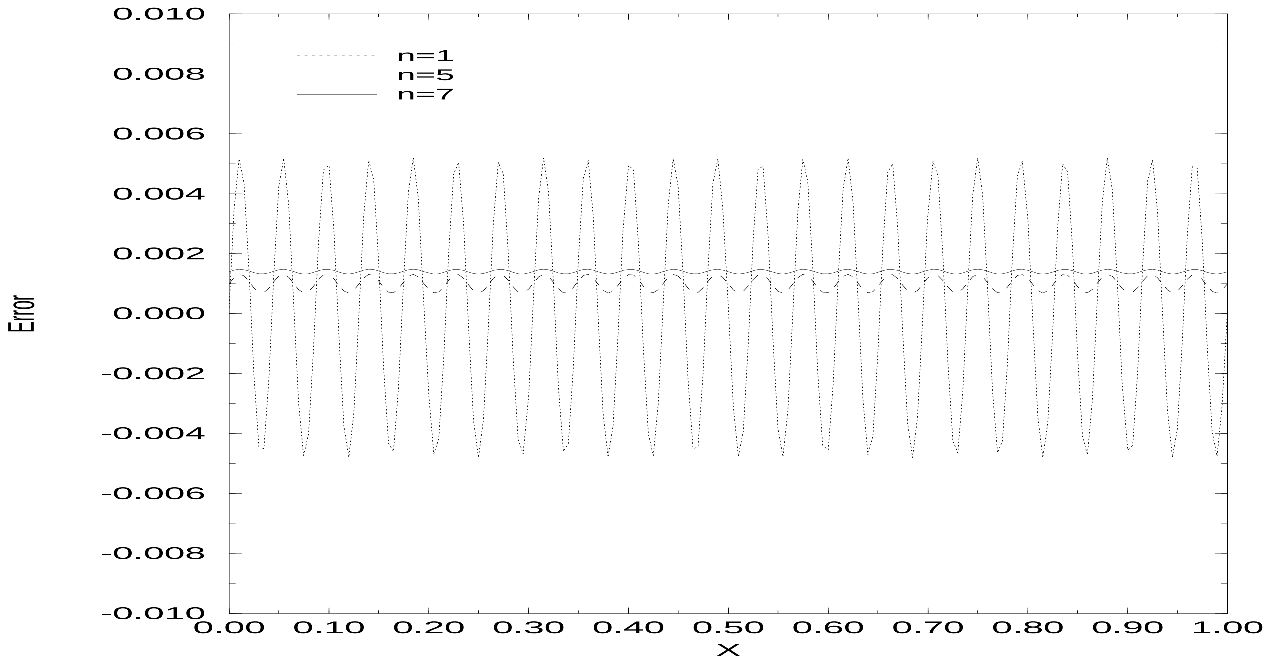


Figure 19: Error found in simulation of EU(2)-I diving the domain into two subdomains at $x = 0.5$, after introducing sinusoidal perturbation at initial conditions. The snapshot is taken after n=1,5 and 7 time-steps.

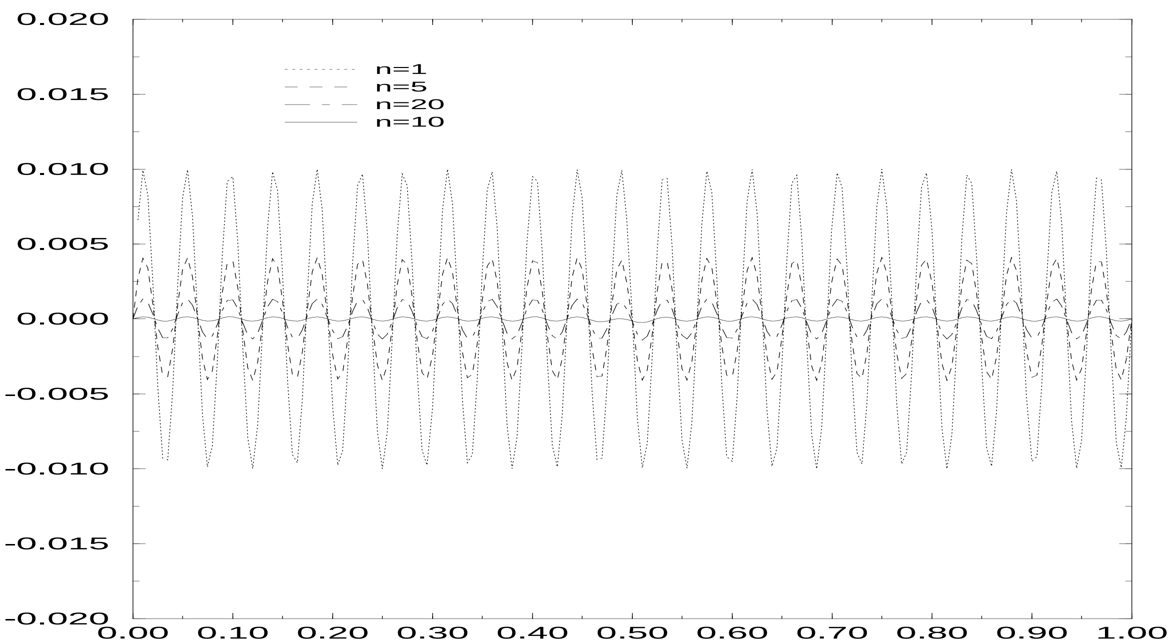


Figure 20: Error found in simulation of PAFD-I diving the domain into two subdomains at $x = 0.5$, after introducing sinusoidal perturbation at initial conditions. Running asynchronously, the right side was updated at the rate of 9:10. The snapshot is taken after n=1, 5, 10 and 20 time-steps.

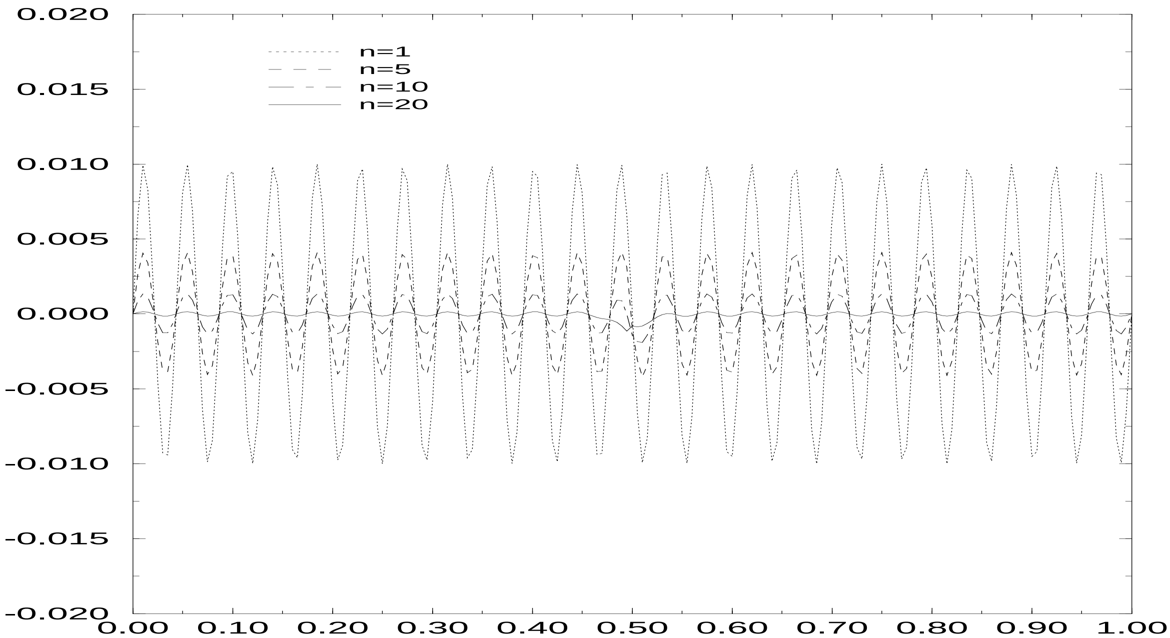


Figure 21: Error found in simulation of PAFD-I with parameters as in Fig. refstabpafd1x9ps, except that the right side is updated at a rate about 50% slower than the left. The snapshot is taken after n=1, 5, 10 and 20 time-steps.

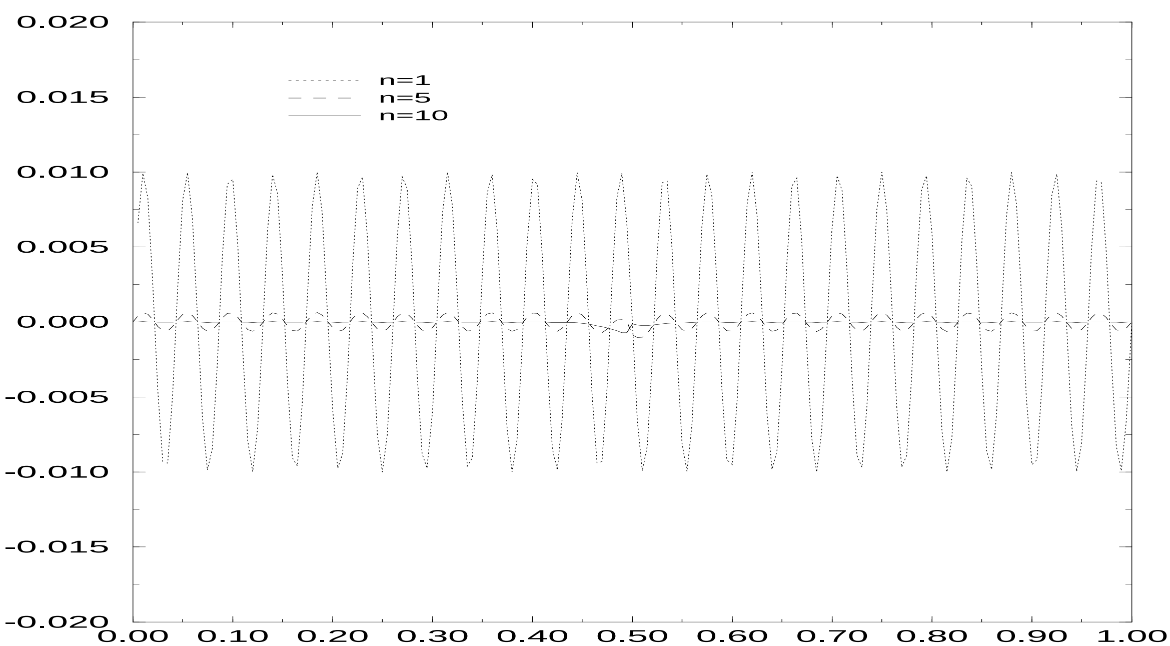


Figure 22: Error found in simulation of PAFD(2)-I after introducing sinusoidal perturbation at initial conditions. Running asynchronously, the right side was updated at up to half the rate of the left. The snapshot is taken after n=1,5 and 10 time-steps.

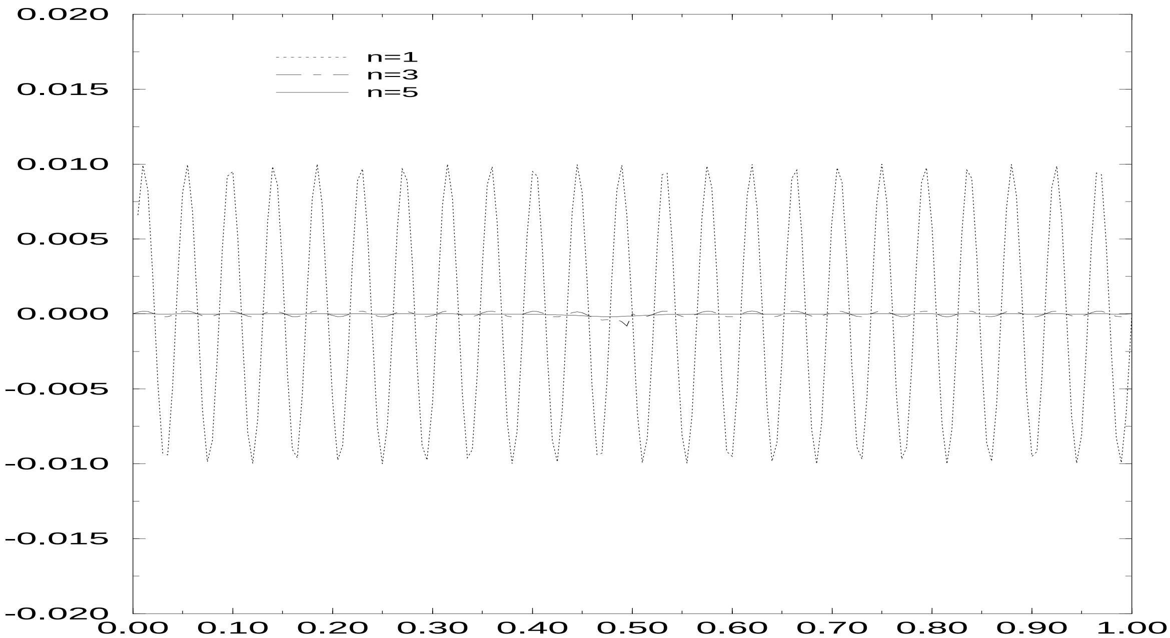


Figure 23: Error found in simulation of PAFD(5)-I after introducing sinusoidal perturbation at initial conditions. Running asynchronously, with the right hand side is updated at a rate about 10% slower than the left. The snapshot is taken after n=1,3 and 5 time-steps.

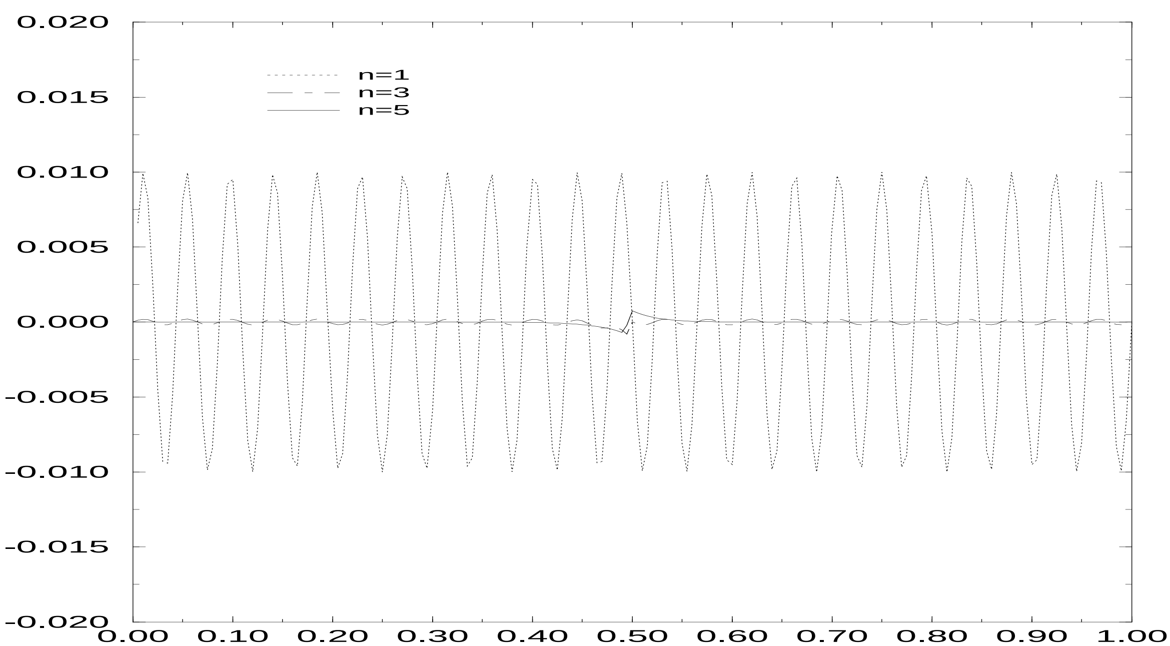


Figure 24: Error found in simulation of PAFD(5)-I after introducing sinusoidal perturbation at initial conditions. Running asynchronously, with the right hand side is updated at a rate about 50% slower than the left. The snapshot is taken after n=1,3 and 5 time-steps.

3 Numerical Results for the PAFD Performance

We have implemented and examined our schemes on two different MIMD architectures: the multiuser *Sequent Symmetry* [11] multiprocessor which is a commercial shared bus&shared memory MIMD multiprocessor, and the *iPSC/860* [14] which is a commercial message-passing machine. The *Sequent Symmetry* machine we have used is composed of 20 i386(16MHz) processors connected by a high speed synchronous bus to a shared memory of 32M bytes. The processors have no local memory, but do have a 64KB local cache. The operating system is DYNIX(R) V3.1.4. The *iPSC/860* machine is composed of 32 i860 nodes, each with 8MB of local memory, 8KB of data cache, and 4KB of code cache, connected in hypercube topology. The operating system is NX 3.3.1.

The performance of the proposed scheme is illustrated and compared with others in Tables 2 and 3 and in Figures 25 and 26. Numerous schemes have been tested to highlight the overhead associated with the different architectures. Specifically, the following *parallel* approximation schemes are considered:

- **SY** - The synchronous explicit Euler scheme (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 0$ and $\theta'_{1_l} = \theta'_{2_l} = 1$) calculated at *every* grid point; all processors are synchronized at each iteration (lock-step) using a barrier.

- **TS** - The asynchronous Time-Stabilizing (TS) method of [5] which else can be derived from Eqs. (4a)-(4b) with $\theta_1 = \theta_2 = 0$ ($\Longrightarrow \theta'_1 = \theta'_2 = 1$). However, as had been indicated earlier, since Eq. (6) is not satisfied its spatial accuracy is first order only. This is an asynchronous scheme with adaptive time-increments. Its *time-stabilizing property* assures that the derivatives of the time-levels with respect to the spatial coordinates are bounded as the number of time-steps increases, and they vanish as the spatial mesh-size approaches zero. Therefore, with its time-stabilizing property this scheme provides accurate results even when there is a fixed delay among the processors, i.e when certain processors are always slower than the others. Thus, unlike those of [2], it is suitable for multiprocessors with non-identical processors or unbalanced workload, provided that the differences among their speeds/loads are not too large.

- **PAFD** - Our Parametric Asynchronous Finite Difference approximation (Eqs. (13a)-(13d) together with the requirements of Lemma 2) is used at *every* single internal grid point.

- **PAFD-E** - An asynchronous domain decomposition scheme which uses Euler explicit approximation (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 0$ and $\theta'_{1_l} = \theta'_{2_l} = 1$) within all subdomains and the PAFD approximation (Eqs. (13a)-(13d) together with the requirements of Lemma 2) at subdomains' boundaries.

- **PAFD-I** - An asynchronous domain decomposition scheme which uses Backwards-Euler implicit approximation (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 1$ and $\theta'_{1_l} = \theta'_{2_l} = 0$ [16]) within all subdomains and the PAFD scheme (Eqs. (13a)-(13d) together with the requirements of Lemma 2) at subdomains' boundaries.

- **PAFD(k)-I** - An asynchronous domain decomposition scheme which uses Backwards-Euler implicit approximation (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 1$ and $\theta'_{1_l} = \theta'_{2_l} = 0$ [16]) within all subdomains, as above, but the PAFD scheme (Eqs. (13a)-(13d) together with the requirements of Lemma 2) is performed at subdomains' boundaries with larger spatial discretization ($\Delta x = kh$ instead of $\Delta x = h$).

- **EU-I** - A parallel scheme based on domain decomposition. Within each subdomain we use Backward-Euler implicit approximation (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 1$ and $\theta'_{1_l} = \theta'_{2_l} = 0$ [16]). The solution values on the subdomains' boundaries are determined by using Forward-Euler explicit approximation (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 0$ and $\theta'_{1_l} = \theta'_{2_l} = 1$)

- **EU(k)-I** - An asynchronous domain decomposition scheme similar to EU-I except that whereas with EU-I the same spatial discretization ($\Delta x = h$) is used within subdomain and on subdomains' boundaries, in this case a larger discretization is used on subdomains' boundaries ($\Delta x = kh$).

The above parallel schemes are compared with the *serial* Backward-Euler (**SER-I**) implicit method as well as with the serial Euler (**SER-E**) explicit scheme. Both are second-order approximations in space, and first-order approximations in time.

The following parameters are used:

- Δx - The discretization length along the $x-$axis.

- Δt - The discretization length along the $t-$axis.

- T - The time level to be reached in the given problem.

- p - The number of processors used for the calculations.

- L - The number of grid points in the considered domain.

- τ_p - The clock time required for completing calculation of the sample problem according to a specific scheme by the last (slowest) processor.

- *Speedup* - Run time of the corresponding serial algorithm relative to the parallel algorithm. The corresponding *efficiency* is defined as the speedup divided by the number of processors.

- e - The maximal error.

- E - The maximal relative error of the numerical result (the numerical solution relative to the analytic solution).

- N - Number of time-steps needed by the corresponding scheme, in order to reach $t = T$.

- I - Number of times when no intersection point was found during the run of PAFD (valid only for PAFD schemes).

We demonstrate the above with the following problem:

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

$$u(x,0) = \frac{1}{12}x^4 \quad , \quad u(0,t) = t^2 \quad , \quad u(1,t) = t(1+t) + \frac{1}{12}$$

whose analytic solution is:

$$u_A = x^2 t + t^2 + \frac{1}{12}x^4$$

Table 2 presents the results for the case where the PAFD scheme is applied at all internal grid points. The serial reference for speedup evaluation is SER-E. Note that since two-levels values (at t^n and at t^{n+1}) are required by the PAFD, in this implementation of the PAFD scheme we obtained the *first* pair of two-levels values, by using the initial condition and the Forward-Euler scheme (Eq. (2) with $\theta_{1_l} = \theta_{2_l} = 0$ and $\theta'_{1_l} = \theta'_{2_l} = 1$).

Scheme	p	$\frac{L}{p}$	Speedup	E	N	I
SY	2	400	1.38	0.009970	2554	-
	5	160	3.41	0.009970	2554	-
	10	80	6.64	0.009970	2554	-
	16	50	10.07	0.009970	2554	-
TS	2	400	1.80	2.1587	1197-1481	-
	5	160	4.45	2.2814	1215-1414	-
	10	80	8.71	4.9950	1219-1394	-
	16	50	13.34	5.5759	1222-1431	-
PAFD	2	400	0.56	0.009966	2554-2777	22
	5	160	1.40	0.009978	2555-2747	13
	10	80	2.76	0.01017	2554-2772	25
	16	50	4.40	0.01085	2916-3272	501
SER-E	1	800	-	0.009970	2554	-

Table 2: Performance comparison for the 1-D case on the *Sequent*: $L = 800, \Delta x = 0.00125, T = 0.002$ where $\Delta t = \frac{1}{2}(\Delta x)^2$ for SER-E and SY. The PAFD scheme is applied on *every* internal grid point.

We observe that the TS scheme is about 30% faster than SY. However, the accuracy is degraded substantially. Performing PAFD at each grid point is almost as accurate as SY, but, at the cost of additional memory (two-level values are maintained per each grid point, instead of one compared with the TS scheme) and additional processing-time (even more than twice as slow than SY), is needed for evaluating t_{I_l}. The results in Table 2 are shown for the *Sequent*, even poorer results are expected on the *iPSC/860* due to slower communication via message-passing.

We note that the higher-order accuracy of our PAFD scheme was even more evident in tests we have done on a serial computer simulating a multiprocessors machine. For instance, in cases where all simulated processors run at the same speed, except one which was slower than the others (10% up to 35% slower). In some cases the PAFD was even more accurate than SY, while TS was quite significantly less accurate.

The PAFD scheme is slow mainly due to the complicated search for the intersection point t_{I_l} performed for each grid point and for $1 \leq l \leq d$. Nevertheless, this search for each single grid point is not necessary since we may assign several grid points per processor such that all points within a single processor's subdomain are synchronized (i.e. are guaranteed to be in the same time-level at any iteration). Then the PAFD scheme can be used only on grid points of subdomains' boundaries.

Scheme	p	$\frac{L}{p}$	τ_p relative to SER-I	τ_p relative to SER-E	$\tau_p(sec)$	e	N	I
PAFD	2	300	0.28	0.32	197.40	0.007095	3589-3649	4
	5	120	0.69	0.78	79.63	0.006783	3590-3667	2
	5	120	0.66	0.74	84.33	0.0	3591-3660	3
	10	60	1.38	1.55	40.15	0.01629	3598-3698	5
	15	40	2.03	2.28	27.22	0.01132	3595-3763	9
	20	30	2.74	3.08	20.17	0.04601	3610-3754	10
PAFD-E	2	300	0.92	1.04	60.03	0.007096	3584-4215	0
	5	120	1.65	1.86	33.50	0.01978	3589-5578	1933
	10	60	2.83	3.17	19.60	0.04686	3591-6105	2120
	15	40	4.00	4.50	13.83	0.04109	3590-6271	1415
	20	30	4.86	5.46	11.39	0.03671	3591-6330	567
PAFD-I	2	300	0.45	0.51	121.99	0.01418	3581-4053	0
	5	120	0.81	0.91	68.29	0.01365	3603-5597	969
	10	60	1.57	1.77	35.23	0.01904	3597-5675	437
	15	40	2.28	2.56	24.28	0.03459	3596-5754	786
	20	30	2.92	3.28	18.96	0.03251	5134-5771	2259
SER-I	1	600	-	-	55.37	0.03548	1196	-
	1	600	-	-	156.59	0.01419	3588	-
SER-E	1	600	-	-	62.19	0.007559	3588	-

Table 3: Performance comparison on the *Sequent* for the 1-D case: $L = 600, \Delta x = 0.00167, T = 0.005$. For PAFD-E and PAFD-I the parametric scheme is applied only at subdomains' boundaries.

Table 3 presents results for the case when the PAFD scheme is applied *only* at subdomains' boundaries while within each subdomain an implicit or explicit scheme is used. Specifically, PAFD-E (PAFD-I) denotes the scheme for which the PAFD approximation is used only along subdomains' boundaries while the explicit Euler (implicit Euler) scheme is used within each subdomain.

We note that using explicit scheme within the subdomain restricts the time-step to that of the explicit scheme even in cases where the PAFD allows us to maintain larger time-steps on the subdomains' boundaries. Using implicit scheme within a subdomain removes this restriction, but then the time-step is restricted by the PAFD which is an explicit scheme in nature. Normally, run-time of implicit scheme equals the run-time of an explicit scheme with time-step three times as much. This is also the case here, where SER-I which is an implicit scheme, and therefore permits larger time-step. We observe from Table 3 that the PAFD-E scheme is faster than PAFD-I and both are faster than PAFD, at the cost of less accuracy; the PAFD provides the best accuracy at the price of quite poor efficiency (= speedup/p).

The same kind of observations were perceived in the two-dimensional case. For two-dimensional problems the PAFD has the same proportional work done per each grid point. Also the same relation between the number of points within the subdomain and on its boundaries is maintained and therefore PAFD-E is still much faster. However, implicit 2-D schemes are significantly more complicated. We used an ADI type of scheme yielding results relatively even poorer than the 1-D case.

Thus, it is evident, that using the higher-order Parametric Asynchronous Finite-Difference scheme per every point within the subdomain, although allowing asynchronous parallel computation, are suboptimal. It takes several processors to achieve the same performance as an implicit scheme performed by a single processor (SER-I). It requires too much processing per grid point or on subdomains' boundaries, and yet is explicit in nature and thereby limits the allowed time-step severely. Using the PAFD scheme at subdomains' boundaries provides better results, but it is still far from linear, where linear is taken as the run time for SER-I divided by the number of processors. This improvement is due to the elimination of unnecessary computation for t_{I_l} within the subdomain.

A side effect of the domain-decomposition, that may have a strong impact on the performance is related to the cache size. In cases where the whole domain cannot be cashed, whereas a subdomain is cashed, the parallel code runs even faster than optimal. This is clearly seen in Fig. 25 in which PAFD(k)-I is faster than optimal.

Figures 25 and Fig. 26 are a graphical representation of run times on the Sequent and on the iPSC/860, respectively. Here, a major improvement is achieved when a larger time step is allowed. This is the case in which PAFD is performed on the subdomains' boundaries based on the k-remote neighbor ($k \cdot \Delta x = H$), i.e the spatial discretization is an integral multiple of H. In the case of Figs. 25 and 26, $k = 3$. We note the effect of the communication overhead: whereas the cost of accessing a neighboring value is insignificant on the Sequent, this cost is still significant on the iPSC/860. Thus, the speedup achieved on the Sequent is better.

4 Summary

This paper presents parallel *asynchronous* schemes in which the need for global synchronization is eliminated. Thus, each processor runs at its own speed (e.g. in a multi-user or a master/slave environments), or alternatively, no a priori assumptions are made on the processors' load (e.g. allowing distribution of unequal load, or processing of unstructured

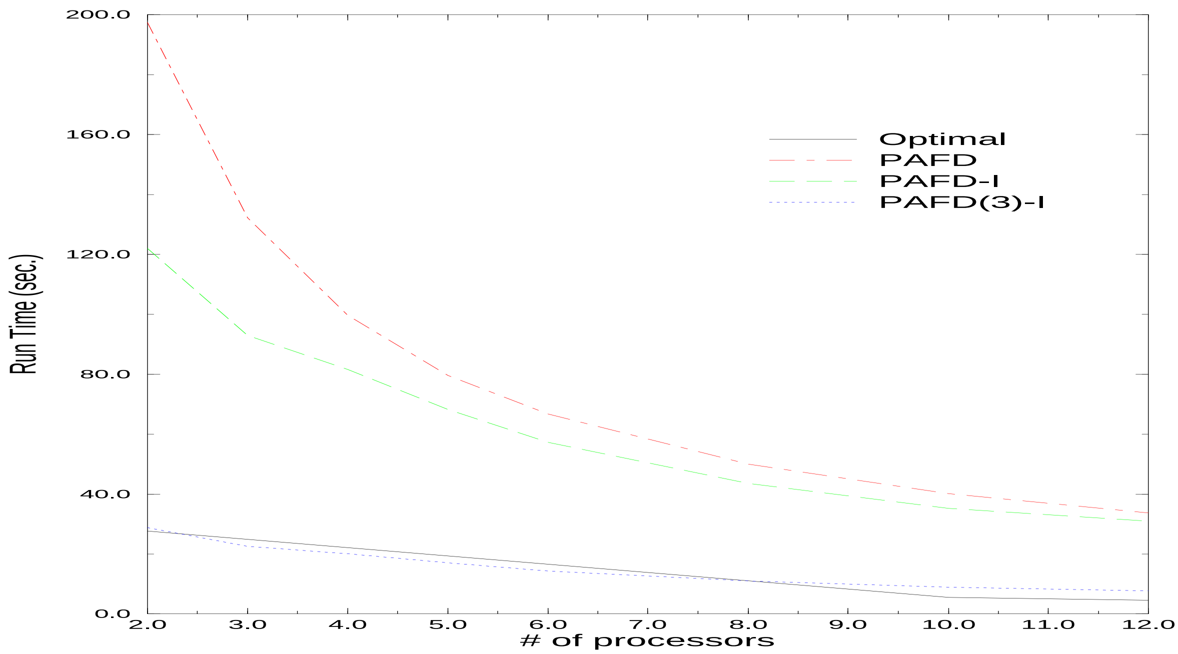


Figure 25: Run time comparison on the Sequent

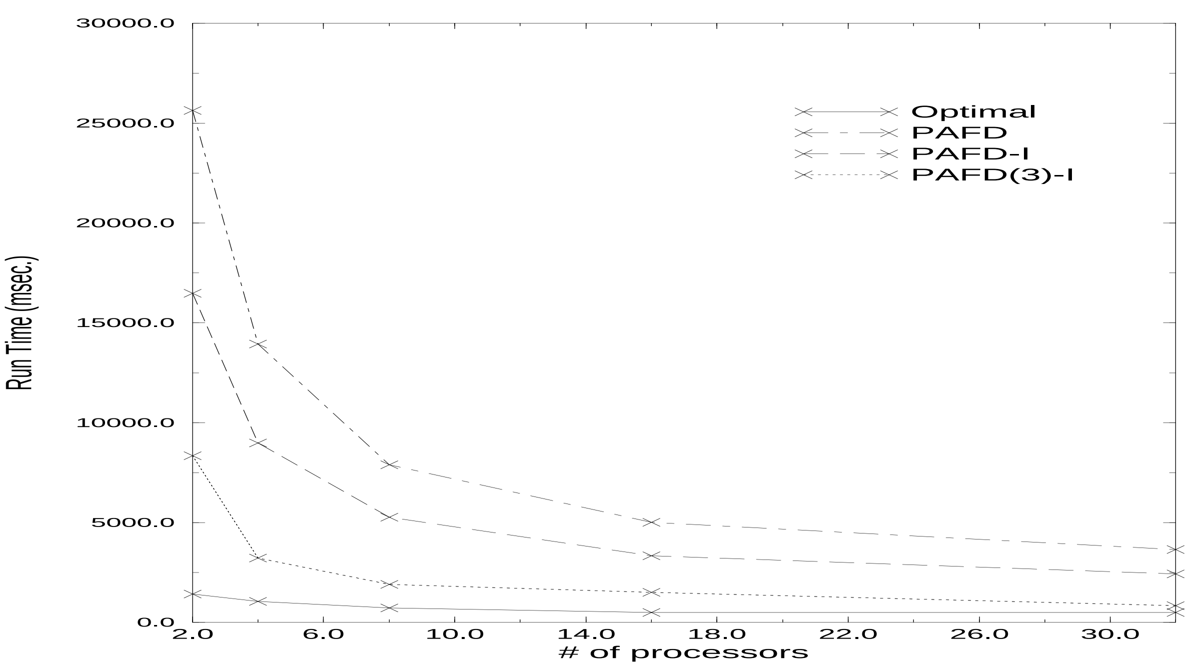


Figure 26: Run time comparison on the iPSC/860

grids). Specifically, we develop new parallel *asynchronous* schemes for the solution of the multi-dimensional heat equation, of which the well-known synchronous schemes are a special case. It is a generalization and expansion of the results in [5] for parabolic PDEs. In addition, a similar approach is applicable for hyperbolic PDEs [3].

These schemes remove the synchronization overhead and are suitable for single (or multi) user shared-memory or message-passing MIMD multiprocessors. They were implemented on the multi-user *Sequent* which is a shared bus&shared memory MIMD multiprocessor, and on the *iPSC/860* which is a message-passing MIMD multiprocessor

First, we have presented a Parametric Asynchronous Finite-Difference (PAFD) scheme where its accuracy is second-order in space, and requires only the communication of neighboring values, but no synchronization. The consistency, stability and convergence of the PAFD scheme has been examined in detail. However, numerical tests showed that although the PAFD scheme provides a high accuracy, its efficiency is inadequate when performed per each grid point.

In an alternative approach based on domain decomposition, each processor is assigned the task of processing its own subdomain *asynchronously* of the other processors. We have presented an algorithm in which an implicit finite-difference scheme is used within each subdomain, while the values at subdomains' boundaries are determined by the PAFD. Numerical tests showed that this approach provides high accuracy and almost linear speedup.

References

[1] D. Amitai, A. Averbuch, S. Itzikowitz, and E. Turkel. Asynchronous Numerical Solution of PDEs on Parallel Computers, In J. Dongarra P. Messina D. Sorensen and R. Voigt, editors, *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing*, pages 131–136. SIAM Press, Philadelphia, (1990).

[2] D. Amitai, A. Averbuch, S. Itzikowitz, and E. Turkel. Asynchronous and Corrected-Asynchronous Numerical Solution of Parabolic PDEs on MIMD Multiprocessors, *ICASE Report No. 91-59*, *NASA CR-187605*, ICASE, NASA Langley Research Center, Hampton VA, (1991).

[3] D. Amitai, A. Averbuch, M. Israeli, and S. Itzikowitz. An Asynchronous Approach to the Numerical Solution of Hyperbolic PDEs on MIMD Multiprocessors, In V.P. Bhatkar, A. Basu, S.C. Purohit and K.M. Rege, editors, *Frontiers in Parallel Computing*, pages 233–242. Narosa Publishing House, New Delhi, (1991).

[4] D. Amitai, A. Averbuch, M. Israeli, and S. Itzikowitz. Parallel Adaptive and Time-Stabilizing Schemes for Constant-Coefficient Parabolic PDEs. *Eskenasy Institute of Computer Science Technical Report 250/92*, Tel-Aviv University, Tel-Aviv, (1992).

[5] D. Amitai, A. Averbuch, M. Israeli, and S. Itzikowitz. Parallel Adaptive and Time-Stabilizing Schemes for Constant-Coefficient Parabolic PDEs. *Comp. & Math. w its Appl.*, Vol. **24** No. 10, 33-53 (1992).

[6] D. Amitai, A. Averbuch, M. Israeli, S. Itzikowitz, and E. Turkel. A Survey of Asynchronous Finite-Difference Methods for Parabolic PDEs on Multiprocessors. *Appl. Numer. Math.*, **12**, 27-45 (1993).

[7] D. Amitai, A. Averbuch, S. Itzikowitz, and E. Turkel. Asynchronous and Corrected-Asynchronous Finite Difference Solutions of PDEs on MIMD Multiprocessors. *Numerical Algorithms*, **6**, No. III-IV, pp. 275-296 (1994).

[8] G. M. Baudet, Asynchronous Iterative Methods for Multiprocessors, *J. of the ACM*, **25**, 226-244 (1978).

[9] D. P. Bertsekas, and J. N. Tsitsiklis, *Parallel and Distributed Computation numerical methods.* Prentice-Hall, 1989.

[10] D. Chazan and W. Miranker, Chaotic Relaxations, *J. of Linear Algebra Applications*, **2**, 199-222 (1969).

[11] Sequent Computer Systems Inc., The Sequent Guide to Parallel Programming, *1003-44459* (1987).

[12] C. N. Dawson, Q. Du and T. F. Dupont, A Finite Difference Domain Decomposition Algorithm for Numerical Solution of the Heat Equation, *Mathematics of Computation*, **57**, 63-71 (1991).

[13] D. J. Evans. New parallel algorithms for partial differential equations. In J. Joubert, M. Feilmeier and U. Schendel, editors, *Parallel Computing 83*, pages 3–56. Elsevier Science Publishers B.V., North-Holland (1984).

[14] Intel Corporation, iPSC/2 and iPSC/86 Programmer's Reference Manual, *order number: 311708* (1990).

[15] D. Mitra, Asynchronous Relaxations for the Numerical Solution of Differential Equations by Parallel Processors, *SIAM J. Sci. Stat. Comput.*, **8**, 43-53 (1987).

[16] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial Value Problems.* J. Wiley & Sons, New-York, 1967.

[17] V. K. Saul'yev (translated by G. J. Tee). *Integration of Equations of Parabolic Type by the Method of Nets.* Pergamon Press, 1949.

[18] A. Sommerfeld. *Partial Differential Equations in Physics.* Academic Press, New-York, 1949.