Implicit-Explicit Parallel Asynchronous Solver of Parabolic PDEs

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

Abstract

This paper presents a method for the solution of parabolic PDEs on *parallel* computers, which is a combination of implicit and explicit finite difference schemes based on a domain decomposition (DD) strategy. Moreover, this method is *asynchronous* (i.e., no explicit synchronization is required among processors). We determine the values at sub-domains' boundaries by our new high-order asynchronous *explicit* schemes. Then, any known high-order *implicit* finite difference scheme can be applied within each sub-domain. We present a technique for derivation of appropriate *asynchronous*-explicit schemes based on *Green's functions.* Synchronous versions of these schemes are obtained as special cases. The applicability of this method is also demonstrated for a family of non-linear problems.

Our new explicit schemes are of high-order and yet stable for a large time step, as established in our analysis of their numerical properties. Moreover, these schemes provide attractive properties for parallel implementation. Being *asynchronous*, they allow local time stepping, thus eliminating the need for a global synchronized time step. Moreover, our asynchronous computation is *time-stabilizing*, in the sense that the calculation implicitly prevents a growing time gap between neighboring sub-domains. The *locality* property, due to the exponential decay of the Green's functions, implies that communication is needed only between neighboring processors.

Hence, this method which is designed to minimize the overhead associated with the synchronization of the multiple processors, is specifically suitable for parallel computers having a high synchronization cost or highly varying load, even in cases in which some processors have persistent speed differences. Furthermore, the implementation of different resolution in each sub-domain (e.g., irregular or unstructured grid) makes it valuable as an adaptive algorithm.

The above schemes were implemented and tested on the shared-memory multi-user Cray J90 and Sequent Balance machines. These implementations prove high accuracy and high degree of parallelism. This work is complementary to our previous work on asynchronous schemes [1] - [4].

Key-Words:
Parallel asynchronous and synchronous finite difference methods, parabolic finite difference approximations with constant coefficients, synchronization overhead, Green's function-based approximations.

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

*Department of Computer Sciences, School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, e-mail: dganit@math.tau.ac.il

†Faculty of Computer Science, Technion, Haifa 32000, Israel. His research was partially supported by the fund for promotion of research at the Technion

‡Current address: The College of Management-Academic Studies, The School of Business Administration, Rishon Lezion Campus, 75420, Israel. This work was done when he was with the Department of Computer Sciences, School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel

1 Introduction

1.1 Domain Decomposition

The numerical solution of partial differential equations (PDEs) is a good example of a computation that has a potential to benefit from parallel processing. This computation usually requires the processing of many grid points that is beyond the capacity of most non-parallel computers, particularly for an increased domain resolution. Moreover, PDEs have the potential for an *efficient* parallelization due to the *local* behavior of partial derivatives. Whereas, poor parallelism is expected in cases where the physical properties dictate *global* communication, or require massive data transfer such as Fourier transform or performing a transpose of a matrix. In our subsequent discussion we introduce a method for an efficient parallel asynchronous solution of parabolic PDEs. It is based on finite difference schemes.

There are two basic types of finite difference schemes. *Explicit* finite difference schemes express the values of variables at a later time as a combination of values in a small local spatial neighborhood at a previous time. As a result, only a small number of neighboring degrees of freedom is involved in the approximations especially for a low order precision. In this sense, explicit schemes are *local* and consequently easy to program and to be implemented on parallel environments. However, they suffer the disadvantage of severely restricted time step size as a result of a stability condition. Thus, explicit time marching schemes are considered inefficient.

Implicit schemes generally have no limitations on their time stepping. Nevertheless, for each time step a system of linear equations has to be solved. Hence, to compute a new time they require values of all the points within the problem domain. In this sense, implicit methods have *global* nature and their implementation on parallel computers is not straight-forward. Moreover, significantly larger number of arithmetic operations is involved in solving the linear system. This is compensated by allowing larger time step and generally by achieving higher accuracy.

In this paper we focus on the implementation of implicit schemes on parallel computers, allowing implementation even in cases where the processors are solving the problem *asynchronously* (i.e., the processors are allowed to compute at different speeds, or an uneven load). In many parallel methods, parallelism is often achieved by a "divide & conquer" strategy using several discrete sub-domains. Likewise, we base our method on a domain decomposition. The fundamental concept of *Domain Decomposition* (DD) techniques is to divide the computational domain into smaller sub-domains and the original problem into sub-problems such that each is solved in a corresponding sub-domain. The division and the computation of solutions at each sub-domain has to be done in such a way to provide an accurate global solution. Domain decomposition is often used for various other motivations such as to simplify problems solved on complicated geometry. When used for parallelization, each sub-domain is allocated to a different processor. Consequently, the computational question is how to manage the interactions among the sub-domains for obtaining an accurate global solution based upon the local solutions while minimizing processor communication and synchronization.

In our case, we use a known implicit scheme within each sub-domain so that a large time step can be used. However, we develop a new scheme for the determination of the solution on sub-domains' boundaries. This scheme addresses two major requisites: First, it is explicit and *local*, in the sense that it necessitates only neighboring values. Thus, it requires a communication only between neighboring processors. Second, it is *asynchronous* in the sense, that the neighboring values are not necessarily associated with the same time step. Thus, we allow the processors to compute at different speeds, or an unbalanced load. Moreover, we assume no explicit synchronization.

Our parallel solution method is actually a combination of a known implicit scheme within each sub-domain and a new asynchronous-explicit scheme on sub-domains' boundaries. Both schemes enable large time steps and thereby increase the efficiency. This combination indeed provides an efficient and accurate method. A summary of the properties of both implicit and explicit schemes is presented in the first two columns of Table 1. In the following section, we develop such an asynchronous-explicit scheme based on Green's functions. We emphasize that it is explicit yet, absolutely stable, allowing a large time step (see claims 3 and 4). Moreover, due to the exponential decay of the Green's functions it is local, and require communication only between neighboring processors (see claim 6). A summary of its properties and a reference to the subsequent discussion is found in the third column of Table 1.

Within sub-domain	*On sub-domain boundaries* *General*	*Green-Based*
implicit	explicit	see schemes Eqs.(3.37) and (3.41)
large Δt	large Δt	absolutely stable, see claims 3 and 4
accurate	accurate	$O(\Delta t^2 + \Delta x^2)$ or higher, see Appendix
—	local	see claim 6
—	asynchronous	see scheme Eq.(3.41)

TABLE 1: Characteristics of the proposed schemes

Numerical methods that utilize both explicit and implicit schemes are not new. The simplest example is the alternate use of forward-Euler (explicit) for even nodes and Backwards-Euler (implicit) for odd nodes. Since even nodes are computed first, the method is indeed explicit. It requires the same number of operations as the forward-Euler explicit method. Its stability condition allows a time step twice as large as that of the standard explicit. Moreover, it is clearly more accurate due to error cancellation ($O(\Delta t^2 + \Delta x^2)$ instead of $O(\Delta t + \Delta x^2)$). This method can be easily applied on parallel computer like any explicit method. A significantly more accurate method is obtained when alternate schemes are applied both in time and space (i.e., four schemes, two for odd and even nodes and two for odd and even time steps) [17]. In both these cases of implicitly-explicit methods, the implicit schemes are computed when all the required values are available, resulting an explicit method which is easy to parallel. Another approach, yielding a stable explicit method is by smoothing. (e.g., [17], p.69-70). First, the standard explicit Forward-Euler scheme is used. However, since this method allows a time step greater than that permitted by the stability condition, the values are distorted. This is compensated by an additional smoothing iterations.

Unlike those "implicitly"-explicit methods, we show how a genuine implicit scheme can be applied on a parallel computer. It is done by applying a genuine explicit scheme only at specific points (sub-domains' boundaries). Thus, our method is characterized as a unique combination of an explicit and a *truly* implicit schemes. This is detailed in section 2. Furthermore, we address the issue of an asynchronous computation by using an asynchronous-explicit scheme, as discussed in the following sub-section.

1.2 Asynchronous Schemes

The straight-forward Domain Decomposition approach usually requires a high degree of global synchronization to assure an accurate global solution derived from solutions at each sub-domain. This is equivalent to the assumption that the algorithm is governed by a global clock which dictates the advance of each processor. This synchronization is significant as it may severely degrade the speedup gained by parallelism. Obviously, the main objective is to perform as much work as possible within each sub-domain while minimizing the overhead of communication and synchronization among the processors. However, to compute time marching finite difference schemes correctly, each processor depends on the results previously computed by other processors. In a *synchronous* algorithm the computation is divided into steps. When computing the next step, each processor depends on the results computed by other processors for neighboring sub-domains, during current and previous steps. Usually, this exchange of values is done at the end of a step and all processors simultaneously advance to the next step. In an *asynchronous* algorithm there is no global simultaneous step but rather each processor advances at its own speed and the coordination of the computations is less strict. Obviously, an *asynchronous* algorithm has the potential to be more efficient as it can execute more time steps without having to use a synchronization mechanism and wait for results obtained by other processors.

Clearly, there is a certain inherent inefficiency in requiring processor not to proceed on to its next iteration before the rest of processors finish their same iteration (see Fig. 1). This overhead consists of several factors. First, a fast processor is delayed until the slowest processor finishes. Thus, the pace of the algorithm is dictated by the slowest processor (see as an example: Fig. 1). There are various reasons why certain processors will be ahead of the others, even though they are physically configured at the same speed:

Load Balancing - In many problems it may be useful to divide the problem unequally among the processors due to different boundary conditions or approximation schemes that are used or complicated geometry that dictates different grid structure. 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 other. This can be significant depending on the problem topology. Even for simple topologies, the borders usually require a different treatment resulting a different computation time than that required for the rest.

Multi-user environment - In such environment many tasks are simultaneously assigned to each processor. Since a single processor can be used for any of these tasks at any particular time, we cannot predict how much will be devoted for the computation of our algorithm. In the case of the *Sequent* we found run-time differences from 5% to 15% for processors executing the same loop, depending on the system load.

Random noise - Any unpredictable perturbation that can cause a momentary delay e.g., on the *Sequent* running as a single user we noted run-time differences from $\frac{1}{2}$% to 3% among processors executing the same loop.

Master/Slave - In such environment it is clear that a certain processor, the *master*, is doing some 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. On the *Sequent* the code is loaded to one processor and than copies are *forked* to the other processors. Thus, this processor is available only after all processors are initiated.

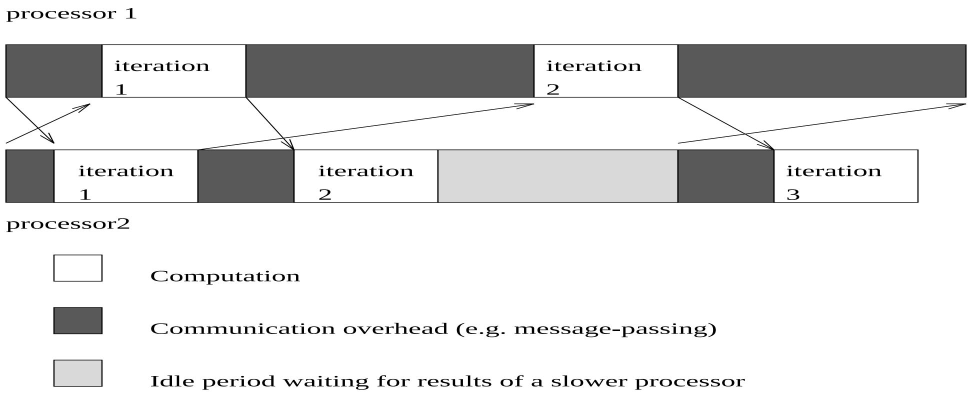

FIG. 1. : Synchronization overhead due to slow processor causing delay while waiting for the second iteration to finish, in addition to the communication overhead.

Moreover, 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. Here, a slow communication channel slows the progress of the entire computation. In addition, in some situations synchronization may cause contentions over communication resources and memory accesses.

An asynchronous algorithm can potentially reduce the synchronization penalty since each processor can execute more iterations where it is not constrained to wait for the most recent results of the computation from other processors. The danger is that iterations performed on the basis of outdated data will not be as efficient or even not be productive. The proposed scheme utilizes additional computation to compensate for an "outdated" data. In addition, it eliminates the programming efforts involved in setting up the synchronization mechanism and thus simplifies the task management.

Previously, *asynchronous iterations* were considered for iterative solutions of systems of linear equations in [5] and in [7] specifically for multiprocessors. For the solution of ordinary differential equations they are used in [14] and proposed for partial differential equations in [15]. Asynchronous finite-difference schemes were developed in [2], [3] for the steady-state and non steady-state multidimensional heat equation. While these schemes are based on constant time increments, those presented in [1] and [4] are asynchronous with adaptive time increments. In addition, they have a *time-stabilizing* property: the derivatives of their time 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 those of [3], they provide accurate results even when there is a fixed delay among the processors i.e., when certain processors are always slower than the others. However, the high efficiency of these adaptive schemes is achieved at the price of having them with only a first-order accuracy.

Similarly, in this paper we study the question of how to evaluate the solution in each sub-domain locally and *asynchronously* (independently) of other sub-domains. But unlike the methods referenced above, we utilize the asynchronous schemes only on sub-domains' boundaries in combination with implicit schemes which are used within each sub-domain. Since the implicit scheme is absolutely stable, it permits large size of time steps. Likewise, we seek an explicit scheme which permits large size of time steps. This combination provides a stable asynchronous method. Hence, it can be utilized when the load is not distributed equally among the processors, or for problems in which the domain is divided into unequal resolutions (e.g., due to their geometry, irregular grid or unstructured grid).

By choosing an adaptive time-step size, we implicitly control the processors' advance. Hence, similar to [1] we have a time-stabilizing property. However, whereas in [1] the time-step size was limited by the stability condition, our method is stable and the restriction is done to assure high-order of accuracy.

Specifically, we examine here the usage of Green's function for obtaining high-order asynchronous schemes which are used for the solution of the multidimensional heat equation in a bounded region. Nevertheless, this should serve only as a model problem for other possible schemes and for other problems. Potential extensions are solving a family of non-linear problems by linearization (following [10]) discussed in section 3.1.3, or else (e.g. for Navier-Stokes equations) by splitting the computation so that the linear part which consists of constant-coefficients can be solved using this method.

The paper is organized as follows. In section 2 we present our method. First, we present the notation and general description in sub-section 2.1. The algorithms for synchronous and asynchronous mixing of implicit and explicit schemes are discussed in sub-section 2.1.1, and the general form of an synchronous-explicit scheme in sub-section 2.1.2. Numerical analysis is presented in sub-section 2.2. First, for the method of mixing implicit and explicit schemes in 2.2.1 and then for the general synchronous-explicit scheme in 2.2.2. In section 3 we discuss the computation of the subdomains' boundaries. In sub-section 3.1 we introduce the technique to derive asynchronous schemes for approximating the boundaries, based on Green's function. The one-dimensional case is presented in subsection 3.1.1, and the multidimensional case in subsection 3.1.2. An extension to a family of non-linear problems is presented in subsection 3.1.3. Implementation and computational considerations are discussed in sub-section 3.2. First, finite differences forms of the sub-domains' boundaries approximations are detailed in 3.2.1 and their numerical properties are analyzed in 3.2.2. The algorithmic performance is illustrated by numerical examples in section 4. First, the implementation specifics are discussed in subsection 4.1, and then we present the results and their analysis. Finally, our conclusions and results are summarized in section 5. Further approximations which relates to the use of Green's functions are presented in the Appendix.

2 The Method

2.1 Definitions and Description

2.1.1 Mixed Implicit-Explicit Algorithms Throughout this paper we consider the parallel solution of parabolic PDEs, where parallelism is achieved by domain decomposition (DD). The spatial domain is decomposed into sub-domains and each processor is assigned the task of processing its sub-domain. Such decomposition reduces the problem into several sub-problems that are solved *independently* by each processor without explicit synchronization. This is done, first by determining the solution values on the sub-domains' boundaries by using an *asynchronous* explicit scheme, and then using within each sub-domain any known high-order implicit finite difference scheme. To allow each processor to compute independently of the others we do not impose restrictions on the relative computing rate. Hence, a processor may encounter at neighboring processor values that are associated with a time step different than its own. This is compensated by the asynchronous scheme.

Another characteristic of the sub-domains' boundary scheme we use, besides being explicit and asynchronous, is that no significant restriction is imposed on the time step. This is important because using an implicit scheme within a sub-domain guarantees that there is no stability restriction on the size of the time step. Thus, a restriction due to a boundary scheme could impose severe limitation. Moreover, complementary to the accuracy of the implicit scheme the sub-domains' boundary scheme should be of high-order. We derive such high-order, absolutely stable, explicit and asynchronous schemes that are based on Green's function.

We begin with presenting the synchronous algorithm since it resembles other DD methods in the sense that the time steps are synchronized. The discussion of the synchronous case will serve us as an introduction to the asynchronous algorithm which differs due to the removal of explicit synchronization. We note, that although the algorithms differ, the synchronous version of the sub-domains' boundaries scheme is a special case of the asynchronous one (see section 3).

In the synchronous case all processors compute the same time step simultaneously. A global variable, n, is the time steps counter. It is increased simultaneously at the end of each step. A global constant of time-step size, Δt, is set during initialization. And thus, the problem time, $t = n\Delta t$, is also global . Besides this global data we also require communication of values between neighboring processors. These values have to be associated with the *same* time step. This can be easily achieved by storing the values in a shared-memory (or else by message-passing). Moreover, simultaneous memory updates cannot occur since each processor updates a different set of values. Values at a neighboring processor are needed only on a read-only basis. Synchronization is achieved explicitly at the end of each simultaneous step by synchronization mechanisms (semaphore). Following is a skeleton of a general framework for synchronous computation:

ALGORITHM 1 (SYNCHRONOUS).

1. *Division of the domain into sub-domains and assigning each sub-domain to a different processor.*

2. *Each processor performs:*

   (a) *Computation of the sub-domains' boundaries using an explicit scheme, or the domain boundaries.*

   (b) *Computation of interior sub-domain points by an implicit scheme using the values previously communicated for sub-domains' boundaries.*

   (c) *Waiting for neighboring processors to complete this step (synchronization).*

3. *Transfer of values among processors (communication).*

4. *Increasing global time ($t = t + \Delta t$) and global time step counter ($n = n + 1$).*

5. *Repeating steps 2 to 4 until time or accuracy reaches the stopping criteria.*

In the asynchronous case each processor can advance at its own rate. Hence, there is no global time steps counter n but a local counter n_p where p is the sub-domain assigned to the processor p. Also, there is no global constant time step size, but an adaptive local step, that may be different for each sub-domain p and for each step n_p. It is denoted by $\Delta t_p^{n_p}$. The time is also local. We denote by $t_p^{n_p}$ the time at sub-domain p (performed on processor p) for step n_p. Following is a skeleton of a general framework for asynchronous computation: ALGORITHM 2 (ASYNCHRONOUS).

1. *Division of the domain into sub-domains and assigning each sub-domain to a different processor.*

2. Each processor performs:

(a) Transfer of the required values from neighboring processors (communication).

(b) Setting the local size of time step $\Delta t_p^{n_p}$.

(c) Computation of sub-domains' boundaries using an asynchronous explicit scheme, or the domain boundaries.

(d) Computation of interior sub-domain points by an implicit scheme, using the values just communicated, for sub-domains' boundaries.

(e) Increasing local time ($t_p^{n_p} = t_p^{n_p} + \Delta t_p^{n_p}$) *and local time step counter* ($n_p = n_p + 1$).

(f) Repeating step 2 until time or accuracy reaches the stopping criteria.

Step *(b)* of Algorithm 2 outlines an algorithm for the determination of the size of the local time steps $\Delta t_p^{n_p}$. The size of the time steps is not restricted by stability since our schemes are absolutely stable. It is restricted only by accuracy considerations. There are many ways to determine the size of which the simplest is to set a fixed size. Our proposed method is based on *adaptive* time steps which are calculated based on the relative positions of the neighboring grid points.

In our implementation, we chose to set a bound on the maximum size which is allowed. This way, we implicitly prevent an increasing gap between the time step calculated by a "fast" processor and those of its neighbors. Therefore, our method has a *time-stabilizing* property [1]: DEFINITION 2.1. *time-stabilizing property: The derivatives of the time with respect to the spatial coordinates are bounded as the number of time steps increases, and vanish as the spatial mesh-size approaches zero.*

Table 2 summarizes our decision table for the choice of $\Delta t_p^{n_p}$ (or the corresponding $t_p^{n_p}$). We consider all points on a sub-domain's boundary (e.g., in the 1-D case, both right and left sub-domain's boundary points are considered). All the points within a sub-domain are at the same time $t_i^{n_i} = t_p^{n_p}$ $\forall i$ where i is a point and p denotes the sub-domain. The time at the left and right neighboring sub-domains is denoted by $t_{p\pm1}^{n_p}$. We set a bound Δt_{max} on the size of the time steps so that $\Delta t_p^{n_p} \leq \Delta t_{max}$ $\forall p, n_p$.

The first line in each box of Table 2 defines one of the nine possible options for the 1-D case. The second line is our choice of $\Delta t_p^{n_p}$ (or the corresponding $t_p^{n_p}$). In the following we outline the strategy for the choice of $\Delta t_p^{n_p}$: ALGORITHM 3 (CHOICE OF STEP SIZE).

- *If the time at one of the neighboring sub-domains is greater set the step size to reach this time.*
- *If the time at all of the neighboring sub-domains is greater set the step size to reach the minimum of those time values.*
- *If the time at one neighboring sub-domain is less and the other is at the same time, or both are at the same time, set the step size to the maximum allowed.*
- *If the time at all of the neighboring sub-domains is less do not advance (i.e., set the step size to zero).*

This choice is made particularly for cases where there is a persistent difference in processing rate. Unlike that of [3], and similar to [1] our method provides accurate results even when there is a fixed delay among the processors i.e., when certain processors are always slower than the others. Had we used the simple strategy of always assigning a *constant* time-step (e.g., $\Delta t_p^{n_p} = \Delta t_{max}$), cases where one sub-domain is updated more often than others would result in an ever increasing gap between its time $t_p^{n_p}$ and those of its neighbors (as illustrated in Fig. (2)). This can eventually degrades the accuracy (in sub-section 3.1 it is established that the accuracy is a function of $|t_p - t_{p\pm1}|$). However, using *adaptive* time-step size according to Table (2) results in rather different profile of processors advances. As illustrated in Fig. (3), the time $t_p^{n_p+1}$ of the second processor is limited by the advance of its neighbors. Therefore, we have the second processor at most $2\Delta t_{max}$ ahead of his neighbors.

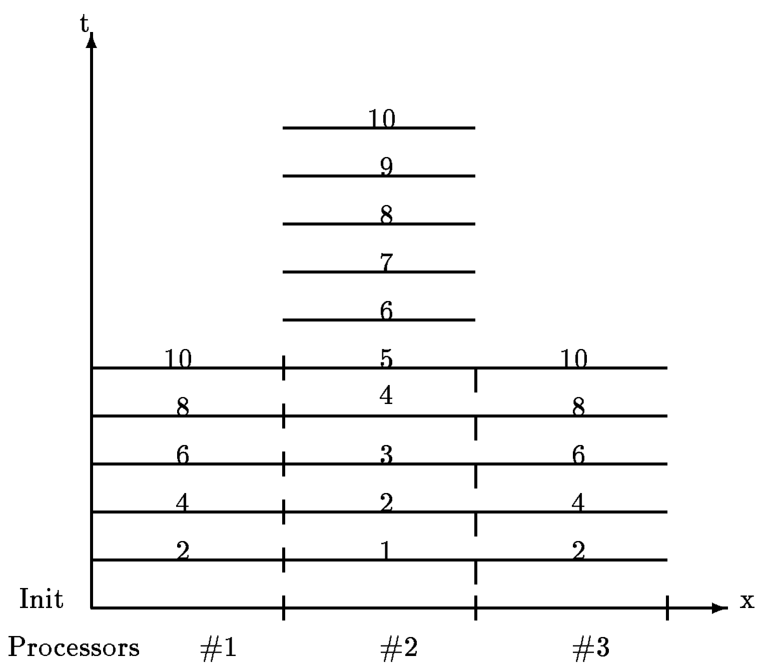

FIG. 2. : A profile of the processors advances with constant time step size $\Delta t_p^{n_p} = \Delta t_{max}$. The second processor is twice faster than its left and right neighbors. The numbers indicate "ticks" of an external clock.

$t_{p-1}^{n_p} < t_p^{n_p}$, $t_p^{n_p} > t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = 0$	$t_{p-1}^{n_p} = t_p^{n_p}$, $t_p^{n_p} > t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = \Delta t_{max}$	$t_{p-1}^{n_p} > t_p^{n_p}$, $t_p^{n_p} > t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = t_{p-1}^{n_p} - t_p^{n_p}$
$t_{p-1}^{n_p} < t_p^{n_p}$, $t_p^{n_p} = t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = \Delta t_{max}$	$t_{p-1}^{n_p} = t_p^{n_p}$, $t_p^{n_p} = t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = \Delta t_{max}$	$t_{p-1}^{n_p} > t_p^{n_p}$, $t_p^{n_p} = t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = t_{p-1}^{n_p} - t_p^{n_p}$
$t_{p-1}^{n_p} < t_p^{n_p}$, $t_p^{n_p} < t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = t_{p+1}^{n_p} - t_p^{n_p}$	$t_{p-1}^{n_p} = t_p^{n_p}$, $t_p^{n_p} < t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = t_{p+1}^{n_p} - t_p^{n_p}$	$t_{p-1}^{n_p} > t_p^{n_p}$, $t_p^{n_p} < t_{p+1}^{n_p}$ $\Delta t_p^{n_p} = \min\{t_{p-1}^{n_p}, t_{p+1}^{n_p}\} - t_p^{n_p}$

TABLE 2: Choice of $\Delta t_p^{n_p}$ (or the corresponding $t_p^{n_p}$) that provides a time-stabilizing property. Our choice is presented at the second line of each box according to the option presented in the first line.

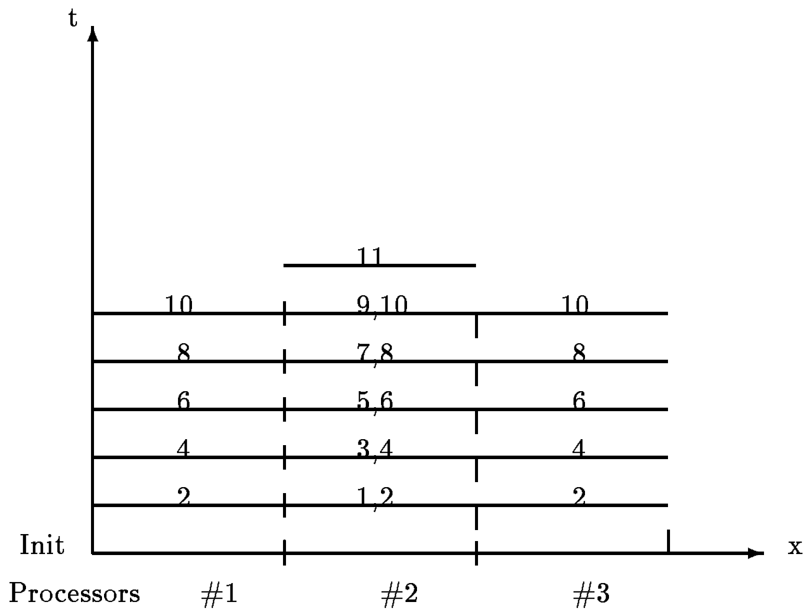

FIG. 3. : A profile of the processors advances with an adaptive time step size, $\Delta t_p^{n_p}$, chosen according to Table 1. The second processor is twice faster than its left and right neighbors. The numbers indicate "ticks" of an external clock.

2.1.2 A General Form for an Asynchronous-Explicit Scheme In the beginning of this discussion we establish the notation that will serve us throughout this paper. Before introducing the asynchronous case, it is worthwhile to consider the synchronous one. For the synchronous algorithm (Algorithm 1) as well as for the synchronous sub-domain's boundary scheme, the following notation for the *global* data is being used:

- i - The grid point at which this scheme is applied, which is at sub-domain boundary ($x_i = \xi$).
- n - A global variable of the time steps counter. It is increased simultaneously at the end of each step.
- Δt - Time discretization which is the size of the time-step. It is a global constant which is set during initialization.
- t - The problem time where $t = n\Delta t$.
- p - Processor number and correspondingly its assigned sub-domain.

In the asynchronous case each processor advances at its own rate. Hence, the global data is replaced by local data. Since values and time steps are uncoordinated we associate values with each local time step counter. This association is denoted by a superscript of the local counter. The following notation for the *local* data is being used:

n_i	-	A local time steps counter at grid point i on a sub-domain's boundary. It is increased locally at the end of each step.
$t_i^{n_i}$	-	The problem time at step n_i of grid point i on a sub-domain's boundary where $t_i^{n_i} = t_i^0 + \sum_{m=0}^{n_i-1} \Delta t_i^m$.
$t_{i\pm1}^{n_i}$	-	The problem time at the left and right neighbors of sub-domain's boundary point i associated with step n_i of the sub-domain's boundary point i.
$\Delta t_i^{n_i}$	-	An adaptive local time step size set for each step n_i at each grid point i on a sub-domain's boundary $(t_i^{n_i+1} - t_i^{n_i} = \Delta t_i^{n_i})$.
$\Delta t_{i\pm1}^{n_i}$	-	Size of neighboring local time steps associated with step n_i of point i on the sub-domain's boundary.
$\Delta t_{i\pm1}^{n_{i\pm1}}$	-	Size of neighboring local time steps associated with their local steps counters $n_{i\pm1}$.
$u_i(t_i^{n_i})$	-	The value of u at point i and time $t_i^{n_i}$.
$a_i(t_i^{n_i})$	-	The value of the coefficient a of Eq. (2.1) at point i and time $t_i^{n_i}$.

Throughout this paper we demonstrate our method on a sample problem. For simplicity, which does not limit the generality of the claims, we chose the multi-dimensional heat equation as the model problem:

$$\frac{\partial u}{\partial t} - \sum_{l=1}^{d} \frac{\partial^2 a(u)}{\partial {x_l}^2} = F(\vec{x}, t) \tag{2.1}$$

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

With the above notation the general form of 1-D explicit two level *asynchronous* finite difference scheme that approximates Eq. (2.1) is:

$$u_i(t_i^{n_i+1}) = \sum_{m=-M}^{M} \phi_{i,m}^{n_i} u_{i+m}(t_{i+m}^{n_i}) + \Delta t_i^{n_i} F_i(t_i^{n_i}) \tag{2.2}$$

where

$\phi_{i,m}^{n_i}$	-	The coefficient of the sub-domain's boundary scheme while calculating the m-th neighbor to i that is associated with time step n_i.
M	-	The number of grid points to the left, or to the right, of a grid point which participate in the scheme.

In this general form a different time is associated with each grid point. However, since we are using this scheme only at sub-domains' boundaries while assuming that there is a sufficient number of points per sub-domain then we can be more specific. In this case, the time steps are determined per sub-domain in its associated processor rather than per each single point. This way the same values are associated with all points at the same sub-domain:

n_p	-	Sub-domain p local time steps counter. It is processed by processor p and is increased locally at the end of each step.
$t_p^{n_p}$	-	The problem time at step n_p at sub-domain p.
$\Delta t_p^{n_p}$	-	An adaptive local time step size set for each step n_p at sub-domain p.

Thus, there are only two time values to consider: that of the sub-domain to which belongs the boundary point $t_p^{n_p}$ and that of its neighboring sub-domain $t_{p\pm1}^{n_p}$. We calculate the *right* sub-domain boundary when $t_p^{n_p} > t_{p+1}^{n_p}$ (e.g., according to Table 2). Otherwise, we can use the same sub-domain's boundary values that have already been computed for the neighboring sub-domain. This is illustrated in the left of Fig. 5 ($t_p^{n_p} = t_L > t_R = t_{p+1}^{n_p}$). The scheme becomes:

$$u_i(t_p^{n_p+1}) = \sum_{m=0}^{M} \left[\phi_{i,-m}^{n_p} u_{i-m}(t_p^{n_p}) + \phi_{i,m}^{n_p} u_{i+m}(t_{p+1}^{n_p})\right] + \Delta t_p^{n_p} F_i(t_p^{n_p})$$

Whereas, the scheme for calculating the *left* sub-domain boundary when $t_{p-1}^{n_p} < t_p^{n_p}$ (see Table 2) as illustrated in the right of Fig. 5 ($t_{p-1}^{n_p} = t_L < t_R = t_p^{n_p}$) becomes:

$$u_i(t_p^{n_p+1}) = \sum_{m=0}^{M} \left[\phi_{i,-m}^{n_p} u_{i-m}(t_{p-1}^{n_p}) + \phi_{i,m}^{n_p} u_{i+m}(t_p^{n_p})\right] + \Delta t_p^{n_p} F_i(t_p^{n_p})$$

2.2 Numerical Analysis of the Proposed Method and Schemes

2.2.1 Numerical Analysis of the Mixed Implicit-Explicit Method

In this section we discuss the general case in which any asynchronous explicit scheme of the form (2.2) is used on sub-domains' boundaries, whereas any implicit scheme is used within each sub-domain. Any such implicit finite difference scheme for the solution of Eq. (2.1) over the entire problem domain can be written in the following matrix form

$$A\vec{u}^{n+1} = B\vec{u}^n + \Delta t F^n \tag{2.3}$$

where $\vec{u}^n$ is a vector of non-boundary values associated with the n^{th} time step and A and B are matrices of the corresponding coefficients of the implicit scheme. Also, for non-zero initial values a vector of the corresponding values is added to right hand side of Eq. (2.3) which is dependent only on i.

Hence, we follow Algorithms 1 and 2 where first, we apply the asynchronous explicit Eq. (2.2) on sub-domains' boundaries and then we separately apply the implicit Eq. (2.3) within each sub-domain.

For the purpose of this numerical analysis we assume a *global* view. We emphasize that, such a global view cannot serve for the actual implementation since maintaining a global counter requires global synchronization. Suppose that the solution by the proposed asynchronous method had been completely determined up to a certain time step ($t = T$). We then consider the fixed interval $0 \leq t \leq T$ with its sequence of finite difference solutions. We define an external *counter* ("clock") of equal increments along the t-axis ("tick") which is denoted by k. The size of an increment, denoted by Δ_k, is set equal to the smallest size of time step ($\Delta t_i^{n_i}$, $\forall i$) in the interval under consideration. In the *synchronous* case $n = k$

and $\Delta t = \Delta_k$. Thus, the synchronous case (Algorithm 1) is a special case of the proceeding analysis.

This can be formulated in the following matrix form,

$$\hat{A}u^{k+1} = \hat{B}u^k + \hat{G}u^k + \Delta_k F^k \tag{2.4}$$

where, $\hat{A}$ and $\hat{B}$ are matrices based on the matrices A and B of Eq. (2.3) with some rows replaced by the unit vector. These rows correspond to the sub-domains' boundaries which are not computed according to this scheme. $\hat{G}$ is based on the zero matrix where the rows that correspond to the positions of sub-domains' boundaries (and to those that are unit vector in $\hat{A}$ and $\hat{B}$) are composed of the coefficients ($\phi_{i,m}^{n_i}$) of the asynchronous explicit scheme Eq. (2.2).

For example, suppose we use within the sub-domains the implicit Backward-Euler scheme and on sub-domains' boundaries the explicit Corrected Asynchronous (CA) scheme ([1], [3]):

$$u_i(t_i^{n_i+1}) = \left(1 - \frac{2\Delta t_i^{n_i}}{\Delta x + \delta^2 t_i^{n_i}}\right) u_i(t_i^{n_i}) + \frac{\Delta t_i^{n_i}}{\Delta x + \delta^2 t_i^{n_i}} \left[u_{i-1}(t_{i-1}^{n_i}) + u_{i+1}(t_{i+1}^{n_i})\right]$$

where, $\delta^2 t_i^{n_i} = t_{i-1}^{n_i} - 2t_i^{n_i} + t_{i+1}^{n_i}$ and for simplicity we assume that $\Delta t_i^{n_i}$ is constant.

We divide the domain into three sub-domains of four points, with the third and sixth points overlapped between their neighboring sub-domains. For the purpose of this example we assume each sub-domain is processed at a different rate. A profile of the rate of computation relative to the external global counter k is illustrated in Fig. 4. The circles mark the computation of a point. Boxes mark the bypass of the computation of a sub-domain boundary point by taking a value already computed for the neighboring sub-domain. These values are pointed by the arrows. The first sub-domain is computed randomly. The second sub-domain is updated every increment of k. The third sub-domain is processed every other increment of k. We emphasize that in the asynchronous case there is a fundamental difference between the external global counter k and the problem time t. Unlike the common synchronous case where there is no distinguish between the two.

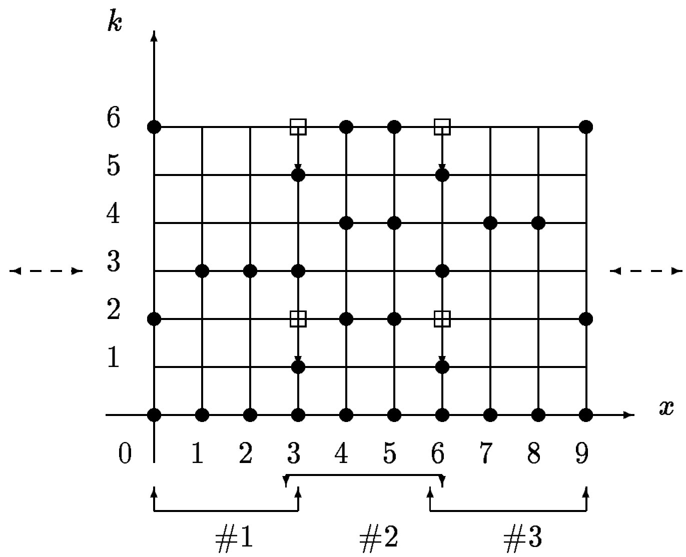

FIG. 4. : A profile of the rate of computation of each sub-domain relative to the external global counter k.

Taking a snapshot at $k = 3$ in Fig. 4 the matrices for Eq. (2.4) are:

$$\hat{A} = \begin{pmatrix} 1 & 0 & & & & & & & & \\ -\xi & 1+2\xi & -\xi & & & & & & & \\ 0 & -\xi & 1+2\xi & -\xi & & & & & & \\ 0 & 0 & 0 & 1 & 0 & & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where, $\xi = \frac{\Delta t}{\Delta x}$

$$\hat{B} = \begin{pmatrix} 1 & 0 & & & & & & & & \\ 0 & 1 & 0 & & & & & & & \\ 0 & 0 & 1 & 0 & & & & & & \\ 0 & 0 & 0 & 1 & 0 & & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\hat{G} = \begin{pmatrix} 0 &&&&&&&& \\ 0 & 0 &&&&&&& \\ 0 & 0 & 0 &&&&&& \\ 0 & 0 & 0 & \frac{\Delta t}{\Delta x+\delta^2 t_3^2} & 1-\frac{2\Delta t}{\Delta x+\delta^2 t_3^2} & \frac{\Delta t}{\Delta x+\delta^2 t_3^2} &&& \\ 0 & 0 & 0 & 0 & 0 &&&& \\ 0 & 0 & 0 & 0 & 0 & 0 &&& \\ 0 & 0 & 0 & 0 & 0 & 0\frac{\Delta t}{\Delta x+\delta^2 t_6^2} & 1-\frac{2\Delta t}{\Delta x+\delta^2 t_6^2} & \frac{2\Delta t}{\Delta x+\delta^2 t_6^2} & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 && \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \end{pmatrix}$$

where,

$$\delta^2 t_3^2 = t_2^2 - 2t_3^2 + t_4^2 = 0 - 2\Delta t + \Delta t = -\Delta t \quad , \quad \delta^2 t_6^2 = t_5^2 - 2t_6^2 + t_7^2 = \Delta t - 2\Delta t + 0 = -\Delta t$$

Following are the values at each point. The subscript denotes the position of the point and in the parentheses is its problem time.

$$u^3 = \begin{pmatrix} u_0(\Delta t) \\ u_1(\Delta t) \\ u_2(\Delta t) \\ u_3(2\Delta t) \\ u_4(\Delta t) \\ u_5(\Delta t) \\ u_6(2\Delta t) \\ u_7(0) \\ u_8(0) \\ u_9(\Delta t) \end{pmatrix} \quad , \quad u^2 = \begin{pmatrix} u_0(\Delta t) \\ u_1(0) \\ u_2(0) \\ u_3(\Delta t) \\ u_4(\Delta t) \\ u_5(\Delta t) \\ u_6(\Delta t) \\ u_7(0) \\ u_8(0) \\ u_9(\Delta t) \end{pmatrix}$$

LEMMA 2.1 (STABILITY). *Algorithm 2 with implicit scheme as of Eq. (2.3) inside each sub-domain and asynchronous explicit scheme as of Eq. (2.2) on sub-domains' boundaries which are both stable results in a stable method.*

Proof. If the implicit scheme Eq. (2.3) is stable then the corresponding matrix A is non-singular. $\hat{A}$ is a block matrix whose blocks are composed of the same elements as of matrix A. It follows that $\hat{A}$ is non-singular. Thus, we can rewrite Eq. (2.4) as

$$u^{k+1} = \hat{A}^{-1}\hat{B}u^k + \hat{A}^{-1}\hat{G}u^k + \Delta_k F^k \tag{2.5}$$

Since, by multiplying the corresponding blocks, we have

$$\hat{A}^{-1}\hat{G} = \begin{pmatrix} A^{-1} && \\ 0 & 1 & 0 \\ && A^{-1} \end{pmatrix} \begin{pmatrix} 0 && \\ \dots & \phi_{i,m}^{n_i} & \dots \\ && 0 \end{pmatrix} = \hat{G}$$

Then,

$$u^{k+1} = \left(\hat{A}^{-1}\hat{B} + \hat{G}\right) u^k + \Delta_k F^k \tag{2.6}$$

Upon repeated application of Eq. (2.6) we obtain

$$u^{k+1} = \left(\hat{A}^{-1}\hat{B} + \hat{G}\right)^{k+1} u^0 + (\Delta_k)^{k+1} F^0$$

where, u^0 is the vector of initial values.

Repeating the same process for increasing intervals $0 \leq t \leq T$, we conclude that as $T \to \infty$ errors will not grow in magnitude and thus stability is assured if all eigenvalues of $(\hat{A}^{-1}\hat{B} + \hat{G})$ are, in absolute value, less than or equal to one. Evidently, this is guaranteed by our assumption on the stability of Eqs. (2.3) and (2.2) and by our construction of $\hat{A}^{-1}$ and $\hat{B}$. Thus, the method is stable. □

LEMMA 2.2 (CONSISTENCY). *Algorithm 2 with implicit scheme as of Eq. (2.3) inside each sub-domain and asynchronous explicit scheme as of Eq. (2.2) on sub-domains' boundaries which are both consistent with Eq. (2.1) results in a consistent method.*

Proof. Our computation is done in two phases. First, we determine the values at the sub-domain's boundaries by applying asynchronous explicit scheme as of Eq. (2.2). If this scheme is consistent then the values produced by this phase are consistent. Second, we apply an implicit scheme as of Eq. (2.3) inside each sub-domain. If this scheme is consistent so is any application of it particularity those inside each sub-domain (providing that the values on sub-domains' boundaries are consistent). Therefore, this two-phases method is consistent with Eq. (2.1). □

2.2.2 Numerical Analysis of the Asynchronous-Explicit Scheme The numerical properties of the known implicit schemes are already well established elsewhere (e.g. [16]). We now present the conditions that any *asynchronous* explicit finite difference scheme should meet. We note that the conditions for the *synchronous* version are obtained as a special case with $t^{n_i}_{i+m} - t^{n_i}_i = 0$ where $-M \leq m \leq M$.

LEMMA 2.3 (CONSISTENCY). *If $\phi^{n_i}_{i,m}$, $-M \leq m \leq M$ are twice continuously differentiable with respect to Δx and if*

$$
(2.7)\qquad
\begin{array}{ll}
\lim_{\Delta x \to 0} \frac{1}{\Delta t^{n_i}_i}\left[\sum_{m=-M}^{M} \phi^{n_i}_{i,m} - 1\right] & \to 0 \\
\lim_{\Delta x \to 0} \frac{\Delta x}{\Delta t^{n_i}_i}\sum_{m=-M}^{M} m\phi^{n_i}_{i,m} & \to 0 \\
\lim_{\Delta x \to 0} \frac{\Delta x^2}{2\Delta t}\sum_{m=-M}^{M} m^2\phi^{n_i}_{i,m} & \to a_i(t^{n_i}_i) \\
\lim_{\Delta x \to 0} \frac{\Delta t^{n_i}_i - \sum_{m=-M}^{M}(t^{n_i}_{i+m} - t^{n_i}_i)\phi^{n_i}_{i,m}}{\Delta t^{n_i}_i} & \to 1 \\
\lim_{\Delta x \to 0} \frac{\Delta x}{\Delta t^{n_i}_i}\sum_{m=-M}^{M}(t^{n_i}_{i+m} - t^{n_i}_i)m\phi^{n_i}_{i,m} & \to 0 \\
\lim_{\Delta x \to 0} \frac{\Delta x^2}{2\Delta t}\sum_{m=-M}^{M}(t^{n_i}_{i+m} - t^{n_i}_i)m^2\phi^{n_i}_{i,m} & \to 0
\end{array}
$$

then the difference scheme (2.2) is consistent with the differential equation (2.1) with respect to any sufficiently smooth solution $u(x,t)$.

Proof. Assume that $u(x,t)$ has a continuous first derivative with respect to t, and a continuous second derivative with respect to x. Then, by expending $u(i\Delta x, n\Delta t)$ in a Taylor series, and substituting in the difference scheme Eq. (2.2) we get,

$$
(2.8)\qquad
\begin{aligned}
u_i(t^{n_i}_i) + (\Delta t^{n_i}_i)(\tfrac{\partial u}{\partial t})_i(t^{n_i}_i) + O(\Delta t^{n_i}_i)^2 \;&= \sum_{m=-M}^{M} \phi^{n_i}_{i,m}\Big\{u_i(t^{n_i}_i) + m(\Delta x)(\tfrac{\partial u}{\partial x})_i(t^{n_i}_i) \\
&\quad + \tfrac{1}{2}m^2(\Delta x)^2(\tfrac{\partial^2 u}{\partial x^2})_i(t^{n_i}_i) \\
&\quad + (t^{n_i}_{i+m} - t^{n_i}_i)\Big[(\tfrac{\partial u}{\partial t})_i(t^{n_i}_i) + m(\Delta x)(\tfrac{\partial^2 u}{\partial x \partial t})_i(t^{n_i}_i) \\
&\quad + \tfrac{1}{2}m^2(\Delta x)^2(\tfrac{\partial^3 u}{\partial^2 x \partial t})_i(t^{n_i}_i)\Big]\Big\} \\
&\quad + \Delta t^{n_i}_i F_i(t^{n_i}_i) + O(t^{n_i}_{i+m} - t^{n_i}_i)^2
\end{aligned}
$$

Therefore,

$$\begin{aligned}\left[\Delta t_i^{n_i} - \textstyle\sum_{m=-M}^{M}(t_{i+m}^{n_i} - t_i^{n_i})\phi_{i,m}^{n_i}\right](\tfrac{\partial u}{\partial t})_i(t_i^{n_i}) &= \tfrac{1}{\Delta t_i^{n_i}}[\textstyle\sum_{m=-M}^{M}\phi_{i,m}^{n_i} - 1]u_i(t_i^{n_i}) \\ &+\tfrac{\Delta x}{\Delta t_i^{n_i}}\textstyle\sum_{m=-M}^{M} m\phi_{i,m}^{n_i}(\tfrac{\partial u}{\partial x})_i(t_i^{n_i}) \\ &+\tfrac{\Delta x^2}{2\Delta t_i^{n_i}}\textstyle\sum_{m=-M}^{M} m^2\phi_{i,m}^{n_i}(\tfrac{\partial^2 u}{\partial x^2})_i(t_i^{n_i}) \\ &+\tfrac{\Delta x}{\Delta t_i^{n_i}}\textstyle\sum_{m=-M}^{M}(t_{i+m}^{n_i} - t_i^{n_i})m\phi_{i,m}^{n_i}(\tfrac{\partial^2 u}{\partial x\partial t})_i(t_i^{n_i}) \\ &+\tfrac{\Delta x^2}{2\Delta t_i^{n_i}}\textstyle\sum_{m=-M}^{M}(t_{i+m}^{n_i} - t_i^{n_i})m^2\phi_{i,m}^{n_i}(\tfrac{\partial^3 u}{\partial^2 x\partial t})_i(t_i^{n_i}) \\ &+F_i(t_i^{n_i})\end{aligned}$$

Equating this approximation with Eq. (2.1) we obtain that Eq. (2.2) is consistent with Eq. (2.1) with the above conditions (2.7). $\square$

In the synchronous case we substitute $t_{i+m} - t_i = 0$ in conditions (2.7) and obtain a similar conditions to those in [12].

The following stability condition is an extension to the one proved in [1].

LEMMA 2.4 (STABILITY). *A sufficient stability condition for Eq. (2.2) is*

$$\sum_{m=-M}^{M} |\phi_{i,m}^{n_i}| \le 1 \qquad \forall i, n_i \tag{2.9}$$

Proof. Assume that some perturbations occured in the initial values of all grid points. We wish to examine the effect of this perturbation on the calculated values on the grid points. Let $\varepsilon_i^{n_i} = u(x_i, t^{n_i}) - u_i^{n_i}$ where $u(x,t)$ is the exact solution and $u_i^{n_i}$ is the perturbate one. Then,

$$\varepsilon_i(t_i^{n_i+1}) = \sum_{m=-M}^{M} \phi_{i,m}^{n_i}\varepsilon_{i+m}(t_{i+m}^{n_i}) + \Delta t_i^{n_i} F_i(t_i^{n_i})$$

By Eq. (2.9) the sum of the absolute values of the coefficients of $\varepsilon_{i+m}^{n_i}$ (where $-M \le m \le M$) is bounded by one. Thus,

$$|\varepsilon_i(t_i^{n_i+1})| \le \max_{\varepsilon_k}\left\{|\varepsilon_{i-m}(t_{i-m}^{n_i})|, \ldots, |\varepsilon_{i+m}(t_{i+m}^{n_i})|\right\}\sum_{m=-M}^{M}\phi_{i,m}^{n_i} \qquad , \quad k = i-m, \ldots, i+m$$

This equation is satisfied whenever a value at sub-domain's boundary is calculated. Therefore, we can apply the last inequality recursively, following the "parents" of each grid point which maximizes $|\varepsilon_i(t_i^{n_i})|$, until the initial values are reached. This may be viewed as going down a tree along the reversed track of the calculation track. Then, we get

$$|\varepsilon_i(t_i^{n_i})| \le \max_k |\varepsilon_k(t_k^{0_k})|$$

which holds for all i, n_i. Hence, the scheme of Eq. (2.2) is stable. $\square$

We note that when $\forall i, n_i\ \phi_{i,m}^{n_i} \ge 0$ (where $-M \le m \le M$) the above stability condition is resulted from the consistency condition

$$\lim_{\Delta x \to 0}\frac{1}{\Delta t_i^{n_i}}\left[\sum_{m=-M}^{M}\phi_{i,m}^{n_i} - 1\right] \to 0$$

We consider now the convergence of (2.2) assuming that the requirements of Lemmas 2.3 and 2.4 are satisfied. We note that the *meaning* of convergence in the case where the time-steps at each sub-domain are determined *adaptively* (which can be viewed as randomly) is discussed in [4].

LEMMA 2.5 (CONVERGENCE). *The method described by Algorithms 2 and 3 while using the asynchronous explicit scheme (2.2) and the implicit scheme (2.3) converges to Eq. (2.1).*
Proof. In the case described by Algorithm 2, our choice of $\Delta t_i^{n_i}$ (see Algorithm 3 and Table 2) guarantees that we may apply Algorithm 3 recursively, until the initial time $t_i^{0_i}$ is reached so that it can not be applied repeatedly any more. We then obtain that

$$t_i^{n_i} = B_i^{n_i} \Delta t_{max} \tag{2.10}$$

where $B_i^{n_i}$ is a constant which depends upon the "roots-trace" of $t_i^{n_i}$ as determined by the recursive of Algorithm 3. The value $B_i^{n_i}$ is independent of Δx. Thus, as by Lax theorem, stability and consistency assure convergence. $\square$

3 Sub-domain Boundary Approximation

3.1 Derivation of Asynchronous-Explicit Approximations Based on Green's Functions

In the former discussion we described the general form of an asynchronous-explicit finite difference scheme to be applied at sub-domains' boundaries. In this section we present a general method for derivation of such a scheme based on Green's functions. To simplify our presentation of Green's functions, we will adopt here a different notation than the one formerly used for finite difference schemes:

$\xi = x_i$	-	The grid point at sub-domain's boundary at which the asynchronous scheme is applied.
M	-	The number of grid points to the left, or to the right, of grid point i, which are transfered between neighboring processors.
$x_L = x_{i-M}$	-	The farthest grid point to the left of ξ that is used for calculating the next step.
$x_R = x_{i+M}$	-	The farthest grid point to the right of ξ that is used for calculating the next step.
$\tau = t_p^{n_p+1}$	-	The time at sub-domain p boundary grid point ξ associated with time step $n_p + 1$. It is determined based on the relative positions of neighboring sub-domains (e.g., Algorithm 3 and Table 2).
$t_L = t_{p-1}^{n_p}$	-	The time at the sub-domain to the left of sub-domain p associated with time step n_p.
$t_R = t_{p+1}^{n_p}$	-	The time at the sub-domain to the right of sub-domain p associated with time step n_p.

Thus, the value calculated at each step for the sub-domain boundary is

$$u(\xi, \tau) = u(x_i, t_p^{n_p+1}) = u(i\Delta x, t_p^0 + \sum_{m=0}^{n_p} \Delta t_p^m)$$

This notation will serve us only for discussing the derivation of the scheme. Later, in section 3.2.1, we present the same schemes using our earlier notation.

3.1.1 1-D Case For the sake of clarity we first discuss the one-dimensional case. In this case we wish to solve the one-dimensional constant coefficients equation of (2.1) which takes the form

$$L(u) = F(x,t) \tag{3.11}$$

where

$$L(u) = \frac{\partial u}{\partial t} - a^2 \frac{\partial^2 u}{\partial x^2} \tag{3.12}$$

Suppose that $x = \xi$ is the boundary between two adjacent sub-domains and the current time at the left and right sub-domains are $t = t_L$ and $t = t_R$, respectively (see Fig. 5).

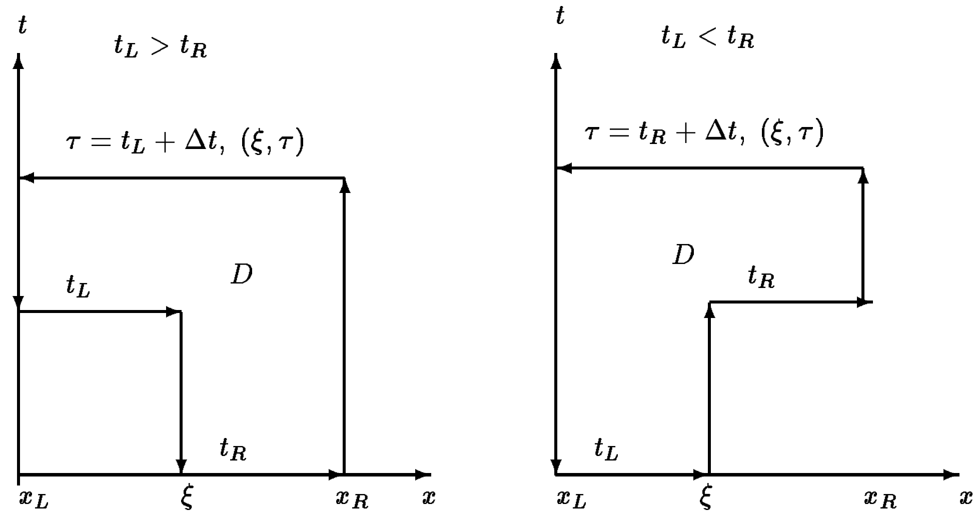

FIG. 5. : 1-D Asynchronous case: $x = \xi$ is a boundary between two adjacent sub-domains and $x = x_L, x_R$ are some inner grid points of the left and right sub-domains, respectively.

In order to update the inner grid points of the faster processor we would need to know the value of the solution on the boundary, $u(\xi, \tau)$, where $\tau = t_L + \Delta t$ if $t_L > t_R$ and $\tau = t_R + \Delta t$ if $t_R > t_L$. In this regard, we note that the adjoint operator of Eq. (3.12) is given by

$$M(v) = -\frac{\partial v}{\partial t} - a^2 \frac{\partial^2 v}{\partial x^2} \tag{3.13}$$

Introducing the fundamental solution:

$$v = \frac{1}{2a\sqrt{\pi(\tau - t)}} e^{-\frac{(x-\xi)^2}{4a^2(\tau - t)}} \tag{3.14}$$

where ξ and $\tau > t$ are as defined above. It can be easily verified that for $t < \tau$

$$vL(u) - uM(v) = \frac{\partial X}{\partial x} + \frac{\partial T}{\partial t} = vF(x,t) \tag{3.15}$$

where $T = uv$ and $X = -a^2(vu_x - uv_x)$.

Then, considering the region D with its boundary curve c in the x-t plane and integrating over Eq. (3.15), with v as in Eq. (3.14) and u as in Eq. (3.11) we get by Gauss' theorem that (see Fig. 5)

$$\begin{aligned} \textstyle\int_D [vL(u) - uM(v)] d\sigma &= \textstyle\int_D (\frac{\partial X}{\partial x} + \frac{\partial T}{\partial t}) d\sigma \\ &= \textstyle\int_C [X \cos(n,x) + T \cos(n,t)] ds \end{aligned}$$

where, $d\sigma$ is an area element of D,ds is a line element of C, $\cos(n,x)$ and $\cos(n,t)$ denote the corresponding normal derivative, and the orientation is positive (counter-clockwise).

$$
\begin{aligned}
\int_D vF(x,t)dxdt &= \int_{t=t_L} uvdx + \int_{t=t_R} uvdx + \int_{t=\tau} uvdx \\
&- \int_{x=x_L} a^2(v\tfrac{\partial u}{\partial n} - u\tfrac{\partial v}{\partial n})dt \\
&- \int_{x=\xi} a^2(v\tfrac{\partial u}{\partial n} - u\tfrac{\partial v}{\partial n})dt \\
&- \int_{x=x_R} a^2(v\tfrac{\partial u}{\partial n} - u\tfrac{\partial v}{\partial n})dt
\end{aligned}
\tag{3.16}
$$

where the integration on the right hand side is in the counter-clockwise direction and $\frac{\partial}{\partial n}$ represents the derivative in the normal direction along the vertical lines $x = x_L, x = \xi, x = x_R$ as the case may be.

Then, we have

$$\lim_{t\to\tau^-} \int_{t=\tau} uvdx = -u(\xi,\tau)$$

Since $v \to \delta(x-\xi)$ as $t \to \tau$, where δ is the Dirac delta function, in the sense that

$$
\begin{aligned}
\lim_{t\to\tau^-} \int_{-\infty}^{\infty} vf(x)dx &= \lim_{t\to\tau^-} \int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi a^2(\tau-t)}} e^{-\frac{(x-\xi)^2}{4a^2(\tau-t)}} f(x)dx \\
&= \lim_{r\to\infty} \int_{-\infty}^{\infty} \frac{r}{\sqrt{\pi}} e^{-r^2(x-\xi)^2} f(x)dx \\
&= \lim_{r\to\infty} \int_{-\infty}^{\infty} \delta_r(x-\xi)f(x)dx \\
&= \int_{-\infty}^{\infty} \delta(x-\xi)f(x)dx \\
&= f(\xi).
\end{aligned}
\tag{3.17}
$$

where f is some function of x, $v = \frac{1}{\sqrt{4\pi a^2(\tau-t)}} e^{-\frac{(x-\xi)^2}{4a^2(\tau-t)}}$ and τ^- denotes the approaching of τ from the left.

There is a difficulty in solving the integral on the left-hand side of Eq. (3.16) at $t = \tau$ which is a *singularity.* To overcome this problem we suggest the following modifications to the integral:

$$\int_{x_L}^{x_R} \int_{t_0}^{\tau} vF(x,t)dxdt = \int_{x_L}^{x_R} \int_{t_0}^{\tau} \frac{1}{\sqrt{4\pi a^2(\tau-t)}} e^{-\frac{(x-\xi)^2}{4a^2(\tau-t)}} F(x,t)dxdt \tag{3.18}$$

in which, $t = \tau$ is a singularity. To overcome this difficulty we first introduce the following notation. Let

$$\Phi(x,\xi,\tau,t) = \frac{1}{\sqrt{4\pi a^2(\tau-t)}} e^{-\frac{(x-\xi)^2}{4a^2(\tau-t)}} \tag{3.19}$$

then, substituting it in Eq. (3.18) and taking θ approaching τ from the left, we have

$$\int_{x_L}^{x_R} \lim_{\theta\to\tau^-} \int_{t_0}^{\theta} \Phi(x,\xi,\tau,t)F(x,t)dtdx \tag{3.20}$$

By using the Trapezoidal Rule, we obtain the following approximation

$$
\begin{aligned}
&\lim_{\theta\to\tau^-} \int_{x_L}^{x_R} (\tfrac{\theta-t_0}{2})[\Phi(x,\xi,\tau,t_0)F(x,t_0) + \Phi(x,\xi,\tau,\theta)F(x,\theta)]dx = \\
&= (\tfrac{\tau-t_0}{2}) \int_{x_L}^{x_R} \Phi(x,\xi,\tau,t_0)F(x,t_0)dx + \lim_{\theta\to\tau^-} (\tfrac{\tau-t_0}{2}) \int_{x_L}^{x_R} \frac{1}{\sqrt{4\pi a^2(\theta-t)}} e^{-\frac{(x-\xi)^2}{4a^2(\theta-t)}} F(x,\theta)dx
\end{aligned}
\tag{3.21}
$$

which is a δ-function as defined in Eq. (3.17). Thus, using the properties of a δ-function, we conclude that the second integral of Eq. (3.21) equals $F(x,\theta)$. Therefore, Eq. (3.21) becomes

$$\frac{\Delta t}{2}\int_{x_L}^{x_R}\frac{1}{\sqrt{4\pi a^2\Delta t}}e^{-\frac{(x-\xi)^2}{4a^2\Delta t}}F(x,t_0)dx+\frac{\Delta t}{2}F(\xi,\tau) \tag{3.22}$$

which is a second order spatial approximation to Eq. (3.18), whereas, the singularity has been eliminated.

Since $\frac{\partial v}{\partial x}(\xi,t) = 0\ \forall t$, then by using the Trapezoidal approximation for the time integration in Eq. (3.16) we obtain

$$\begin{aligned} u(\xi,\tau) &= \tfrac{2\tau-(t_L+t_R)}{4}F(\xi,\tau) \\ &+(\tfrac{\tau-t_L}{2})\textstyle\int_{x_L}^{\xi}(vF)(x,t_L)dx+(\tfrac{\tau-t_R}{2})\int_{\xi}^{x_R}(vF)(x,t_R)dx \\ &+\textstyle\int_{x_L}^{\xi}(uv)(x,t_L)dx+\int_{\xi}^{x_R}(uv)(x,t_R)dx \\ &-\left|\tfrac{t_R-t_L}{2}\right|\left[a^2(vu_x)(\xi,t_R)+a^2(vu_x)(\xi,t_L)\right] \\ &-\left(\tfrac{\tau-t_L}{2}\right)\left[a^2(vu_x-uv_x)(x_L,t_L)\right]-\left(\tfrac{\tau-t_R}{2}\right)\left[a^2(vu_x-uv_x)(x_R,t_R)\right] \\ &+O(\tau-t_L)^3+O(\tau-t_R)^3+O(t_L-t_R)^3 \end{aligned} \tag{3.23}$$

Note that higher order approximations can be obtained (see Appendix).

The synchronous version is a special case of the above, with $t_0 = t_L = t_R$ in Eqs. (3.14)-(3.23) (see Fig. 6). Thus, Eq. 3.23 becomes

$$\begin{aligned} u(\xi,\tau) &= \tfrac{\Delta t}{2}F(\xi,\tau)+\tfrac{\Delta t}{2}\textstyle\int_{x_L}^{x_R}(vF)(x,t_0)dx \\ &+\textstyle\int_{x_L}^{x_R}(uv)(x,t_0)dx \\ &-\tfrac{\Delta t}{2}\{[a^2(vu_x-uv_x)(x_L,t_0)]+[a^2(vu_x-uv_x)(x_R,t_0)]\} \\ &+O(\Delta t)^3 \end{aligned} \tag{3.24}$$

3.1.2 Multi-dimensional Case In this section we extend our derivation of an asynchronous-explicit scheme to the multi-dimensional case. In the d-dimensional case:

$$L(u)=\frac{\partial u}{\partial t}-\sum_{l=1}^{d}a_l^2\frac{\partial^2 u}{\partial x_l^2} \tag{3.25}$$

and we seek a solution $u(\vec{x},t)$ of

$$L(u)=F(\vec{x},t) \tag{3.26}$$

where $\vec{x}=(x_1,\ldots,x_d)$.

The adjoint operator of Eq. (3.26) is

$$M(v)=-\frac{\partial v}{\partial t}-\sum_{l=1}^{d}a_l^2\frac{\partial^2 v}{\partial x_l^2} \tag{3.27}$$

and the solution of $M(v)=0$ is given by the Green's function

$$v(\vec{x},t)=\prod_{l=1}^{d}\frac{1}{2a_l\sqrt{\pi(\tau-t)}}e^{-\frac{(x_l-\xi_l)^2}{4a_l^2(\tau-t)}} \tag{3.28}$$

for any $\vec{\xi}=(\xi_1,\ldots,\xi_d)$ and $\tau>t$. It can be shown that

$$vL(u)-uM(v)=\frac{\partial}{\partial t}(uv)+\sum_{l=1}^{d}a_l^2\frac{\partial X_l}{\partial x_l}$$

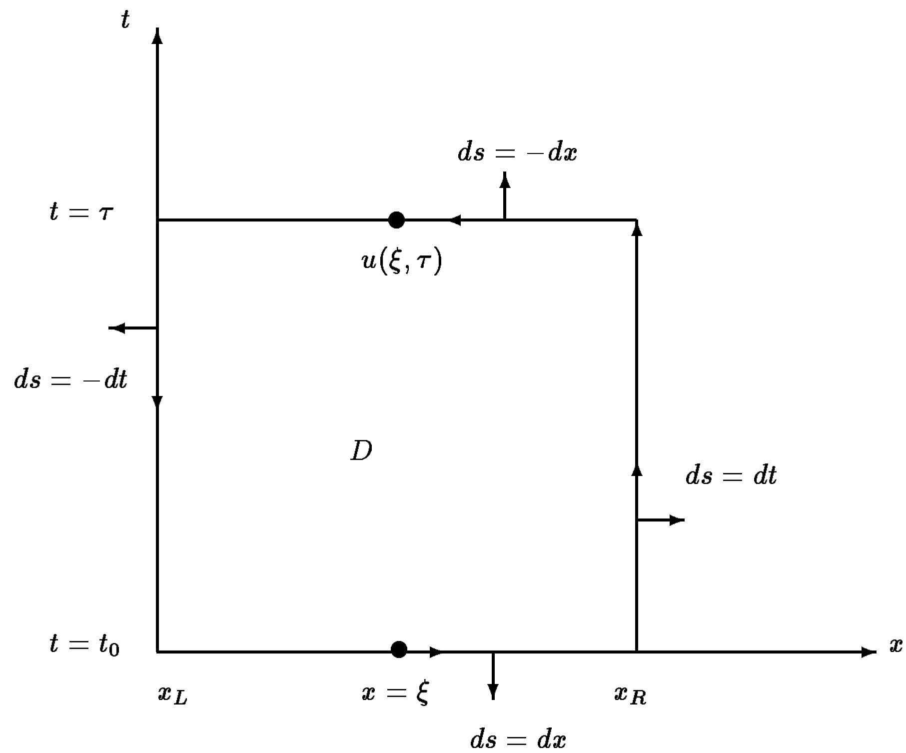

FIG. 6. : 1-D Synchronous case: $x = \xi$ is a boundary between two adjacent sub-domains.

where $X_l = -a_l^2(v\frac{\partial u}{\partial x_l} - u\frac{\partial v}{\partial x_l})$.

Without loss of generality we assume that the domain is decomposed along the x_1 axis. We choose ξ_l, $1 \le l \le d$ in the above equations such that $(\xi_1, \ldots, \xi_d)$ is a spatial intersection grid point of two adjacent processors' sub-domains. Then by the Divergence Theorem we have:

$$\int_W [vL(u) - uM(v)]\,dw = -\int_{t=t_L} uv ds - \int_{t=t_R} uv ds + \int_{t=\tau} uv ds - \sum_{l=1}^{d} a_l^2 \int_S \left(v\frac{\partial u}{\partial x_l} - u\frac{\partial v}{\partial x_l}\right) n_l dt$$

(3.29)

In Eq. (3.29),

$$W = W_1 \bigcup W_2$$

where

$$W_1 = \{(\vec{x}, t):\ \ x_L \le x_1 \le \xi,\ \ 0 \le x_j \le 1 \ \text{ for } \ 2 \le \mathrm{j} \le \mathrm{d},\ \ \mathrm{t_L} \le \mathrm{t} \le \tau\},$$

$$W_2 = \{(\vec{x}, t):\ \ \xi \le x \le x_R,\ \ 0 \le x_j \le 1 \ \text{ for } \ 2 \le \mathrm{j} \le \mathrm{d},\ \ \mathrm{t_R} \le \mathrm{t} \le \tau\},$$

$x_1 = x_L$ $(x_1 = x_R)$ is some inner grid point along x_1 axis of the left (or correspondingly, right) sub-domain, $x_1 = \xi_1$ is the boundary between the adjacent processors' sub-domains, $t = t_L$ (or correspondingly, $t = t_R$) is the time of the left (right) sub-domain, S is the surrounding surface when the surfaces $t = t_L$, $t = t_R$ and $t = \tau$ are excluded and n_l is the surface-normal component in the $l - th$ axis direction (see Fig. 7).

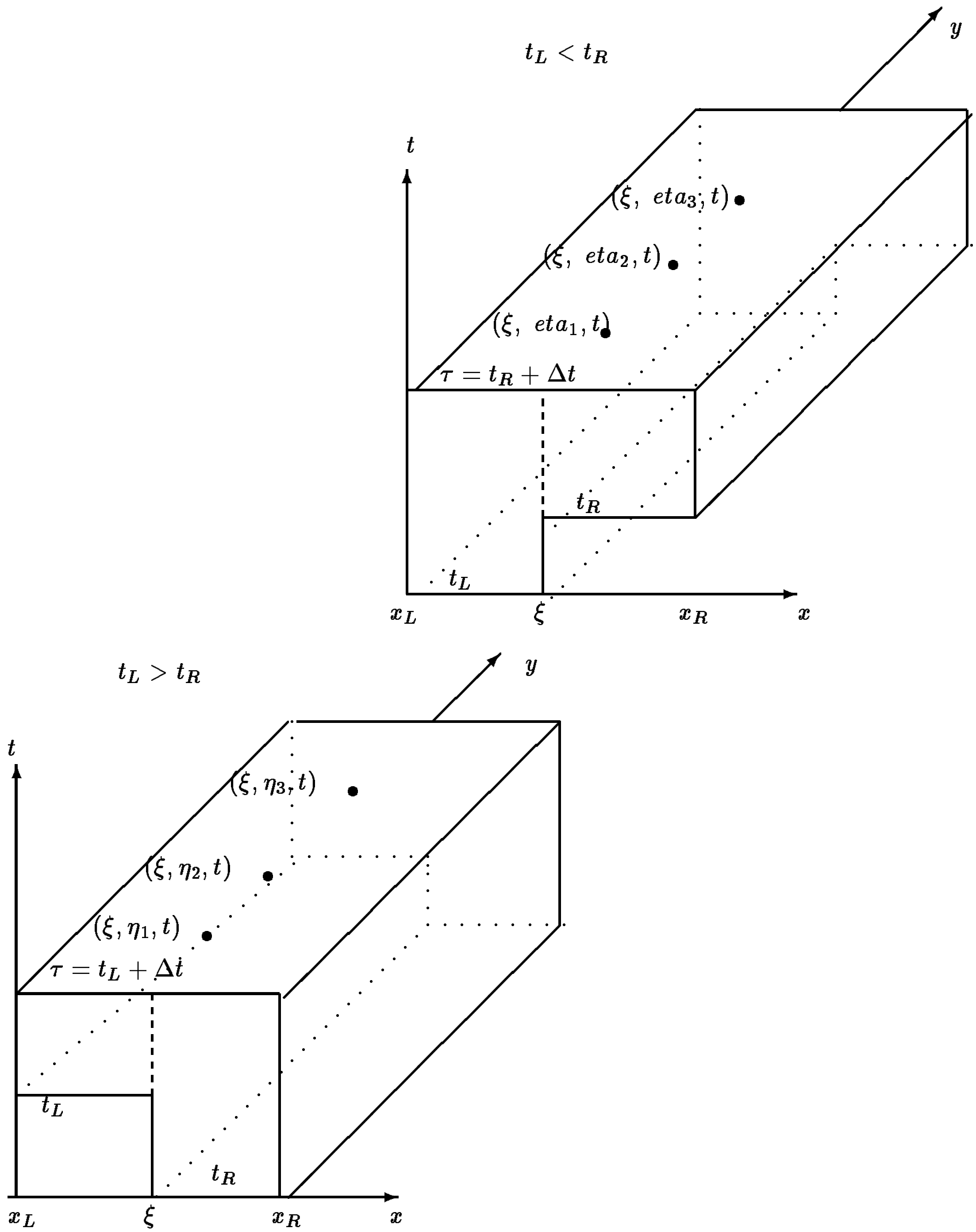

FIG. 7. : 2-D Asynchronous case: $x = \xi$ is a boundary between two adjacent processors and $x = x_L, x_R$ are some inner grid points of the left and right processors, respectively, along the x-axis.

In order for the fast processor to evaluate the solution u at $t=\tau$ for all its grid points, the values at $t=\tau$ of the intersection grid points $(\xi_1,\xi_2,\ldots,\xi_d)$ have to be known. Since $v(\vec{x},t)\longrightarrow\delta(x_1-\xi_1)\cdots\delta(x_d-\xi_d)$ as $t\to\tau^-$, we obtain on the right-hand side of Eq. (3.29) that

$$\int_{t=\tau} uv\,ds = u(\xi_1,\ldots,\xi_d)$$

Therefore, Eq. (3.29) may be used to determine the updated values of the intersection grid points (i.e. $u(\vec{\xi},\tau)$), by applying again the Trapezoidal rule in time. Once more, the synchronous case is obtained when $t_L=t_R$.

For example, Fig. 7 illustrates the 2-D case when $t_L>t_R$ and when $t_L<t_R$. In this case $x_1=x,\ x_2=y,\ x=\xi$ is the boundary plane between two adjacent sub-domains and $\xi_2=\eta_j$ such that the spatial intersection points of the sub-domains are $(\xi,\eta_j),\ \ 0\le\eta_j\le 1$. The surface S is the union of the following sub-surfaces:

$$\begin{aligned}
S_1 &= \{(x,y,t):\ x_L\le x\le\xi,\ \ y=0,\ \ t_L\le t\le\tau\}\\
S_2 &= \{(x,y,t):\ \xi\le x\le x_R,\ \ y=0,\ \ t_R\le t\le\tau\}\\
S_3 &= \{(x,y,t):\ x=x_R,\ \ 0\le y\le 1,\ \ t_R\le t\le\tau\}\\
S_4 &= \{(x,y,t):\ x_L\le x\le\xi,\ \ y=1,\ \ t_L\le t\le\tau\}\\
S_5 &= \{(x,y,t):\ \xi\le x\le x_R,\ \ y=1,\ \ t_R\le t\le\tau\}\\
S_6 &= \{(x,y,t):\ x=x_L,\ \ 0\le y\le 1,\ \ t_L\le t\le\tau\}
\end{aligned}$$

$$S_7=\{(x,y,t):\ \ x=\xi,\ \ 0\le y\le 1,\ \ t_R\le t\le t_L\}\qquad \text{if } \mathrm{t_R}<\mathrm{t_L}$$

or

$$S_7=\{(x,y,t):\ \ x=\xi,\ \ 0\le y\le 1,\ \ t_L\le t\le t_R\}\qquad \text{if } \mathrm{t_L}<\mathrm{t_R}$$

Then, using Eq. (3.29) with the Trapezoidal rule in time we get:

$$\begin{aligned}
u(\xi,\eta_j,\tau) &= \left(\tfrac{\tau-t_L}{2}\right)\int_{y=0}^{1}\int_{x=x_L}^{\xi}(vF)(x,y,t_L)\,dx\,dy\\
&+\left(\tfrac{\tau-t_R}{2}\right)\int_{y=0}^{1}\int_{x=\xi}^{x_R}(vF)(x,y,t_R)\,dx\,dy\\
&+\tfrac{2\tau-(t_R+t_L)}{4}F(\xi,\eta_j,\tau)\\
&+\int_{y=0}^{1}\left[\int_{x=x_L}^{\xi}(uv)(x,y,t_L)\,dx+\int_{x=\xi}^{x_R}(uv)(x,y,t_R)\,dx\right]dy\\
&-\left|\tfrac{t_L-t_R}{2}\right|\int_{y=0}^{1}\left[a_1^2(vu_x-uv_x)(\xi,y,t_L)+a_1^2(vu_x-uv_x)(\xi,y,t_R)\right]dy\\
&+\left(\tfrac{\tau-t_R}{2}\right)\int_{y=0}^{1}a_1^2(vu_x-uv_x)(x_R,y,t_R)\,dy\\
&+\left(\tfrac{\tau-t_L}{2}\right)\int_{y=0}^{1}a_1^2(vu_x-uv_x)(x_L,y,t_L)\,dy\\
&-\left(\tfrac{\tau-t_L}{2}\right)\int_{x=x_L}^{\xi}\left[a_2^2(vu_y-uv_y)(x,0,t_L)-a_2^2(vu_y-uv_y)(x,1,t_L)\right]dx\\
&-\left(\tfrac{\tau-t_R}{2}\right)\int_{x=\xi}^{x_R}\left[a_2^2(vu_y-uv_y)(x,0,t_R)-a_2^2(vu_y-uv_y)(x,1,t_R)\right]dx\\
&+O(\tau-t_L)^3+O(\tau-t_R)^3+O(t_R-t_L)^3
\end{aligned}\tag{3.30}$$

Once again, the synchronous case is obtained by substituting $t_0=t_L=t_R$ in our previous expressions, (see Fig. 8), resulting:

$$\begin{aligned}
u(\xi,\eta_j,\tau) &= \tfrac{\Delta t}{2}F(\xi,\eta_j,\tau)\\
&+\tfrac{\Delta t}{2}\int_0^1\int_{x_L}^{x_R}(vF)(x,y,t_0)\,dx\,dy\\
&+\int_0^1\int_{x_L}^{x_R}(uv)(x,y,t_0)\,dx\,dy\\
&+\tfrac{\Delta t}{2}\int_0^1\left[a_1^2(vu_x-uv_x)(x_R,y,t_0)+a_1^2(vu_x-uv_x)(x_L,y,t_0)\right]dy\\
&-\tfrac{\Delta t}{2}\int_{x_L}^{x_R}\left[a_2^2(vu_y-uv_y)(x,0,t_0)-a_2^2(vu_y-uv_y)(x,1,t_0)\right]dx\\
&+O(\Delta t)^3
\end{aligned}\tag{3.31}$$

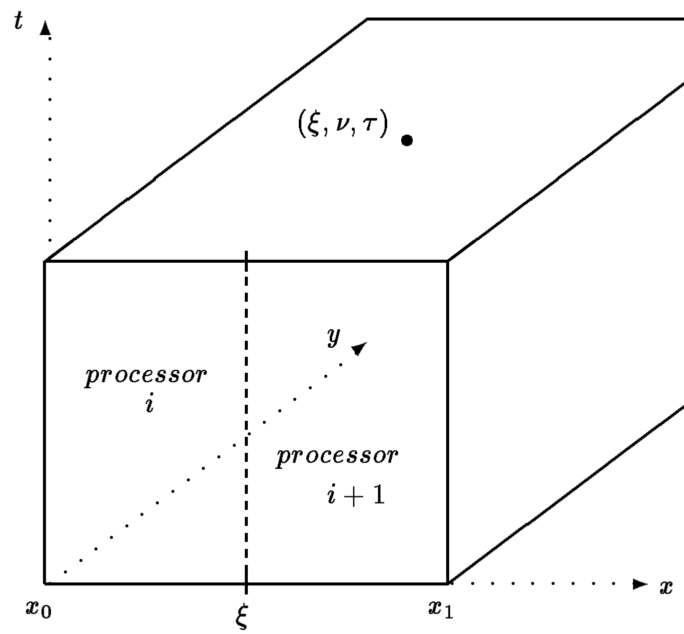

FIG. 8. : 2-D synchronous case

In the 3D case, the integration on the left hand side of Eq. (3.29) is extended over a four-dimensional cylinder whose generatrix is parallel to the time axis. The integration in the last term on the right hand side of Eq. (3.29) is extended over the three-dimensional lateral surface of this cylinder. In this case each term in Eq. (3.30) would include also an integration in the $x_3 = z-$domain and extra terms will be added that correspond to $z-$dependence.

3.1.3 Non-linear Case In this section we would like to demonstrate how the Green's functions method can be extended by a simple device to more complicated situations. For example, following an idea of [10] it can be applied to a family of non-linear problems:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 \beta(u)}{\partial x^2} + f(x,t,u) \tag{3.32}$$

along with the initial and boundary conditions, (e.g., $u(0,t) = u(\pi,t) = 0$ and $u(x,0) = u_0(x)$), where $\beta(u(x,t))$ is a function of $u(x,t)$. Following [10], we rewrite Eq. 3.32 as,

$$\mu(\beta(u))_t = (\beta(u))_{xx} + F(x,t,u,u_t) \tag{3.33}$$

where, $F(x,t,u,u_t) = f(x,t,u) + \mu\beta(u)_t - u_t$ and μ ($\mu > 0$) is chosen so that

$$0 \leq \mu \leq L_\beta^{-1} \tag{3.34}$$

where L_β^{-1} is the Lipschitz constant for β.

Eq. (3.33) is now in the form of Eq. (2.1). Thus, we can derive the analog to Eq. (3.16):

$$\begin{aligned} \int_D vF\,dx\,dt &= \int_{t=t_L} v\beta dx + \int_{t=t_R} v\beta dx + \int_{t=\tau} v\beta dx \\ &- \int_{x=x_L} \frac{1}{\mu}\left(v\frac{\partial \beta}{\partial n} - \beta\frac{\partial v}{\partial n}\right)dt \\ &- \int_{x=\xi} \frac{1}{\mu}\left(v\frac{\partial \beta}{\partial n} - \beta\frac{\partial v}{\partial n}\right)dt \\ &- \int_{x=x_R} \frac{1}{\mu}\left(v\frac{\partial \beta}{\partial n} - \beta\frac{\partial v}{\partial n}\right)dt \end{aligned} \tag{3.35}$$

Then, with similar approximations and analogous to Eq. 3.23 we get

$$
\begin{aligned}
\beta(\xi,\tau) &= \tfrac{2\tau-(t_L+t_R)}{4}\tfrac{1}{\mu}f(\xi,\tau,u(\xi,\tau)) \\
&+(\tfrac{\tau-t_L}{2})\int_{x_L}^{\xi}(vf)(x,t_L,u(x,t_L))dx+(\tfrac{\tau-t_R}{2})\int_{\xi}^{x_R}(vf)(x,t_R,u(x,t_R))dx \\
&+\beta(u(\xi,\tau))-\beta(u(\xi,t_R))-\tfrac{1}{\mu}u(\xi,\tau)+\tfrac{1}{\mu}u(\xi,t_R) \\
&+\int_{x_L}^{\xi}(v\beta)(x,t_L)dx+\int_{\xi}^{x_R}(v\beta)(x,t_R)dx \\
&-\left|\tfrac{t_R-t_L}{2}\right|\left[\tfrac{1}{\mu}(v\beta_x)(\xi,t_R)+\tfrac{1}{\mu}(v\beta_x)(\xi,t_L)\right] \\
&-\left(\tfrac{\tau-t_L}{2}\right)\left[\tfrac{1}{\mu}(v\beta_x-\beta v_x)(x_L,t_L)\right]-\left(\tfrac{\tau-t_R}{2}\right)\left[\tfrac{1}{\mu}(v\beta_x-\beta v_x)(x_R,t_R)\right]
\end{aligned}
\tag{3.36}
$$

CLAIM 1. *(***Stability***): The asynchronous-explicit scheme of Eq. (3.35) is stable if:*

$$
\begin{aligned}
&0\le\mu\le L_\beta^{-1}\\
&\beta'(\nu)\ge 0 \qquad \forall\nu\\
&-Q\le f_u\le 0
\end{aligned}
$$

where L_β^{-1} is the Lipschitz constant for β and Q is constant.

3.2 Implementation and Computational Considerations

We now present the finite differences form of the approximations Eqs. (3.23), (3.24) and (3.36). We use our finite differences notation, as opposed to the one used in these sections. We examine and highlight some computational and practical aspects (e.g., cases where a singularity exists or when there is not enough grid points). Finally, we analyze the numerical properties to prove their compliance with our initial requirements for a mixed asynchronous implicit-explicit method.

3.2.1 The Finite Difference Form of the Sub-domain Boundary Approximations An approximation to the solution at sub-domains' boundaries is given by Eq. (3.30) for the multi-dimensional case (or reduced to Eq. (3.23) for the 1-D case). Each of the spatial integrals in this equation can be approximated by some high-order numerical quadrature. In our implementation we used Simpson's Rule, along with high-order finite-difference approximation for the spatial partial derivative.

There are several issues that have to be addressed when calculating the integrals. First, we note that the contributions of the integrals at $x = x_L$ and $x = x_R$ are insignificant and therefore can be neglected. We also assume that x_L and x_R are at neighboring sub-domains, which is always the case when there are sufficient grid points per sub-domain. This is significant for some parallel computing architectures in which communication between neighboring processors is highly efficient than any other form of communication.

The decomposition of the problem domain is straightforward. The domain is divided into sub-domains each with more or less the same number of grid points. Yet, to obtain better approximations in the asynchronous case, unlike the synchronous one, we overlap the sub-domain boundaries. Thus, in the one-dimensional case the grid point at a sub-domain boundary may be calculated at both sub-domains.

Following, is a simple example how to apply the Trapezoidal Rule to the one-dimensional synchronous equation (3.24). It yields an explicit scheme in which the computation of the next step is based on the current values. The scheme coefficients are not constant. They depend on Δx and Δt (and on a^2 when it is non-constant). In fact, in cases where Δx and Δt are constants and a^2 depends only on x, we can prepare *in advance* a lookup table with the corresponding coefficients. In addition, if a^2 is constant the coefficients are symmetric. Thus, the lookup table has to contain only half of the coefficients.

We use the same notation as defined in section 2.2.1. the 1-D synchronous scheme becomes:

$$u_i^{n+1} = \frac{1}{2}\Delta t F_i^{n+1} + \sum_{m=-M}^{M} \frac{\Delta x}{\sqrt{4\pi a^2 \Delta t}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t}} (u_{i+m}^n + \frac{1}{2}\Delta t F_{i+m}^n) \tag{3.37}$$

The 2-D synchronous scheme is:

$$u_{i,j}^{n+1} = \frac{1}{2}\Delta t F_{i,j}^{n+1} + (\Delta x)(\Delta y)\sum_l \sum_m \left(v_{i,j}^n u_{i+m,j+l}^n + \frac{1}{2}\Delta t F_{i+m,j+l}^n \right) \tag{3.38}$$

where,

$$v = \frac{1}{4\pi a^2 \Delta t} e^{-[\frac{(m\Delta x)^2}{4a^2\Delta t} + \frac{(l\Delta y)^2}{4a^2\Delta t}]}$$

Whereas, the 1-D *non-linear* synchronous scheme is:

$$(1+\mu)\beta_i^{n+1} - \frac{1}{2}\Delta t f_i^{n+1} + u_i^{n+1} = \sum_{m=-M}^{M} \frac{\Delta x \mu}{\sqrt{4\pi \Delta t}} e^{-\frac{\mu(m\Delta x)^2}{4\Delta t}} (\beta_{i+m}^n + \frac{1}{2}\Delta t f_{i+m}^n) - \mu\beta_i^n + u_i^n \tag{3.39}$$

with β, f and μ as in Eq. (3.33) and the conditions set Claim 1.

In the asynchronous case Eqs. (3.23) and (3.30) represent the general case where all the points $(i,\ i \pm 1)$ are at different time. However, if we assume that there are many grid points per processor, the points to the left of the sub-domain's right boundary (and similarly, those to the right of the sub-domain's left boundary) are at the same time as that of the boundary. Thus, we have two forms for the asynchronous-explicit equation. One for the right sub-domain boundary where $t_{i-1}^{n_i} = t_i^{n_i}$ (or $t_R = t$) and the other for left sub-domain boundary where $t_{i+1}^{n_i} = t_i^{n_i}$ (or $t_L = t$).

Moreover, unlike the synchronous case, caution should be practiced when calculating the derivatives so that all values will be at the same time. This is achieved by using high-order one-sided finite-difference approximations, such as:

$$(u_x)_i(t_i^{n_i}) = \frac{1}{2\Delta x}[\, u_{i-2}(t_i^{n_i}) - 4u_{i-1}(t_i^{n_i}) + 3u_i(t_i^{n_i})\,] + O(\Delta x^2) \tag{3.40}$$

or

$$(u_x)_i(t_i^{n_i}) = \frac{1}{2\Delta x}[\, -u_{i+2}(t_i^{n_i}) + 4u_{i+1}(t_i^{n_i}) - 3u_i(t_i^{n_i})\,] + O(\Delta x^2)$$

The 1-D asynchronous-explicit scheme (*right* sub-domain boundary form) becomes:

$$\begin{aligned} u_i(t_i^{n_i+1}) &= \tfrac{1}{4}(\Delta t_i^{n_i} + \Delta t_{i+1}^{n_i}) F_i(t_i^{n_i+1}) \\ &+ \textstyle\sum_{m=-M}^{0} \frac{\Delta x}{\sqrt{4\pi a^2 \Delta t_i^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_i^{n_i}}} \left[u_{i+m}(t_i^{n_i}) + \tfrac{1}{2}\Delta t_i^{n_i} F_{i+m}(t_i^{n_i}) \right] \\ &+ \textstyle\sum_{m=0}^{M} \frac{\Delta x}{\sqrt{4\pi a^2 \Delta t_{i+1}^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_{i+1}^{n_i}}} \left[u_{i+m}(t_{i+1}^{n_i}) + \tfrac{1}{2}\Delta t_{i+1}^{n_i} F_{i+m}(t_{i+1}^{n_i}) \right] \\ &- \tfrac{1}{2}(t_i^{n_i} - t_{i+1}^{n_i}) \left\{ \frac{a^2}{\sqrt{4\pi a^2 \Delta t_i^{n_i}}} \frac{1}{2\Delta x} \left[u_{i-2}(t_i^{n_i}) - 4u_{i-1}(t_i^{n_i}) + 3u_i(t_i^{n_i}) \right] \right. \\ &+ \left. \frac{a^2}{\sqrt{4\pi a^2 \Delta t_{i+1}^{n_i}}} \frac{1}{2\Delta x} \left[-u_{i+2}(t_{i+1}^{n_i}) + 4u_{i+1}(t_{i+1}^{n_i}) - 3u_i(t_{i+1}^{n_i}) \right] \right\} \end{aligned} \tag{3.41}$$

where the left sub-domain boundary form is similar to the above.

The 2-D schemes are essentially similar. When the domain is divided into strips, if there are not sufficient grid points to the left (or to the right) of the evaluated point to calculate the derivatives by standard finite differences, we apply one-sided approximations (e.g., Eq. 3.40).

Although obtaining an high-order accuracy may seem difficult, we emphasize that other higher-order schemes besides the above synchronous (Eq. 3.37) and asynchronous (Eq. 3.41) are obtained. The spatial integrals may be approximated by high-order numerical quadratures using more neighboring points. In general, when more grid points are used for the approximations a higher accuracy is achieved. However, an high-order approximation of the time integrals is not straightforward. In the Appendix we present high-order direct approximations for the time integrals.

3.2.2 Numerical Analysis of the Green's Functions Based Asynchronous-Explicit Scheme In the following we discuss the numerical properties of the explicit-asynchronous scheme given by Eq. 3.41.

CLAIM 2. (**Consistency**): *The asynchronous-explicit scheme (Eq. 3.41) is consistent with Eq. (2.1).*

Proof. By construction (as detailed in section 3.1). □

It is easy to verify that the synchronous scheme (SY) (given by Eq. (3.37)) comply with the consistency condition (2.7).

For the proof of stability, we first establish the stability of the synchronous scheme (SY) given by Eq. (3.37).

CLAIM 3. (**Stability SY**): *The synchronous scheme (Eq. 3.37) is stable.*

Proof. By lemma 2.4 it is sufficient to show that the sum of the absolute value of the scheme coefficients is equal or less than one. Since,

$$(3.42)\quad \sum_{m=-M}^{M} \left|\frac{\Delta x}{\sqrt{4\pi a^2 \Delta t}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t}}\right| \le \sum_{m=-\infty}^{\infty} \left|\frac{\Delta x}{\sqrt{4\pi a^2 \Delta t}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t}}\right| = \frac{\Delta x}{\sqrt{4\pi a^2 \Delta t}} \frac{\sqrt{4a^2\Delta t}}{\Delta x}\sqrt{\pi} = 1$$

Then, the scheme of Eq. (3.37) is stable. □

Whereas, for the asynchronous-explicit scheme (AS) given by Eq. (3.41), we have,

CLAIM 4. (**Stability AS**): *The asynchronous-explicit scheme (Eq. 3.41) is stable.*

Proof. By lemma 2.4 it is sufficient to show that the sum of the absolute value of the scheme coefficients is equal or less than one,

$$(3.43)\quad \begin{aligned} &\textstyle\sum_{m=-M}^{0} \left|\frac{\Delta x}{\sqrt{4\pi a^2 \Delta t_i^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_i^{n_i}}}\right| + \textstyle\sum_{m=0}^{M} \left|\frac{\Delta x}{\sqrt{4\pi a^2 \Delta t_{i+1}^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_{i+1}^{n_i}}}\right| \\ - &\tfrac{1}{2}|t_i^{n_i} - t_{i+1}^{n_i}| \left|\frac{a^2}{\sqrt{4\pi a^2 \Delta t_i^{n_i}}}\frac{1}{2\Delta x}8 + \frac{a^2}{\sqrt{4\pi a^2 \Delta t_{i+1}^{n_i}}}\frac{1}{2\Delta x}8\right| \end{aligned}$$

We apply the asynchronous-explicit equation given by Eq. (3.41) only in cases where

$$\Delta t_{i+1}^{n_i} > \Delta t_i^{n_i}$$

Otherwise, we use the value already computed at the neighboring sub-domain. Particularly, with our choice of $\Delta t_i^{n_i}$ (e.g., Algorithm 3 and Table 2)

It follows that,

$$\frac{1}{\sqrt{4\pi a^2 \Delta t_{i+1}^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_{i+1}^{n_i}}} \le \frac{1}{\sqrt{4\pi a^2 \Delta t_i^{n_i}}} e^{-\frac{(m\Delta x)^2}{4a^2\Delta t_i^{n_i}}}$$

Therefore, the coefficients are bounded by those of the synchronous case, which are bounded by one. □

CLAIM 5. (**Error Estimate**): *The error bound is a function of the number of grid points used in the computation.*

Proof. Since v (v - defined in Eq. 3.28) decreases as $(x-\xi)^2$ increases (for small Δt), the integral of Eq. 3.29 is bounded by the maximum value of u and the exponential. The dominant part of the kernel is $e^{\frac{-\xi^2}{4\Delta t}}$, which is essentially the exponential. Assume, $\xi = m\Delta x$, then $e^{-\frac{x_i^2}{4\Delta t}} = e^{-\frac{m^2\Delta x^2}{4\Delta t}} = e^{-\frac{m^2}{4r}}$ where, $\xi = x_i$ and $r = \frac{\Delta t}{\Delta x^2}$. □

This can be used for controlling the number of steps or the number of neighboring points used in relation to the allowed error. For example, comparing with explicit schemes where the stability requirement is $r = \frac{1}{2}$ then $e^{-\frac{m^2}{2}}$. Hence, the third neighbor ($m \approx 3$) already yields 0.01. For implicit schemes, where larger Δt values are possible, denote $\alpha \approx \frac{\Delta t}{\Delta x}$ then, $e^{-\frac{m^2\Delta x^2}{4\Delta t}} = e^{-\frac{m^2\Delta x}{4\alpha}}$ we need $m^2\Delta x = o(1)$, then if $I\Delta x = 1$, $\Delta x \approx \frac{1}{I}$ and we require $\frac{m^2}{I} = o(1)$ or $m \approx \sqrt{I}$. These estimates were found to be practical.

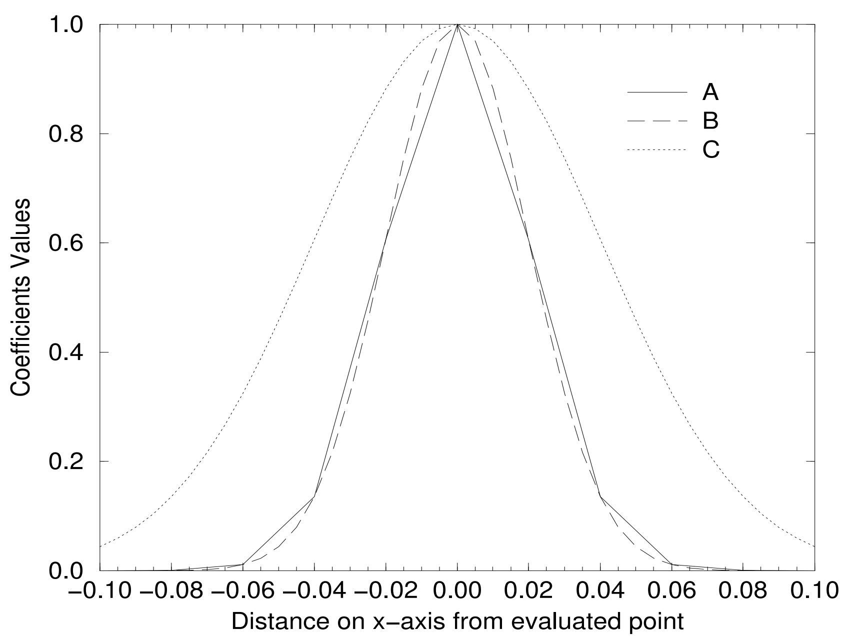

FIG. 9. : The coefficients as a function of Δx and $\Delta t_i^{n_i}$: (A) $\Delta x = 0.005$ and $\Delta t_i^{n_i} = 0.0002$, (B) $\Delta x = 0.02$ and $\Delta t_i^{n_i} = 0.0002$, (C) $\Delta x = 0.005$ and $\Delta t_i^{n_i} = 0.0008$

The behavior of the coefficients as a function of Δx and $\Delta t_i^{n_i}$ is illustrated in Figure 9. To achieve the same accuracy, if Δx is divided by a factor of Q, the number of points to consider is multiplied by Q. If $\Delta t_i^{n_i}$ is divided by a factor of Q, the number of points to consider is divided by $\sqrt{Q}$.

DEFINITION 3.1. *An explicit scheme is local when its computation requires grid points only from neighboring sub-domains.*

CLAIM 6. (**Locality**): *The asynchronous-explicit scheme of Eq. 3.41 is local.*

Proof. By claim 5 the number of grid points used by the asynchronous-explicit scheme of

Eq. 3.41 can be controlled. Thus, we can discretize the spatial domain (Δx) so there are sufficient grid points at each sub-domain (at least $2M$), so that all points are at neighboring processors, for any M chosen for accuracy purposes. Alternatively, we can control any of those parameters, so that for known Δx and known M we may achieve a certain accuracy. □

4 Numerical Examples and Performance Analysis

4.1 Implementation Specifics

This section summarizes our specific implementation (problems and approximations). The results presented in the subsequent section are related to this implementation.

We have implemented and examined our proposed schemes on two different multi-user MIMD architectures: the *Cray J90* and the *Sequent Symmetry* [18] which are commercial shared bus and shared memory multi-processors. The *Sequent* served during our initial implementation and testing phase whereas more up to date results were later obtained on the *Cray J90*. Table 3 highlights the features of the specific computers we used.

Model	Cray J932/16	Sequent Symmetry 81
# CPUs	16	24
Processors	YMP, 200 MFlops	Intel 80386 processors, Intel 80387 and Weitek WTL 3167 floating-point co-processors
Central Memory	4 GB	32 MB
Local Memory	none	none
Cache Size	for vectorization	64 KB (copyback)
Operating System	UNICOS sos.3	DYNIX(R) V3.1.4

TABLE 3: Features of the multi-processors on which the algorithms are implemented.

The following parameters were used in our implementation:

- Δx - The discretization length along the $x-$axis.
- Δy - The discretization length along the y-axis in the two dimensional case.
- Δt - The discretization length along the $t-$axis.
- T - The time to be reached for a given problem ("problem time").
- N - Number of time-steps needed by the corresponding *scheme*, in order to reach $t = T$.
- p - The number of processing elements used for the calculation.
- τ_p - The CPU time in seconds with p processors.
- γ_p - The aggregate CPU time accumulated by the p processors.
- E - The maximal relative error, in absolute value, of the numerical result relative to the analytic solution.
- *Speedup* - Run time of a corresponding serial program relative to the parallel program. The corresponding *efficiency* is defined as speedup divided by p.

The schemes used are *implicit* within the sub-domain but vary in their *explicit* approximation used on sub-domains' boundaries. Here, we denote by $\mathrm{GR}-\mathrm{SY}_{(\nu)}$ the synchronous approximation and by $\mathrm{GR}-\mathrm{AS}_{(\nu)}$ the asynchronous one. In this notation, the first qualifier denotes the strategy for processing the interfaces (e.g. GR denotes the usage of the Green function based approximation), and the second denotes whether the processors are globally synchronized. Specifically, the following is a description of the *parallel* approximations we consider along with the notations we use:

- $\mathrm{GR}-\mathrm{SY}_{(\nu)}$: we use the Simpson's Rule as the numerical quadrature for the spatial integral with spatial accuracy $O(\Delta x)^{\nu}$ to evaluate the sub-domains' boundary values. Within each sub-domain we use an implicit finite-difference scheme of the same order. A barrier mechanism is used for synchronization. The domain is split equally among the processors.

- $\mathrm{GR}-\mathrm{AS}_{(\nu)}$: we use the Simpson's Rule as the numerical quadrature for the asynchronous integral with spatial accuracy $O(\Delta x)^{\nu}$. Within each sub-domain we use an implicit finite-difference scheme of the same order. Each processor computes its own sub-domain and the calculations are performed without synchronizing with other processors. Unlike the synchronous case, sub-domains' boundary points are maintained at both neighboring processors (overlap). If the neighboring processor is faster, we use its sub-domain's boundary values and save the computation. Otherwise, we calculate the corresponding time integral based on the values associated with different time.

In addition, for comparison with *serial* schemes (denoted by SR), and also within each sub-domain, the following implicit schemes are used:

- SR_2: Backward-Euler is used for $\nu = 2$ and is denoted by SR_2.

- SR_4: The following implicit scheme is denoted by SR_4:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \frac{\theta\delta_x^2 u_i^{n+1} + (1-\theta)\delta_x^2 u_i^n}{\Delta x^2} \qquad , \quad \text{where} \quad \theta = \frac{1}{2}(1-\frac{\Delta \mathrm{x}^2}{6\Delta \mathrm{t}})$$

  This scheme is $O(\Delta x^4 + \Delta t^2)$ [16]. When using this scheme within sub-domain, we maintain the same order of accuracy for calculating the interfaces.

  In the two-dimensional case, we use the updated Cranck-Nicholson scheme ($O(\Delta x)^2 + O(\Delta y)^2$) for ADI [16]:

$$\frac{u_{ij}^{n+\frac{1}{2}} - u_{ij}^n}{\Delta t} = \frac{1}{2}\frac{\delta_x^2 u_{ij}^{n+\frac{1}{2}}}{\Delta x^2} + \frac{1}{2}\frac{\delta_y^2 u_{ij}^n}{\Delta y^2}$$

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+\frac{1}{2}}}{\Delta t} = \frac{1}{2}\frac{\delta_x^2 u_{ij}^{n+\frac{1}{2}}}{\Delta x^2} + \frac{1}{2}\frac{\delta_y^2 u_{ij}^{n+1}}{\Delta y^2}$$

  Along the sub-domains' boundaries we use the Trapezoidal Rule ($\nu = 2$) for the spatial integrals in either the synchronous or asynchronous case.

Following are the problems we solved. Although these problems are simple, we constructed them specifically in order to investigate the nature of the error as polynomials of the same order of accuracy as that of the schemes. The implementation of more complicated problems is essentially the same.

For the *one-dimensional* case, we solved the following problem:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad 0 < x < 1 \tag{4.44}$$

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

whose analytic solution:

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

For the *non-homogeneous* problem we solved:

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = x^2 \tag{4.45}$$

whose analytic solution is:

$$u_A = x^2 t + x + t^2$$

As an example for a *two-dimensional* case we solve:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \qquad 0 < x, y < 1 \tag{4.46}$$

with initial condition, $u(x, y, 0) = \cos(\pi x)\cos(\pi y)$ and boundary conditions,

$$u(0,y,t) = \cos(\pi y)e^{-2\pi^2 t} \qquad , \qquad u(1,y,t) = -\cos(\pi y)e^{-2\pi^2 t}$$

$$u(x,0,t) = \cos(\pi x)e^{-2\pi^2 t} \qquad , \qquad u(x,1,t) = -\cos(\pi x)e^{-2\pi^2 t}$$

whose analytic solution is

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

4.2 Results and their Analysis

It is evident that both synchronous and asynchronous algorithms (Algorithms 1 and 2) have inherent builtin parallelism. They are recognized as 99% parallel by the *atexpert* tool of *Cray* (The atexpert is a software tool with which *Cray* checks the parallel nature of a program. It predicts the speedup using extrapolation based on data gathered during run-time on a single CPU). In the succeeding discussion we present the actual results and analyze them with respect to the theoretical estimates. In our analysis we use the following terms:

CPU time	Time elapsed in seconds until the program ended. Although the system is multi-user (not dedicated) it does not include the time spent on other programs, but it includes system wait time.
Aggregate CPU time	Adding up CPU time of all processors that run simultaneously.
Multi-processor utilization	During run-time the *Cray* system records the CPU time used by a single processor only, by only two processors simultaneously, by only three simultaneously, and etc. The average of the recorded figures is the multi-processor utilization. This utilization times the CPU time results the aggregate CPU time.
Atexpert speedup	A tool on the *Cray* which predicts the speedup calculated using extrapolation based on data of a single processor gathered during run-time.
Speedup	The relation between CPU time when running with several processors divided to the CPU time when running with a single processor: $speedup = \frac{\text{CPU} \quad \text{time}(p \quad processors)}{\text{CPU} \quad \text{time}(single \quad processor)}$

Observations to the parallel nature of the synchronous implementation are illustrated in Fig. 10. The solid line presents the optimal (linear) speedup. The dotted line is the speedup predicated by the *atexpert* tool.

The multi-processor utilization met this predication and was even better when the overall system load was less than 85%. This is presented by the long-dashed line. When the system load was higher than 85% the multi-processor utilization deteriorated as illustrated by the dashed line.

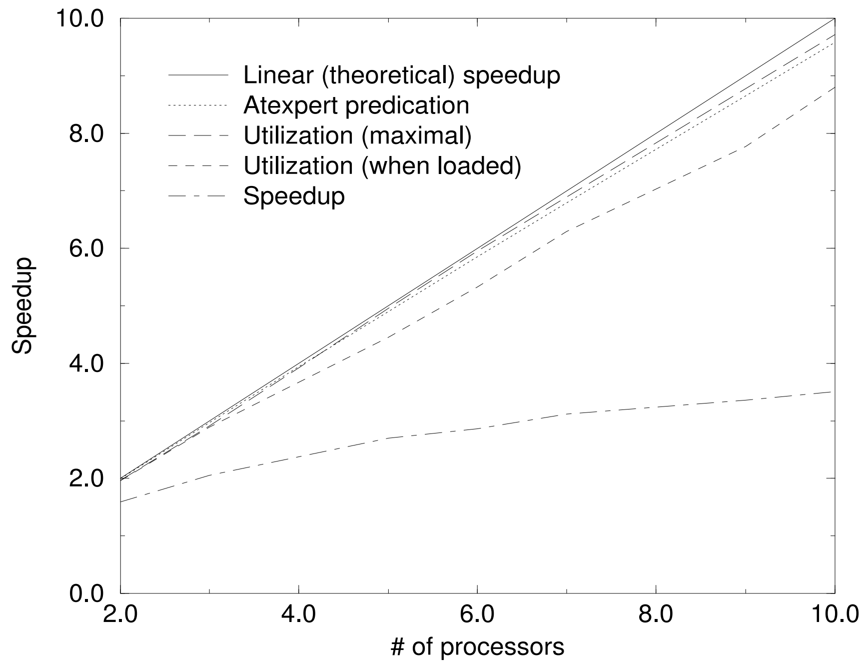

FIG. 10. : Speedup comparison on the *Cray*: 1-D synchronous results for the homogeneous problem (4.44) with $\nu = 2$, $T = 0.1$, and $\Delta x = 0.001587$.

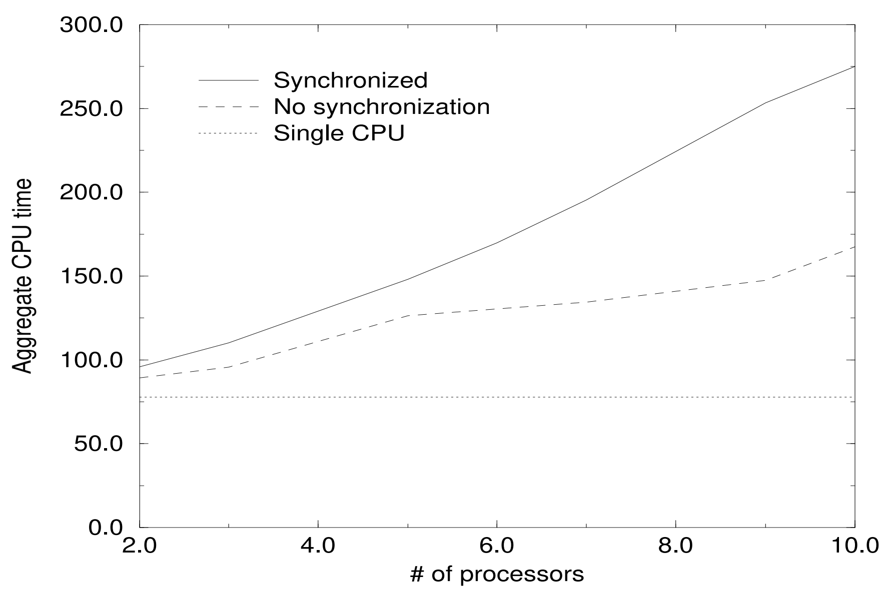

FIG. 11. : Aggregate CPU time comparison on the *Cray*, associated with the speedup results in Fig. 10.

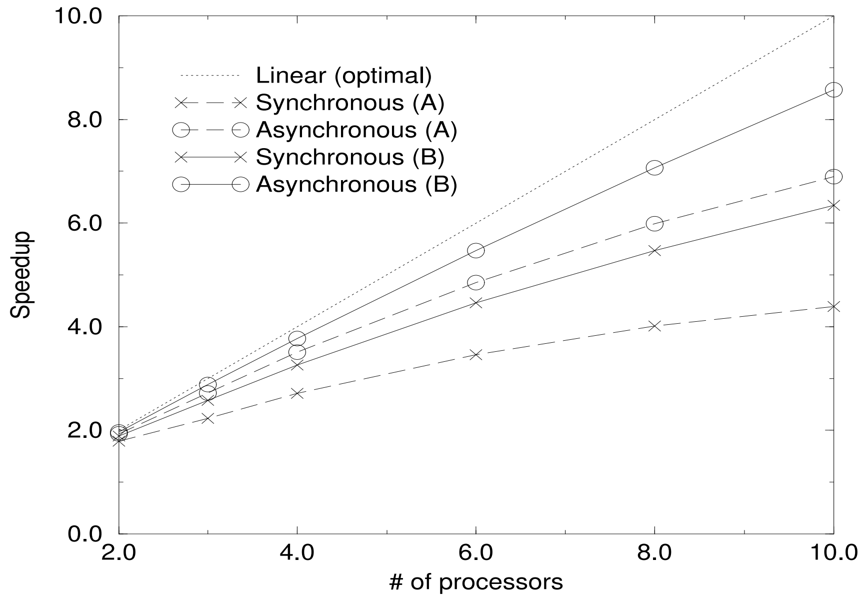

FIG. 12. : Speedup comparison on the *Sequent*: 1-D synchronous and asynchronous results for the homogeneous problem (4.44) with $\nu = 4$, $T = 0.05$, and different values of Δx; (A) $\Delta x = 0.00412$, (B) $\Delta x = 0.00209$.

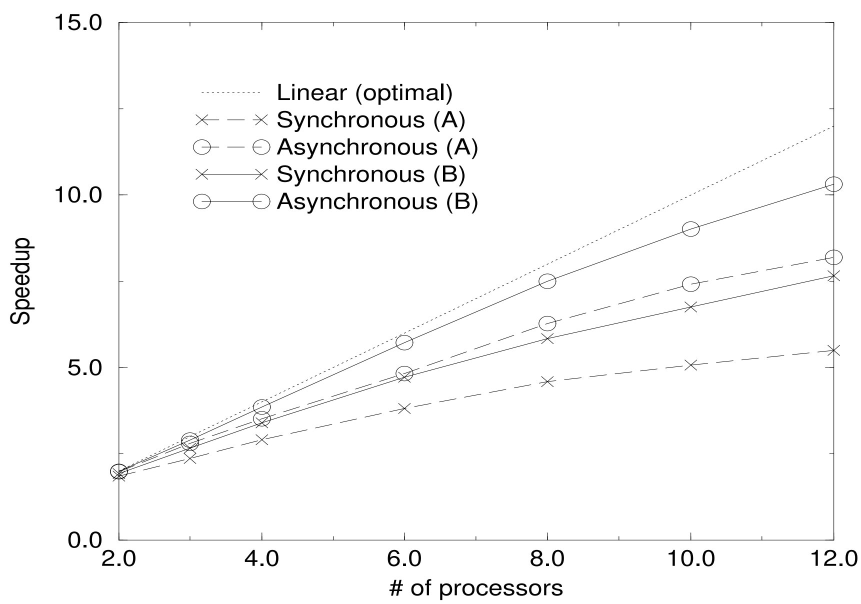

FIG. 13. : Speedup comparison on the *Sequent*: synchronous and asynchronous results for the homogeneous 2-D problem (4.46) with $\nu = 2$, $T = 0.005$, and different values of Δx; (A) $\Delta x = 0.00412$, (B) $\Delta x = 0.00209$.

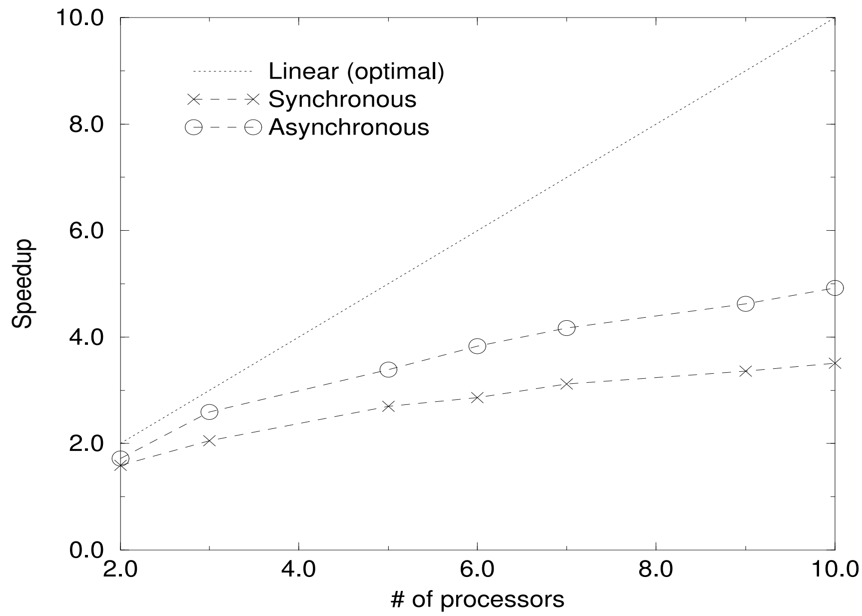

FIG. 14. : Speedup comparison on the *Cray*: 1-D synchronous and asynchronous results for the homogeneous problem (4.44) with $\nu = 2$, $T = 0.1$, and $\Delta x = 0.001587$.

Unfortunately, these figures and estimate are far better than the actual speedup presented by the dotted-dashed line. It is due to the following factors:

1. Synchronization overhead.
2. Communication overhead including access to shared variables.
3. Spawning the parallel processes.
4. Cost of computing the sub-domains' boundaries.

The above factors contribute additional CPU time to the aggregate CPU time. The aggregate time increases as the number of processors is increased. This is clearly illustrated by the solid line in Fig. 11 where the aggregate CPU time for 10 processors is almost 3 times the aggregate time of 2 processors.

We concentrate on the first factor (synchronization overhead) which is eliminated by the asynchronous algorithm. The dashed line in Fig. 11 represents the aggregate CPU time while running the synchronous program without activating synchronization. This is not the same as the asynchronous program since we do not use the asynchronous schemes on the sub-domains' boundaries and thus the computed values are corrupted. Thus, it is meant to serve only for the study of the synchronization overhead. The area between the solid and the dashed lines portrays the overhead associated with synchronization. The dotted line represents the CPU time of a single processor. The area between the dashed and the dotted lines depicts the overhead of the other three factors (without considering the special treatment that is needed at sub-domains' boundaries for the asynchronous case).

The speedup results which are illustrated for the *Sequent* in Fig. 12 are slightly better, although far from optimal. In Fig. 12 the dotted line presents the optimal (linear) speedup. The synchronous lines are marked with x. Two lines are presented for two different sizes of Δx. As Δx gets smaller there are more points per sub-domain and more computations per processor. Thus, the speedup improves due to a better ratio of computation to communication (see e.g., [6], [15]).

The 2-D results for the *Sequent* are illustrated in Fig. 13. The additional computation involved as compared with the 1-D case, results in better computation to communication ratio and thus improve speedup.

The speedup results on the *Cray* of the program implementing the asynchronous algorithm is illustrated by the dashed line with circles in Fig. 14. As in the synchronous case, the speedup is less than the multi-processor utilization due to the increased aggregate CPU time. Speedup results for the *Sequent* are illustrated by the dashed line with circles in Figs. 12 and 13. The asynchronous implementation is 15% - 35% times better than the speedup of the synchronous. In this case, more operations are required for accurate approximation of sub-domains' boundaries (compare the asynchronous scheme Eq. (3.41) with the synchronous one Eq. (3.37)). The communication penalty is reduced by asynchronous implementation at the expense of additional computations at sub-domains' boundaries. However, in many instances this computation is bypassed by taking an already available neighboring value (see Algorithm 3). Therefore, with the asynchronous algorithm less time is required for computing all sub-domains' boundaries than with the synchronous one.

Moreover, a real advantage is seen when the domain is decomposed unequally among the processors and in particularly for a loaded/slower processor with less loaded/faster neighbors. With the synchronous program all processors are delayed by the most loaded (and thus, slowest) processor. Whereas, in the asynchronous program the slower processors can take advantage of the faster ones and set a larger step size, while other processors can continue with the computation.

A computational advantage of either the synchronous or the asynchronous algorithms is that the size of the time step is unlimited.

5 Summary

This paper presents new parallel asynchronous high-order algorithms for the solution of Parabolic PDEs. Synchronous algorithms are obtained as special cases. These algorithms were implemented on the multi-user *Cray J90* and the *Sequent Balance* MIMD multiprocessors. The algorithms are based on a domain decomposition were each processor is assigned the task of processing its own sub-domain asynchronously. A high-order implicit finite-difference scheme is used within each sub-domain, after the values at sub-domains boundaries are determined by a high-order asynchronous-explicit scheme. It is derived by numerical quadratures of the corresponding *Green's function*. By this approach, one can obtain *any* desired accuracy. Numerical tests showed that such schemes provide high efficiency and degree of parallelism.

Appendix

To highlight the practical computational aspects we now show that the compensation in the asynchronous case for $x = \xi$ (see Eq. (3.23)) can be calculated to any desired accuracy. Let

$$I = I(\xi,\tau) = \int_{t_R}^{t_L} (\tau - t)^{-\frac{1}{2}} u(\xi,t)dt$$

LEMMA 5.1. *For* $k = 1, 2, \ldots$

$$I = -\sum_{j=0}^{k-1} J_j + I_k$$

where

$$J_j = \frac{2^{j+1}}{(2j+1)!!} \left[(\tau - t)^{j+\frac{1}{2}} \frac{\partial^j u}{\partial t^i} \right]_{t_R}^{t_L}$$

and

$$I_k = \frac{2^k}{(2k-1)!!} \int_{t_R}^{t_L} (\tau - t)^{k-\frac{1}{2}} \frac{\partial^k u}{\partial t^k} dt$$

Proof. By induction. For $k = 1$, from integration by parts we get:

$$I = -2 \left[(\tau - t)^{\frac{1}{2}} u \right]_{t_R}^{t_L} + 2 \int_{t_R}^{t_L} (\tau - t)^{\frac{1}{2}} \frac{\partial u}{\partial t} dt = -J_0 + I_1$$

Assume that it is true for $k = n$, i.e

$$I = -\sum_{j=0}^{n-1} J_j + I_n$$

where

$$I_n = \frac{2^n}{(2n-1)!!} \int_{t_R}^{t_L} (\tau - t)^{n-\frac{1}{2}} \frac{\partial^n u}{\partial t^n} dt$$

Then, using integration by parts we obtain,

$$\begin{aligned} I_n &= -\tfrac{2^n}{(2n-1)!!}\left\{\tfrac{2}{2n+1}\left[(\tau-t)^{n-\frac{1}{2}}\tfrac{\partial^n u}{\partial t^n}\right]_{t_R}^{t_L} + \tfrac{2}{2n+1}\int_{t_R}^{t_L}(\tau-t)^{n-\frac{1}{2}}\tfrac{\partial^{n+1}u}{\partial t^{n+1}}dt\right\} = \\ &= -\tfrac{2^n}{(2n-1)!!}\left[(\tau-t)^{n-\frac{1}{2}}\tfrac{\partial^n u}{\partial t^n}\right]_{t_R}^{t_L} + \tfrac{2^{+1}}{(2n+1)!!}\int_{t_R}^{t_L}(\tau-t)^{n-\frac{1}{2}}\tfrac{\partial^{n+1}u}{\partial t^{n+1}}dt = \\ &= -J_n + I_{n+1}. \end{aligned}$$

Hence,

$$I = -\sum_{j=0}^{n-1} J_j - J_n + I_{n+1} = -\sum_{j=0}^{n} J_j + I_{n+1}$$

□

LEMMA 5.2.

$$I = -\sum_{j=0}^{k-1} J_j + O(\Delta t)^{k+\frac{1}{2}}$$

Proof. By Lemma 5.1

$$I = -\sum_{j=0}^{k-1} J_j + I_k$$

using the Mean Values theorem for integrals we have,

$$I_k = \frac{2^k(t_L-t_R)}{(2k-1)!!}(\tau-\xi)^{k-\frac{1}{2}}\frac{\partial^k u}{\partial t^k}(\xi) = O(\Delta t)\cdot(\Delta t)^{k-\frac{1}{2}} = O(\Delta t)^{k+\frac{1}{2}}$$

□

REMARK 5.1. *The quadrature error for I_k is a function of the derivative of some order, say n, times $O(\Delta t)^{n+1}$ which is,*

$$O(\Delta t)^{k-\frac{1}{2}-n}(\Delta t)^{n+1} = O(\Delta t)^{k+\frac{1}{2}}$$

However, this is the order of I_k itself (Lemma 5.2). Therefore, there is no point for approximating I_k by any quadrature, and to get better accuracy one should take k large enough and approximate I by $-\sum_{j=0}^{k-1} J_j$.

We consider only the first two cases:

1. For $O(\Delta t)^{\frac{3}{2}}$ $(k=1)$ order of approximation, we have

$$I \approx -J_0 = -2\left[(\tau-t)^{\frac{1}{2}}u\right]_{t_R}^{t_L} = 2\left[u(\xi,t_R)\sqrt{\tau-t_R} - u(\xi,t_L)\sqrt{\tau-t_L}\right]$$

i.e

$$I = 2\left[u(\xi,t_R)\sqrt{\tau-t_R} - u(\xi,t_L)\sqrt{\tau-t_L}\right] + O(\Delta t)^{\frac{3}{2}}$$

2. For $O(\Delta t)^{\frac{5}{2}}$ $(k=1)$ order of approximation we have

$$I \approx -J_0 - J_1$$

$$J_0 = 2\left[-u(\xi,t_R)\sqrt{\tau-t_R} + u(\xi,t_L)\sqrt{\tau-t_L}\right]$$

$$J_1 = \frac{4}{3}\left[(\tau-t)^{\frac{3}{2}}\frac{\partial u}{\partial t}\right]_{t_R}^{t_L} = \frac{4}{3}\left[(\tau-t)^{\frac{3}{2}}\frac{\partial u}{\partial t}(t_L) - (\tau-t_R)^{\frac{3}{2}}\frac{\partial u}{\partial t}(t_R)\right]$$

so that

$$I = 2\left[u(\xi,t_R)\sqrt{\tau-t_R} - u(\xi,t_L)\sqrt{\tau-t_L}\right] + \frac{4}{3}\left[(\tau-t_R)^{\frac{3}{2}}\frac{\partial u}{\partial t}(\xi,t_R) - (\tau-t_L)^{\frac{3}{2}}\frac{\partial u}{\partial t}(\xi,t_L)\right] + O(\Delta t)^{\frac{5}{2}}$$

Let

$$L = \frac{\partial}{\partial x} I$$

1. Using the $O(\Delta t)^{\frac{3}{2}}$ order of approximation we obtain

$$L = 2\left[\frac{\partial u}{\partial x}(\xi, t_R)\sqrt{\tau - t_R} - \frac{\partial u}{\partial x}(\xi, t_L)\sqrt{\tau - t_L}\right] + O(\Delta t)^{\frac{3}{2}}$$

We may use the following approximation:

$$\frac{\partial u}{\partial x}(x_i, t_R) = \frac{-u(x_{i+2}, t_R) + 4u(x_{i+1}, t_R) - 3u(x_i, t_R)}{2(\Delta x)} + O(\Delta x)^2$$

$$\frac{\partial u}{\partial x}(x_i, t_L) = \frac{u(x_{i-2}, t_L) - 4u(x_{i-1}, t_L) + 3u(x_i, t_L)}{2(\Delta x)} + O(\Delta x)^2$$

Hence,

$$\begin{aligned} L \;&= 2\left\{\left[\tfrac{-u(x_{i+2},t_R)+4u(x_{i+1},t_R)-3u(x_i,t_R)}{2(\Delta x)} + O(\Delta x)^2\right]\sqrt{\tau - t_R} - \right. \\ &\left. - \left[\tfrac{u(x_{i-2},t_L)-4u(x_{i-1},t_L)+3u(x_i,t_L)}{2(\Delta x)} + O(\Delta x)^2\right]\sqrt{\tau - t_L}\right\} + O(\Delta t)^{\frac{3}{2}} \end{aligned}$$

From which it follows that

$$\begin{aligned} L \;&= -\tfrac{1}{\Delta x}\left\{\left[u(x_{i+2}, t_R) + 4u(x_{i+1}, t_R) - 3u(x_i, t_R)\right]\sqrt{\tau - t_R} + \right. \\ &\left. + \left[u(x_{i-2}, t_L) - 4u(x_{i-1}, t_L) + 3u(x_i, t_L)\right]\sqrt{\tau - t_L}\right\} + O(\Delta t)^{\frac{1}{2}}(\Delta x)^2 + O(\Delta t)^{\frac{3}{2}} \end{aligned}$$

2. Using the $O(\Delta t)^{\frac{5}{2}}$ order of approximation we get

$$\begin{aligned} L \;&= 2\left[\tfrac{\partial u}{\partial x}(x_i, t_R)\sqrt{\tau - t_R} - \tfrac{\partial u}{\partial x}(x_i, t_L)\sqrt{\tau - t_L}\right] + \\ &+ \tfrac{4}{3}\left[(\tau - t_R)^{\frac{3}{2}}\tfrac{\partial^2 u}{\partial x^2}(x_i, t_R) - (\tau - t_L)^{\frac{3}{2}}\tfrac{\partial^2 u}{\partial x^2}(x_i, t_L)\right] + O(\Delta t)^{\frac{5}{2}} \end{aligned}$$

For $\frac{\partial u}{\partial x}$ we use the previous finite-difference one-sided approximation which is $O(\Delta x)^2$. For $\frac{\partial^2 u}{\partial x^2}$ we use the following one-sided $O(\Delta x)^2$ approximation:

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_R) = Fu_j = \frac{-u_{j+6} + 12u_{j+5} - 57u_{j+4} + 136u_{j+3} - 171u_{j+2} + 108u_{j+1} - 27u_j}{8(\Delta x)^3}$$

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_L) = Bu_j = \frac{u_{j-6} - 12u_{j-5} + 57u_{j-4} - 136u_{j-3} + 171u_{j-2} - 108u_{j-1} + 27u_j}{8(\Delta x)^3}$$

So that the overall approximation of L is $O(\Delta t)^{\frac{5}{2}} + O(\Delta x)^2$.

References

[1] D. Amitai, A. Averbuch, M. Israeli, and S. Itzikowitz. *Parallel Adaptive and Time-Stabilizing Schemes for Constant-Coefficient Parabolic PDEs*, Comp. & Math. with its Applications, 24, 10 (1992), pp. 33–53.

[2] 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 (1993), pp. 27–45.

[3] D. Amitai, A. Averbuch, S. Itzikowitz, and E. Turkel, *Asynchronous and Corrected-Asynchronous Numerical Solution of Parabolic PDEs on Parallel MIMD Multiprocessors*, Numerical Algorithms, 6 , 10, III-IV (1994), pp. 275–296.

[4] D. Amitai, A. Averbuch, M. Israeli and S. Itzikowitz, *On Parallel Asynchronous High-Order Solutions of parabolic PDEs*, Numerical Algorithms, 12 (1996), pp. 159–192.

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

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

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

[8] P. J. Davis and P. Rabinowitz. *Numerical Integration.* Blaisdell, Waltham, Mass., 1967.

[9] 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 (1991), pp. 63–71.

[10] J. F. Epperson. *On the Use of Green's Functions for Approximating Nonlinear Parabolic PDEs*, Appl. Math. Lett., 2, 3 (1989), pp. 293–296.

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

[12] G. E. Forsythe and W. R. Wasow. *Finite Difference Methods for Partial Differential Equations*, Wiley, New-York, 1960.

[13] A. R. Mitchell. *Computational Methods in Partial Differential Equations.* Wiley, London, 1969.

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

[15] J. M. Ortega and R. G. Voigt, *Solution of Parallel Differential Equations on Vector and Parallel Computers*, SIAM Review, 27, 2 (1985), pp. 149–239.

[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. *Integration of Equations of Parabolic Type by the Method of Nets*, Pergamon Press, New-York, 1964.

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

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