Asynchronous Finite Difference Scheme for Partial Differential Equation

A REPORT SUBMITTED BY

KUMAR SAURABH
(MA14M004)
M.TECH
(INDUSTRIAL MATHEMATICS AND SCIENTIFIC COMPUTING)
Indian Institute of Technology, Madras

UNDER THE GUIDANCE OF

PROF. DR. MARTIN FRANK
CCES, Department of Mathematics,
RWTH Aachen University, Germany

&

DR. S. SUNDAR
Department of Mathematics,
Indian Institute of Technology, Madras

1 Abstract

The numerical solution of partial differential equations in large-scale problems might require the usage of parallel computing. In this context, the computational domain is often split into smaller sub-domains that are handled by separate processors. Due to the non-local nature of the underlying problem, communication as well as synchronization between these processing elements (PEs) is required. Since the discretization by finite- difference schemes results in mostly simple arithmetic operations, the overhead due to communication and synchronization might become a bottleneck.Current trends in massively parallel computing systems suggest that the number of processing elements (PEs) used in simulations will continue to grow over time and known problem in this context is the overhead associated with communication and/or synchronization between PEs as well as idling due to load imbalances. Simulation at extreme levels of parallelism will then require an elimination, or at least a tight control of these overheads. While stability is conserved when these schemes are used asynchronously, accuracy is greatly degraded. Since message arrivals at PEs are essentially random processes, so is the behavior of the error. Within a statistical framework the average errors drop always to first-order regardless of the original scheme.A new scheme is proposed to robust the asynchrony. The analytical results are compared against numerical simulations.

2 Introduction

Many natural and engineered systems and processes can be accurately described by partial differential equations (PDEs).In a number of real applications, however, due to the complexity of the equations themselves as well as the geometrical aspects of the problem, analytical solutions are not known and numerical simulations provide invaluable information to understand these systems.

Even with numerical simulations, the complexity of systems at realistic conditions typically requires massive computational resources. In the last few decades, this computational power has been realized through increasing levels of parallelism. When a problem is decomposed into a number of processing elements (PEs), solving the PDE typically requires communication between PEs to compute spatial derivatives.As the number of PEs increases, this communication becomes more challenging . In fact, this may well be a major bottleneck at the next generation of computing systems which may comprise an extremely large number of PEs. At those extreme levels of parallelism, even small imbalances due to noise in otherwise perfectly balanced codes can represent enormous penalties as PEs idle waiting to receive data from other PEs. This is especially critical when, as commonly done, a global synchronization is imposed at each time step to finalize all communications as well as to obtain information to determine the time-step size unsteady calculations subjected to a so-called PEs. This is especially critical when, as commonly done, a global synchronization is imposed at each time step to finalize all communications as well as to obtain information to determine the time-step size in unsteady calculations subjected to a so-called Courant Friedrichs Lewy condition. condition. Thus, in order to take advantage of computational systems at extreme levels of parallelism, relaxing all (especially global)synchronizations is of prime necessity.

3 Concept

Finite difference schemes are one way to solve partial differential equations. Consider for example the one dimensional heat equation in its simplest case:

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2} \tag{3.1}$$

where $u(x,t)$ is the temperature at a spatial location $x \in [0,l]$ and time t and α is the thermal diffusivity of the medium. With N uniformly distributed grid points, Eq. (3.1) can be discretized using a second-order central difference in space and first-order forward difference in time to obtain a numerical scheme with well-know characteristics.

$$\frac{u_i^{n+1} - u_i^n}{\triangle t} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\triangle x^2} + \mathscr{O}(\triangle x^2, \triangle t) \tag{3.2}$$

where u_n^i is the temperature at a point $x = x_i$ at the time level n. Here $x_i = i\triangle x$ with $\triangle x = l/N$ being the grid spacing and i = 1 , . . . , N. The time step size is $\triangle t$. The last term represents the

order of the truncation error in time and space for this approximation. The above Eq. (3.2) can be re-written as

$$u_i^{n+1} = u_i^n + \frac{\alpha \triangle t}{\triangle x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{3.3}$$

which shows that to advance the solution from time level n to $n+1$, one needs the value of the function at neighbouring points at time level n. This is trivially implemented in a serial code where all the values u_i^n are available in the PE memory

The stencil corresponding to Eq (3.3) is shown as:

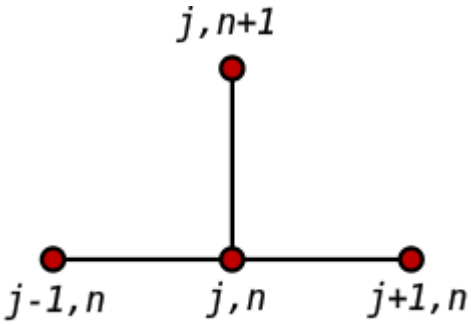


Figure 1: Standard explicit stencil from heat equation.

Since the error is $(\triangle x^2, \triangle t)$ we could increase the number of spatial and temporal nodes to increase the accuracy of the solution. The corresponding system can be written in matrix form as $u^{n+1} = Mu^n$ with corresponding matrix $M \in \mathbb{R}^{N_x \times N_x}$. Extending the problem into 3 dimensions the problem size grows since the linear system is of size $N_x^3 \times N_x^3$. Therefore, it might me helpful, to solve these systems on multiple processors. One common approach to split the workload onto multiple processors, is to decompose the computational domain accordingly to the physical domain which can be seen in Figure 2.

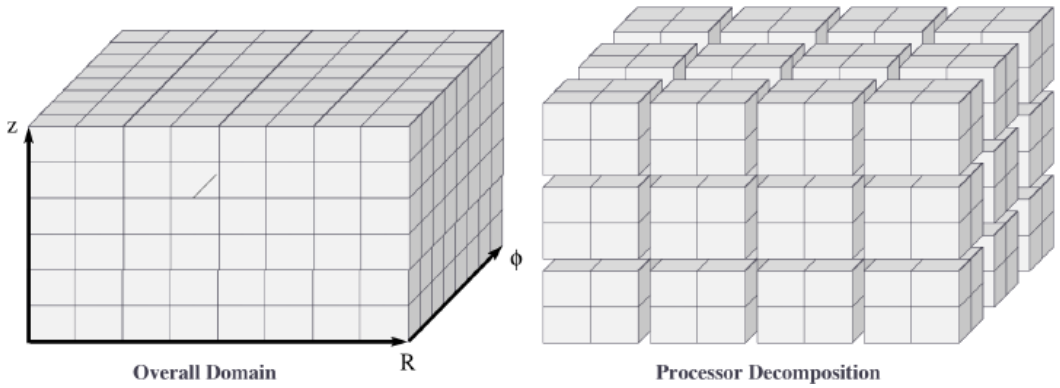


Figure 2: Domain Decomposition for the 3D case. Split every part of the domain into different PEs

The problem, that arises in the context of domain decomposition is the need for informa- tion exchange on the boundary elements of each PE.

3.1 Synchronous Finite Difference Scheme

Consider this problem in greater detail in the simplest case. In Figure 3 we we can see grid points of a one dimensional domain decomposition. Computations at interior points remain trivial as the required information is available locally to the PEs. Updating the values at grid points close to PE boundaries, however, require values from other PEs, that is, either u^n_{i+1} or u^n_{i-1} from the corresponding neighbouring PEs. These values are typically communicated over the network into buffer (or "ghost") arrays. Computations are halted until all PEs receive data in these so-called halo exchanges.

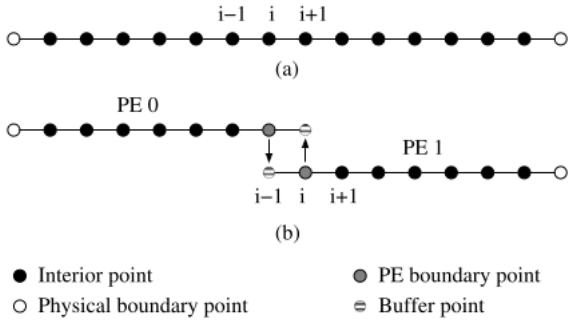


Figure 3: Discretized one-dimensional domain. (a) Domain in serial codes. (b) Same domain decomposed into two PEs.

One possible way to implement the data exchange between two PEs would be by using `MPI_Send` and `MPI_Receive`. We do not only have to consider the costs for the data transfer itself, but for the synchronization between all different PEs after each time layer. By the definition of our stencil, all PEs have to compute quantities for the time layer, because we would have asynchronicity between the data that has to be exchanged. Therefore, we have to make sure that we synchronize all PEs after the computation of one time layer, possibly by the use of `MPI_Barrier`. The implementation of `MPI_Barrier` inserts a barrier in the algorithm. A PE can not continue its computation, until all PEs have reached the barrier. Since not all PEs finish the computation of a time layer in the exact same time, this results in possible large idle times for multiple PEs. A typical code for the implementation of the scheme on different processor would look like:

Listing 1: Synchronous finite difference scheme

```
for(t = 0; t < final_T; t+=delta_t){
  for(i = 0; i < ne; i++){
  //Calculate according to the scheme
  //ne: number of elements on each processor
  .......
  }
  MPI_Send(boundary nodes);
  MPI_Recv(boundary nodes);
  MPI_Barrier(MPI_COMM_WORLD)

}
```

By using asynchronous finite-difference schemes, Donzis & Aditya propose a way to over- come this synchronization overhead and reduce the communication. Asynchronous schemes eliminate the synchronization after each time layer by allowing the different PEs to run asynchronous in time. The question arises, under which conditions these methods are still convergent and whether the accuracy remains the same.

3.2 Asynchronous Finite Difference Scheme

The problem of synchronization can be avoided by allowing all PEs to continue calculations regardless of the status of the messages that are to be received by the corresponding PEs. In the context of Figure 3, PE 1 is not required to wait for the most updated value u_{i-1}^{n}, but instead, can compute derivatives using $u_{i-1}^{\tilde{n}}$ where $\tilde{n}$ is the latest time level available to PE 1 at x_{i-1}. This modifies the finite difference equation (3.3) for points close to PEs boundaries. In particular, the expressions for the leftmost and rightmost grid points in each PE are, respectively:

$$u_i^{n+1} = u_i^n + \frac{\alpha \triangle t}{\triangle x^2} u_{i+1}^n - 2u_i^n + u_{i-1}^{\tilde{n}} \tag{3.4}$$

$$u_i^{n+1} = u_i^n + \frac{\alpha \triangle t}{\triangle x^2} u_{i+1}^{\tilde{n}} - 2u_i^n + u_{i-1}^n \tag{3.5}$$

Hereby, $\tilde{n}$ is the introduced asynchronicity. In the synchronous case, we have $n = \tilde{n}$, but now we allow $\tilde{n} = n, n-1, n-2, ..., n-(L-1)$ to be any of the prior $(L-1)$ time layers.The occurrence of a particular level for $\tilde{n}$ at a particular location and time depends on how fast the communications take place, which in turn depends on a number of factors like hardware, network topology, network traffic, message size, etc. some of which are unpredictable and turn the process into a random one. Indeed, it is known that communication times possess random characteristics in real systems. Since $\tilde{n}$ has to be modelled as a discrete random variable that takes the values between n and $n-(L-1)$,

let us rewrite $\tilde{n} = n - \tilde{k}$, then the random variable becomes $\tilde{k}$ with probabilities $p(\tilde{k})$ such that

$$\sum_{\tilde{k}=0}^{L-1} p(\tilde{k}) = 1 \tag{3.6}$$

4 Stability

The proof has been taken from the paper by Donzis and Konduri.

4.1 Synchronous Case

To prove the stability of the asynchronous case, let us start with the synchronous formulation. Eqn (3.3) can be extended to compute the value at all the nodes of u_i^{n+1} at all the nodes $i = 1, 2, ..., I$ for the time layer $n+1$. For a periodoc domain, we can explicitly write:

$$\begin{pmatrix} u_1^{n+1} \\ u_1^{n+1} \\ \vdots \\ u_I^{n+1} \end{pmatrix} = \begin{pmatrix} 1-2\sigma & \sigma & 0 & 0 & \dots & 0 & 0 & \sigma \\ \sigma & 1-2\sigma & \sigma & 0 & \dots & 0 & 0 & 0 \\ & & \ddots & & & & & \\ \sigma & 0 & 0 & \dots & 0 & 0 & \sigma & 1-2\sigma \end{pmatrix} \begin{pmatrix} u_1^n \\ u_1^n \\ \vdots \\ u_I^n \end{pmatrix} \tag{4.1}$$

or

$$u^{n+1} = Mu^n \tag{4.2}$$

with M being the matrix of Eqn. (4.1). Since M is time independent, we can recursively write:

$$u^{n+1} = Mu^n = M^2 u^{n-1} = ... = M^{n+1} u^0 \tag{4.3}$$

Thus stability results from $\|M\| \leq 1$. Let $\|.\|_\infty$ be the infinity norm, defined as $\|A\|_\infty = \max_i \sum_j |A_{ij}|$. Given the matrix M, above method is stable, if

$$|\sigma| + |1-2\sigma| + |\sigma| \leq 1 \tag{4.4}$$

which is fulfilled for $0 \leq \sigma \leq 1/2$. This result is known as **CFL** condition for the given problem and its discretizaion.

4.2 Asynchronous Case

The bound on the maximal delay is needed for the theoretical analysis and the existence of such a bound can be considered a reasonable assumption. For a more simpler case, let us assume that the

scheme is either synchronized or has a delay 1, i.e. $\tilde{n} = n$ or $n-1$. Furthermore we will assume that we only need data exchange for the rightmost boundary nodes. Additionally we will assume, that the number of PEs is the number of spatial points, which is no realisitic assumption, but does not influence the validity of the proof. Under the given assumtions, we can write:

$$\begin{pmatrix} u^{n+2} \\ u^{n+1} \end{pmatrix} = \tilde{C} \begin{pmatrix} A_0 & A_1 \\ I & 0 \end{pmatrix} \tag{4.5}$$

where $(u^{n+2}, u^{n+1})^T$ is of dimension $2I\,X\,I$ as well as $(u^{n+1}, u^n)^T$. The matrix C is of the size $2I\,X\,2I$ and defined as

$$C = \begin{pmatrix} A_0 & A_1 \\ I & 0 \end{pmatrix} \tag{4.6}$$

with the identity matrix I and

$$A_0 = \begin{pmatrix} (1-2\sigma) & (1-\tilde{k_2})\sigma & 0 & \dots & \sigma \\ \sigma & (1-2\sigma) & (1-\tilde{k_3})\sigma & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ (1-\tilde{k_1})\sigma & 0 & \dots & \sigma & (1-2\sigma) \end{pmatrix} \tag{4.7}$$

as well as

$$A_1 = \begin{pmatrix} 0 & \tilde{k_2}\sigma & 0 & \dots & 0 \\ 0 & 0 & \tilde{k_3}\sigma & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ \tilde{k_1} & 0 & \dots & 0 & 0 \end{pmatrix} \tag{4.8}$$

The $\tilde{k_1} \in \{0,1\}$ enables the switch between $\tilde{n} = n$ and $\tilde{n} = n-1$ Given this formulation and time dependency of $\tilde{C}$, Eqn (4.5) can be re-written as

$$\begin{pmatrix} u^{n+2} \\ u^{n+1} \end{pmatrix} = \begin{pmatrix} u^{n+1} \\ u^n \end{pmatrix} = \dots = \tilde{C}^{n+1} \begin{pmatrix} u^1 \\ u^0 \end{pmatrix} \tag{4.9}$$

Let us define $W^n = (u^{n+1}, u^n)^T$. For stability we want $\|W^n\| \leq \|W^0\|$ for a given norm. Thus the norm of C has to be bounded by unity. Considering the infinity norm as for the synchronous case, from the structure of C we get:

$$\|C\|_\infty \leq 1 \leftrightarrow |\sigma| + |1-2\sigma| + |(1-\tilde{k_i})\sigma| + |\tilde{k_1}\sigma| \leq 1 \tag{4.10}$$

Since $\tilde{k_i}$ is either 0 or 1, this simplifies to

$$|\sigma| + |1-2\sigma| + |\sigma| \leq 1 \tag{4.11}$$

which is same exactly the stability criterion for the synchronous case in Eq. (4.4). Therefore, in this simplified case, the asynchronous scheme is stable, if the initial scheme is stable.

4.2.1 General case

Due to the choice of infinity norm for our analysis of the scheme, only those rows that include delay (the corresponding grid points close to the boundaries) are relevant, because all other rows are bounded by unity as they are essentially synchronous.

Each finite difference stencil of size $2S+1$, that includes only two time layers can be written as:

$$u_i^{n+1} == \sum_{j=-S}^{S} c_j u_{i+j}^n \tag{4.12}$$

Therefore the stability criteria when using the infinity norm turns out to be:

$$\sum_{j=-S}^{S} |c_j| \leq 1 \tag{4.13}$$

which is a sufficient condition for stability.

We now introduce the asynchronicity by adding an additional sum over all possible delays from 0 to $L-1$. Since only one of these terms is non-zero, we use Kronecker delta to accomplish this. Let Kronecker delta be given by:

$$\delta_{k_i,m} = \begin{cases} 1, & \text{iff } k_i = m, \\ 0, & \text{iff } k_i \neq m. \end{cases} \tag{4.14}$$

We can now write:

$$u_i^{n+1} = \sum_{m=0}^{L-1} \sum_{j=-S}^{S} c_j \delta_{k_i,m} u_{i+j}^{n-m} \tag{4.15}$$

Generalising the formulation of Eq. (4.13) to all possible time layers we can deduce the stability criteria for the general asynchronous case, i.e. our scheme is stable, if

$$\sum_{m=0}^{L-1} \sum_{j=-S}^{S} |c_j \delta_{k_i,m}| = 1 \tag{4.16}$$

Since the Kronecker delta is either 0 or 1, and adds upto unity when summed over m we get

$$\begin{aligned} \sum_{m=0}^{L-1} \sum_{j=-S}^{S} |c_j \delta_{k_i,m}| &= \sum_{m=0}^{L-1} \sum_{j=-S}^{S} |c_j| \delta_{k_i,m} \\ &= \sum_{j=-S}^{S} |c_j| \sum_{m=0}^{L-1} \delta_{k_i,m} \\ &= \sum_{j=-S}^{S} |c_j| \end{aligned} \tag{4.17}$$

which is the same condition for stability of synchronous scheme as in Eq. (4.13). Therefore asynchronous scheme is stable, if the synchronous scheme is stable.

The theoretical analysis, showed that the question of convergence does not depend on the statistics of the delay $\tilde{k}_i$ nor on the number of PEs. The first fact is important, since the $\tilde{k}_i$ are in principle random variables and assumptions on their statistics can not be made in the general case. The latter ensures, that the method is stable independent of the domain decomposition, as well of the underlying architecture. Furthermore, stability is kept when the problem is scaled to a finer grid decomposition and a higher number of PEs.

5 Consistency

By performing Taylor expansion, one can easily show, that for the synchronous case and by using the above stencil, the error between the correct solution and the approximated solution can is given by

$$E_i^n = -\frac{u_{tt}}{2}\triangle t + \frac{\alpha u_{xxxx}}{12}\triangle x^2 + O(\triangle t^2, \triangle x^4) \tag{5.1}$$

For $(\triangle t, \triangle x) \to 0$ the error vanishes, therefore the numerical approximation is consistent with the analytical problem with first order convergence in time and second order in space.

Consider for the asynchronous case. For arbitrary delay k at a grid point $i+1$, we deduce with the help of Taylor expansion:

$$\tilde{E}_i^n|_{\tilde{k}_{i+1}} = -\frac{u_{tt}}{2}\triangle t + \frac{\alpha u_{xxxx}}{12}\triangle x^2 - \alpha k u_t \frac{\triangle t}{\triangle x^2} + \alpha k u_{tx}\frac{\triangle t}{\triangle x} - \alpha k u_{txx}\frac{\triangle t}{2} + O(\triangle x^3, \triangle t^2, \triangle x^p \triangle t^q) \tag{5.2}$$

where the parameter p and q depends on the bound of the delay. By introducing the CFL number $r_\alpha = \alpha\triangle t/\triangle x^2$, we can rewrite the equation and get

$$\tilde{E}_i^n|_{\tilde{k}_{i+1}} = -\frac{r_\alpha u_{tt}}{2\alpha}\triangle x^2 + \frac{\alpha u_{xxxx}}{12}\triangle x^2 - r_\alpha k u_t + r_\alpha k u_{tx}\triangle x - r_\alpha k u_{txx}\frac{\triangle x^2}{2} + O(\triangle x^3, \triangle t^2, \triangle x^p \triangle t^q) \tag{5.3}$$

We see, that this scheme is therefore not consistent with the initial scheme, when r_α is kept constant. Since the error contains zeroth order contributions, the error does not vanish under grid refinement. However, this error analysis is only valid for nodes where two PEs work asynchronous. For the majority of points, this is not the case. It is therefore necessary to perform a statistical error analysis, that includes both, the statistics of the delays as well as the number of PEs.

In the following, let I_B denote the set of boundary nodes and I_I the set of interior nodes. That is, $i \in I_B$ if we need communication between PEs for the computation of u_i^{n+1} and $i \in I_I$ otherwise. Clearly I_B and I_I are disjoint. We introduce a spatial average $\langle f \rangle_S = 1/N_S \sum_{i\in S} f_i$ for a set S of size N_S. Furthermore let $\bar{f}$ denote the ensemble average over the statistical properties. The combination

of the spatial and ensemble average for a given time layer n is

$$\langle E \rangle = \frac{1}{N} \sum_{i=1}^{N} \bar{E}_i^n \tag{5.4}$$

since we only need to take an ensemble average over nodes that belong to I_B this is equivalent to

$$\langle E \rangle = \frac{1}{N} \left[\sum_{i \in I_I} E_i{}^n + \sum_{i \in I_B} \bar{E}_i^n \right] \tag{5.5}$$

From Eqn. 5.1, we know that

$$\begin{aligned}
\sum_{i \in I_I} E_i^n &\approx \sum_{i \in I_I} \left(-\frac{u_{tt}}{2} \triangle t + \frac{\alpha u_{xxxx}}{12} \triangle x^2 \right) && (5.6) \\
&= \sum_{i \in I_I} \left(-\frac{u_{tt}}{2\alpha} r_\alpha + \frac{\alpha u_{xxxx}}{12} \triangle x^2 \right) && (5.7) \\
&= \triangle x^2 \sum_{i \in I_I} K_S && (5.8) \\
&= \triangle x^2 N_I \langle K_S \rangle_{I_I} && (5.9)
\end{aligned}$$

The number of interior grid points for P PEs and a stencil of size S under the assumption of one-sided delay is given as or the computation of the second term in Eqn 5.5, we have to take into account the statistical nature of the delays. The final result than computed be:

$$\sum_{i \in I_B} \bar{E}_i^n \approx N_B \langle K_S \rangle_{I_B} \triangle x^2 + \left(-r_\alpha N_B \langle u_t \rangle_{I_B} + r_\alpha N_B \langle u_{xt} \rangle_{I_B} \triangle x - \frac{N_B}{2} \langle u_{txx} \rangle_{I_B} \right) \bar{\bar{k}} \tag{5.10}$$

For detail derivation one should refer the original paper. Thus overall statistical error is given by:

$$\langle E \rangle = \langle K_S \rangle \triangle x^2 + N_B \langle K_S \rangle_{I_B} \triangle x^2 + \left(-r_\alpha N_B \langle u_t \rangle_{I_B} + r_\alpha N_B \langle u_{xt} \rangle_{I_B} \triangle x - \frac{N_B}{2} \langle u_{txx} \rangle_{I_B} \right) \bar{\bar{k}} \tag{5.11}$$

For the synchronous case, $\bar{\bar{k}} = 0$ and we have second order consistency in space. As soon as we consider the asynchronous case $\triangle x \to 0$, we are left with

$$\langle E \rangle \approx -\frac{N_B}{N} \bar{\bar{k}} r_\alpha \langle u_t \rangle_{I_B} = -S \frac{P}{N} \bar{\bar{k}} \langle u_t \rangle_{I_B} \tag{5.12}$$

and the original scheme drops to first order consistency in space.

And therefore with all parameter kept constant, we get:

$$\langle E \rangle \in O(P \bar{\bar{k}} \triangle x) \tag{5.13}$$

6 Implementation and Numerical Result

The test problem for the simulation is advection-diffusion equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \alpha\frac{\partial^2 u}{\partial x^2} \tag{6.1}$$

with $u = u(t,x)$. Further we assume a periodic domain of length 2π and the initial condition are superimposed sinusoidal waves, i.e.

$$u(x,0) = \sum_{\kappa} A(\kappa) sin(\kappa x) \tag{6.2}$$

which has the analytical result:

$$u_a(x,t) = \sum_{\kappa} \exp^{(-\alpha\kappa^2 t)} A(\kappa) sin(\kappa(x-ct)) \tag{6.3}$$

which means that the waves are shifted by $-ct$ and damped by $\exp^{-\alpha\kappa^2 t}$.

The discretized form of advection diffusion for the interior nodes are:

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) + \frac{c}{2\triangle x}(u_{i+1}^n - u_{i-1}^n) = \frac{\alpha}{\triangle x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{6.4}$$

whereas for the left boundary nodes is:

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) + \frac{c}{2\triangle x}(u_{i+1}^n - u_{i-1}^{\tilde{n}}) = \frac{\alpha}{\triangle x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^{\tilde{n}}) \tag{6.5}$$

and for the right boundary nodes is:

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) + \frac{c}{2\triangle x}(u_{i+1}^{\tilde{n}} - u_{i-1}^n) = \frac{\alpha}{\triangle x^2}(u_{i+1}^{\tilde{n}} - 2u_i^n + u_{i-1}^n) \tag{6.6}$$

The parameter used for simulation are:

Listing 2: Parameter for numerical simulation

```
len = 2*pi; % length of domain
A = 1;
k = 2;
alpha = 1;
r_alpha = 0.1; % CFL
c = 1;
final_t = 0.08*len/c
```

where

$$r_\alpha = \frac{\alpha\triangle t}{\triangle x^2} \tag{6.7}$$

6.1 Matlab implementation

To reproduce the numerical results Matlab code was used to simulate the problem with artificially simulating the delay.

Listing 3: Matlab code for numerical simulation

```
function [ err ] = async( delay, N, PE )
% delay: Delay in time steps
% N: Number of Grid Points
% PE: Number of Processors
U_old = U_initial
while(t < final_t)
  %Calculate the value at the boundary points and store it in array bp
  %by taking the corresponding value from U_old.
  for proc = 1:PE
    for k = 1: (delay + 1)
      for i = 1:ne
         %ne: number of element in each processor
        %Calculate boundary nodes by using value from bp ...
        %and other nodes according to scheme
        U_new = Calculation from U_old values.
      end
      U_old = U_new;
    end
  end
end
end
```

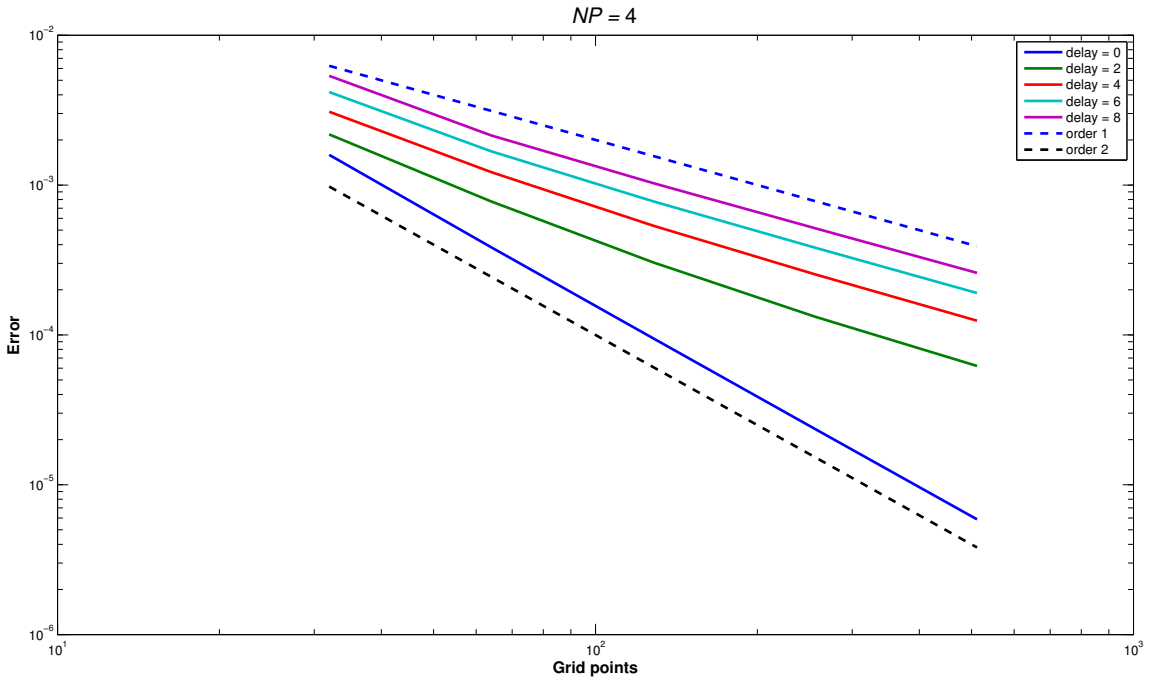
The above Matlab code is designed so that there is a communication between different processor after a certain `delay` number of time steps. `delay` $= 0$ corresponds to synchronous case.

6.1.1 Results from Matlab Code

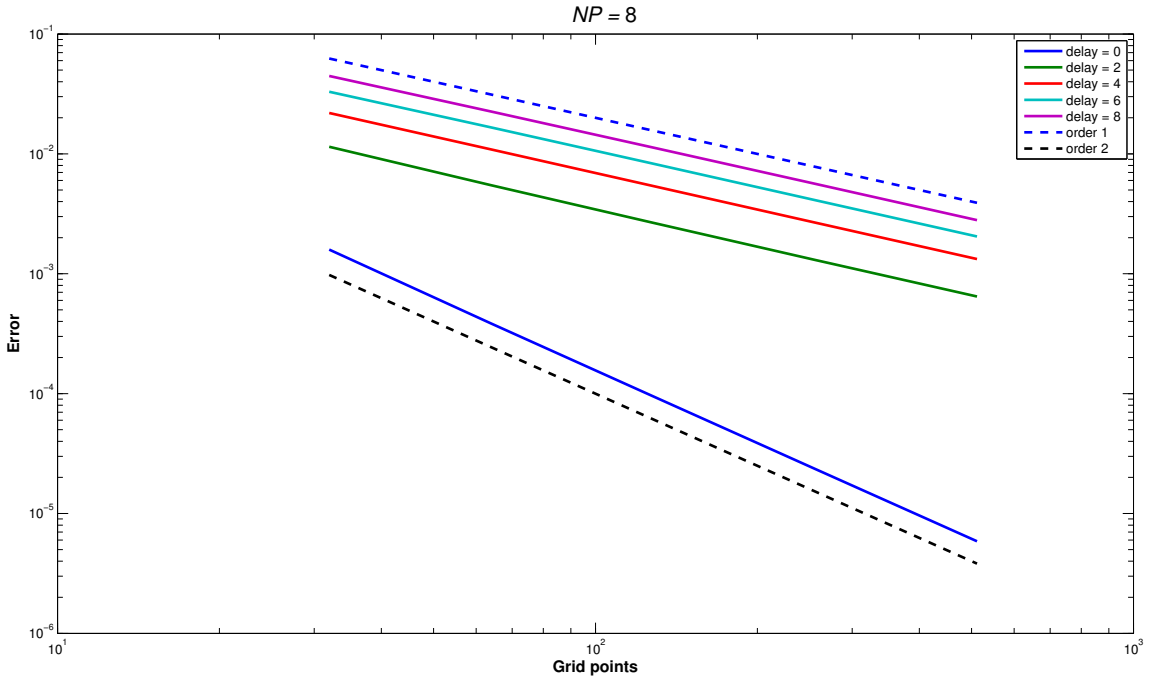
During the numerical experiments we vary the number of PEs, the statistics of delay as well as the number of grid points.

From log-log plot in Figure 4a and Figure 4b, we see the second order convergence in space for the synchronous simulation and drops to first order convergence for the delay irrespective of the number of processor or the delay. However larger the delay, larger is the error.

In Figure 5a and 5b, the number of grid point is kept constant at 256 and 512 respectively. For each case a simulation with different number of PEs is performed and vary the delay for each scenario from 0 to 8. As expected, the error is same for all number of PEs if the delay is 0 because we are essentially simulating the synchronous case. As soon as we increase the delay, we can observe two things. Firstly, the higher the delay, the larger the error as higher delay results in increasing the

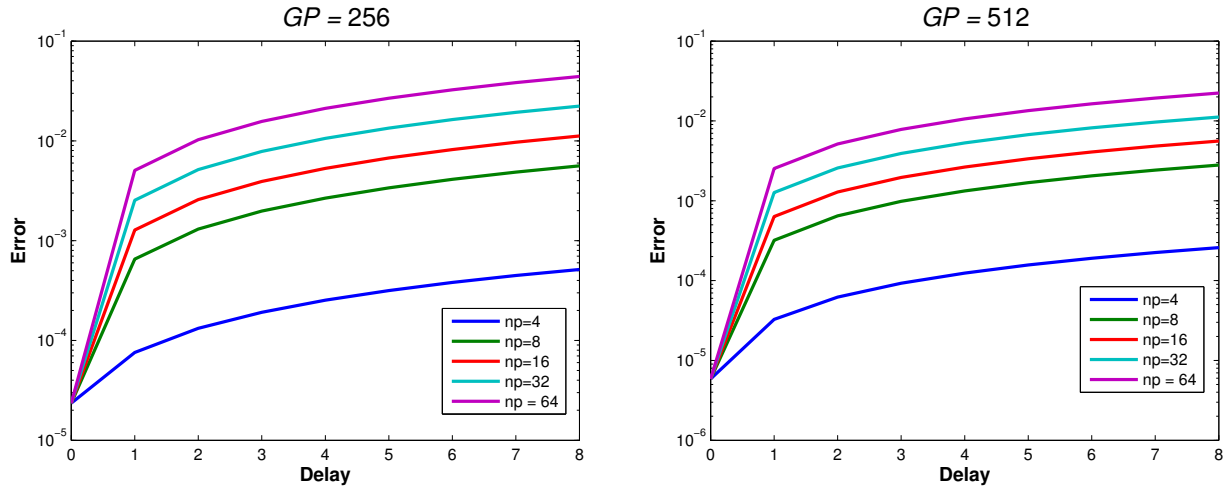


(a) Number of PEs = 4



(b) Number of PEs = 8

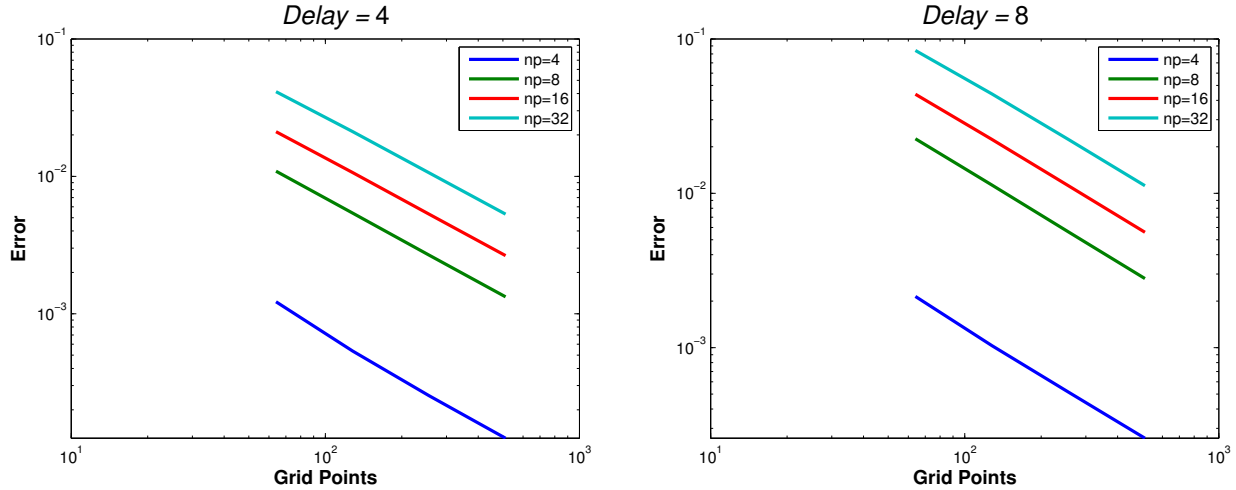
Figure 4: Synchronous vs Asynchronous Schemes



(a) Number of Grid Point = 256

(b) Number of Grid Point = 512

Figure 5: Influence of Delay on Grid Point



(a) Delay = 4

(b) Delay = 8

Figure 6: Influence of number of Grid Point for different number of PEs

number of asynchronous steps. Secondly the error is larger for a greater number of PEs which is in agreement with $\langle E \rangle \in O(P \triangle x)$

In Figures 6a and 6b the delay is fixed to 4 and 8 respectively and simulation is performed with varying number of PEs with varying number of grid points. Here again we observe the first degree convergence and error increase as we increase the number of PEs.

6.2 C++ implementation using MPI/OpenMP

Different approaches were used to simulate the asynchronous case using MPI and OpenMP. MPI uses the distributed memory structure whereas OpenMP uses the shared the memory architecture.

We have tried three different approaches to simulate the case. Brief algorithm are are mentioned in the following Section.

6.2.1 Master Work Paradigm Using MPI

In this approach, we used the master work paradigm approach is used. `Processor 0` acts as a "Master Processor" while the other processor carries the main part of the work. All the processor ("except the Master Processor") computes the values for all the nodes and send the boundary nodes to the master processor. The master processor then receives the values of the boundary nodes from the $processor_i$ and then sends the values that is needed by the $processor_i$ to carry out the further computation.

"Master processor" maintains a buffer `Vector<queue> bp` which contains the boundary points at the different time instants. The data-structure `queue` is capable to resize the array according to the need. This is needed as one processor may be more than one step ahead of the another process. Let's say $processor_i$ has finished computing at the time layer m and needed the value of the boundary nodes at time layer $m$to start the computation at time layer $m+1$. If the neighbouring processor is carrying out the computation at time layer $m+n, n > 1$, then it becomes necessary to store the value at different time layers. But once the value at a particular time layer is used, it is of no more use. So, we have incorporated `push` and `pop` function to the queue data-structure which take cares of the above mentioned scenerio.

Approach 1: Master Work Paradigm using MPI

```
if(rank == 0){  // Master Processor
  // Do initialization and distribute the corresponding boundary
     values at initial condition
  Vector<queue> bp[2*(numprocs - 1)]; // Vector of queues
     corresponding to each boundary points
  int count = 0;
  Vector<double> temp_bp[2*(numprocs - 1)];
  // Initialize temp_bp with initial condition at respective point
  while(1){
    MPI_Probe(,..,MPI_ANY_SOURCE,MPI_ANY_TAG,stats) ;
    if(stats.MPI_TAG == 'f'){
      count++;
    }
    else{
    // Calculate left and right boundary point index in Vector bp
       corresponding to stats.MPI_SOURCE
    if(bp[left_pop_index].get_size() > 0){
      temp_bp[left_pop_index] = bp[left_pop_index].pop();
    }
    if(bp[right_pop_index].get_size() > 0){
      temp_bp[right_pop_index] = bp[right_pop_index].pop();
    }
    // Copy the value of temp_bp[left_index] and temp_bp[right_index]
       to the a temporary array.
    MPI_Send(...);// Send the value to stats.MPI_SOURCE
    MPI_Receive(...);// Complete the receive process;
    // Calculate left and right boundary point index in Vector bp
       corresponding to stats.MPI_SOURCE
    bp[left_push_index].push(left_val);
    bp[right_push_index].push(right_val);
    }
    if (count == (numprocs - 1)){
      break;
    }
  }

}
else{
  // Accept the value of boundary points from rank 0 at the initial
     condition
  for (t = 0; t < final_t; t += delta_t)
  {
    // Calculate boundary nodes value.
```

```
    MPI_Send(...) // Send boundary node to rank 0 with tag = rank
    Calculate the value at the interior nodes.
    MPI_Recv(..) // Receive the value from rank 0
  }
  MPI_Send(..) // Send value with a special tag 'f' to rank 0
}
```

The time-line generated with the help of Vampir Trace following this approach is shown in figure (7).

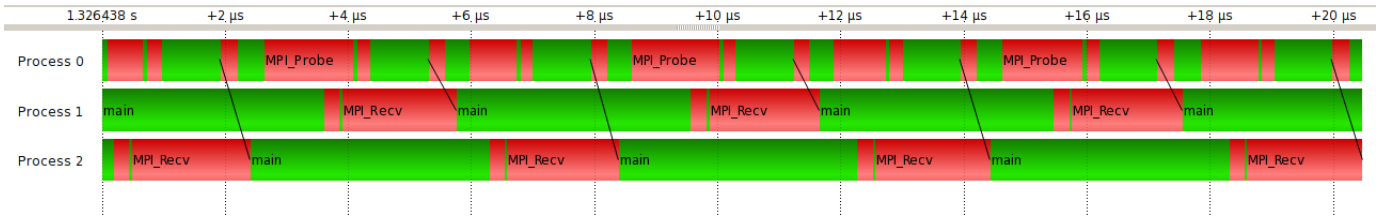


Figure 7: Vampir Trace with Grid Point 512.

6.2.2 Asynchronous Computation Using OMP

Approach 2:Asynchronous Computation Using Shared Memory

```
//Calculate for the initial Condition
Vector<queue> bp[2*(numprocs - 1)]; // Vector of queues corresponding
   to each boundary points in the shared section.
//Push all the boundary nodes to the Vector bp at the time level 0
#pragma omp parallel
{
  //Calculate left_push, right_push, left_pop ,right_pop index
  #pragma omp critical
  {
    if(bp[left_pop].get_size() > 0){
      left = bp[left_pop].pop();
    }
    if(bp[right_pop].get_size() > 0){
      right = bp[right_pop].pop();
    }
  }
  for(t = 0; t < final_t; t++){
    for(i = 0; i < ne; i++){
      // Carry out the calculation according to scheme
    }
```

```
    #pragma omp critical
    {
      if(bp[left_pop].get_size() > 0){
      left = bp[left_pop].pop();
    }
    if(bp[right_pop].get_size() > 0){
      right = bp[right_pop].pop();
    }
    bp[left_push].push(left_val);
    bp[right_push].push(right_val);
    }
  }
}
```

In the above multi-threaded application the `Vector<queue> bp` is in the shared region. This contains the values of the boundary points at different time layer as in the previous case. Since different processor may try to read and write the shared memory buffer `bp` at the same time, it forms a critical section.

In both of the approach mentioned in 6.2.1 and 6.2.2, the problem of scalability arises. If the program is carried out with a large amount of processors, the delay is going to increase. In 6.2.1 more processes will try to write to the master process due to which there will be more waiting time incorporated by the processor. In 6.2.2, with increase in the number of processor more processor will try to enter the critical section leaving the other process to wait till it completes the process.

6.2.3 MPI approach

In this approach there is a direct data transfer between the two processor without the involvement of any third processor. It is to be noted that we have checked the result just with two processes and result were better than the approach mentioned in 6.2.1 and 6.2.2 .At this moment we can make only an informal statement about the scalability about the algorithm but the actual implementation has yet to be done. The time-line generated with the help of Vampir Trace following this approach is shown in figure (8).

Approach 2:Asynchronous Computation Using MPI

```
// Carry out the initial condition calculation
if(rank == 0){
  step = 0;
  for(t = 0; t < final_t; t+= delta_t){
    step++;
    MPI_Irecv(...) // non-blocking receive for the nodes with tag =
       'step'
    // Calculate only for the boundary nodes
```

```
      MPI_Send(...) // writes in the buffer with tag = 'step'
      calculate() // calculate for the other nodes.
      MPI_Test(For request to complete);
      if(requset not completed){
        MPI_IProbe(..); // non-blocking probe with tag = step
        if(flag == true){ // Data in buffer is available
          MPI_Wait();
        }
      }
    }
}
else{
  // Same for the other processors
}
```

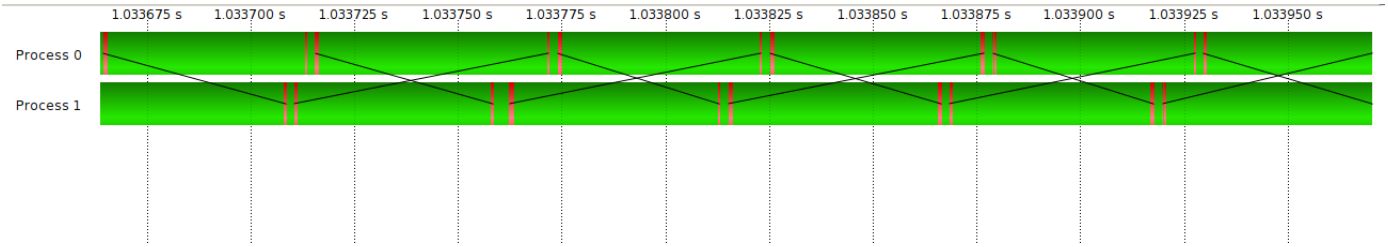
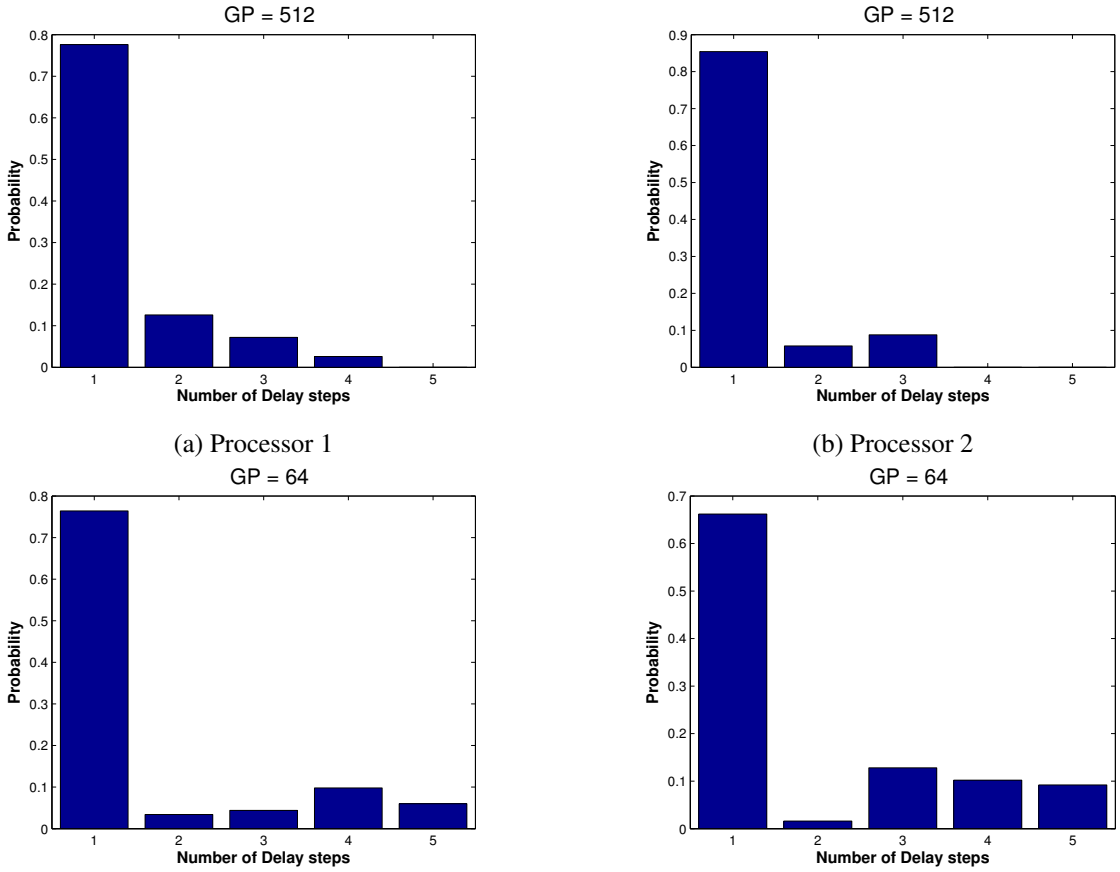


Figure 8: Vampir Trace with Grid Point 512.

The simulation is ran for grid size 64 and 512 with maximum allowable delay of 5 steps and without any maximum delay (i.e no waiting time for processor). The plot are presented in the Figure 9 and Figure 10. We can observe that we observe more delay for the run with grid size = 64 as compared to the run with grid size = 512. This was expected as the with Grid Size = 64 the communication is more frequent as compared to the run with grid size = 512. The above results are compared with 500 step runs.



(a) Processor 1

(b) Processor 2

(c) Processor 1

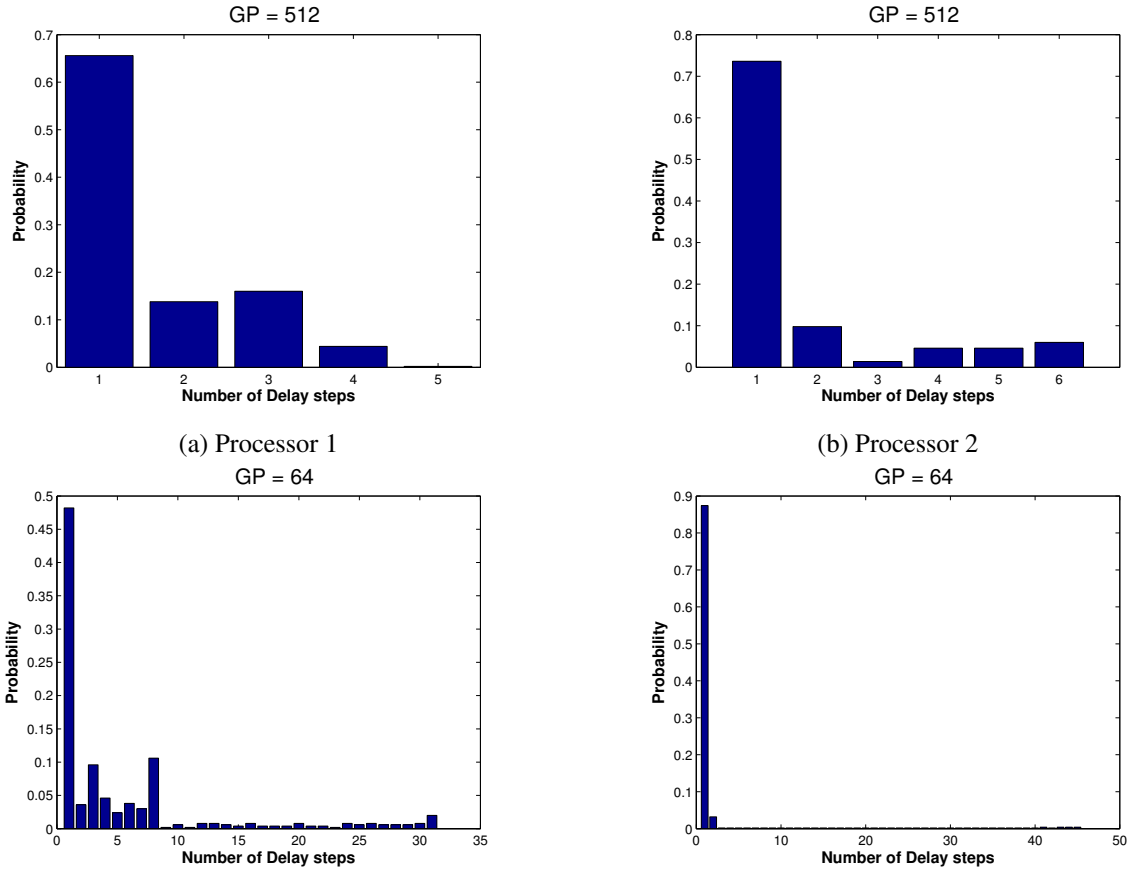
(d) Processor 2

Figure 9: Effect of Grid Size on Statistics of Delay. Max Allowable Delay = 5 steps

7 Conclusion

In the context of parallel computing, the problem of communication is becoming a bottleneck in the efficiency of algorithms. Domain decomposition for finite difference schemes for partial differential equations require communication at the domain boundaries between processors. Since especially for simple computations, the costs of communication outweigh those for arithmetic operations, it is desirable to reduce communication as much as possible. One way to accomplish this goal is by using asynchronous schemes. Instead of synchronization at each time layer, it is possible that processors compute independently from each other in an asynchronous fashion. The question remains, whether convergence and consistency behavior of the original method changes when it is no longer synchronous.

The requirement for synchronization at boundary nodes is removed and the computation can continue In section (4) we used the matrix method together with the inifinity norm, to prove, that the asynchronous scheme is stable, if the original scheme is stable. This result was independent of the characteristics of the delay, the number of PEs and the way, the domain is decomposed. In section



(a) Processor 1

(b) Processor 2

(c) Processor 1

(d) Processor 2

Figure 10: Effect of Grid Size on Statistics of Delay. No Waiting Time

(5) we derived an averaged error description for the consistency of the asynchronous method. We were able to show, that any higher order method drops to first order consistency in the asynchronous case. The averaged error description is required due to the statistical nature of the delays, as well as the spatial position of the boundary nodes. We derived $\langle E \rangle \in O(\tilde{\tilde{k}} P \triangle x)$ which shows, that the error is of first order in space and proportional to the number of PEs and proportional to the average delay which is in principle a random variable. Numerical experiments where reproduced on the problem description of Donzis and Aditya and are in agreement with the theoretical description of section (5), as well as with the original results.

8 Future Work

We would like to have a formal comparison of all the approaches mentioned in 6.2 and carry out the simulation with more than one processor. As we have observed that the accuracy drops down to first order in case of delay we would like to recover the order by modifying the current order so

that the accuracy remains second order.

References

[1] Diego A. Donzis and Konduri Aditya. Asynchronous Finite Difference Scheme for Partial Difference Equations *Journal of Computational Physics*. 274(0):370-392,2014

[2] Thomas Camminady. CES Seminar Paper on Asynchronous Finite Difference Scheme for Partial Difference Equation. January 9,2015

[3] MPICH , `http://www.mpich.org/`, 4 12 2015.