International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 8, Issue 2, March - April 2019 ISSN 2278-6856

Distributed Performance Improvement of Alternating Iterative Methods Running on Master-Worker Paradigm with MPI

Ewedafe Simon Uzezi[1], **Rio Hirowati Shariffudin**[2]

[1]Department of Computing, Faculty of Science and Technology,
The University of the West Indies, Mona Kingston 7, Jamaica

[2]Institute of Mathematical Sciences, Faculty of Science,
University of Malaya, 50603 Kuala Lumpur, Malaysia.

Abstract: ***In this paper we present methods that can improve numerical parallel implementation using a scalability-related measure called input file affinity measure on 2-D Telegraph Equation running on a master-worker paradigm. The implementation was carried out using Message Passing Interface (MPI). The Telegraph Equation was discretized using Alternating Direction Implicit (ADI), Iterative Alternating Direction Explicit Method by D'Yakonov (IADE-DY) and Mitchell-Fairweather Double Sweep Method (MF-DS). Parallel performance compare of these methods using the Input File Affinity Measure (I_{aff}) were experimentally evaluated and numerical results show their effectiveness and parallel performance. We show in this paper that, the input file affinity measure has scalability performance with less information to schedule tasks.***

Keywords: Parallel Performance, Input File Affinity, Master-Worker, 2-D Telegraph Equation, MPI.

1. INTRODUCTION

Computing infrastructures are reaching an unprecedented degree of complexity. First, parallel processing is coming to mainstream, because of the frequency and power consumption wall that leads to the design of multi-core processors. Second, there is a wide adoption of distributed processing technologies because of the deployment of the Internet and consequently large-scale grid and cloud infrastructures. In master-worker paradigm a master node is responsible for scheduling computation among the worker and collecting the results [5]. The master-worker paradigm has fundamental limitations: both communications from the master and contention in accessing file repositories may become bottlenecks to the overall scheduling scheme, causing scalability problems. Parallelization of Partial Differential Equations (PDEs) by time decomposition was first proposed by [22] following earlier efforts at space-time methods [15, 16, 30]. The application of the alternating iterative methods to solve problem of 2-D telegraph equations have shown that they need high computational power and communications. In the world of parallel computing MPI is the de facto standard for implementing programs on multiprocessors. To help with program development under a distributed computing environment a number of software tools have been developed MPI [13] is chosen here since it has a large user group.

An alternative and cost effective means of achieving a comparable performance is by way of distributed computing, using a system of processors loosely connected through a Local Area Network (LAN). Relevant data need to be passed from processor to processor through a message passing mechanism [12, 18, 27]. The telegraph equation is important for modeling several relevant problems such as wave propagation [24], and others. In this paper, a number of iterative methods are developed to solve the telegraph equations [8]. Some of these iterative schemes are employed in various parallel platforms [15, 16, 30]. Parallel algorithms have been implemented for the finite difference method [9, 10, 8], the discrete eigen functions method used in [1] and [27] used the AGE method on 1-D telegraph problem, but not implemented on parallel platform for parallel improvement. We present in this paper algorithms which have scalability performance comparable to other algorithms in several circumstances. We describe the design of our parallel system through I_{aff} for understanding parallel execution. Through several implementations and performance improvement analysis of the alternating iterative methods, we explore how to use the I_{aff} on master-slave paradigm for identifying parallel performance on 2-D telegraph equation. We assume the amount of work is increased exclusively by increasing the number of tasks. The reason for this is that it is relatively easy to increase the range of parameters to be analyzed in an application. On the order hand, in order to increase the amount of work related to each individual task, the input data for that task should be changed, which may change the nature of the problem to be processed by those tasks and it is not always feasible.

It is worth noting that some of the related works mentioned focus on the problem of efficient scheduling of the decomposition to execute on distributed architectures organized either as pure master-slave or hierarchical platforms, while lacking scalability analysis. We consider that the amount of computation associated with each task is fixed, each task depends on one or more input files for execution, and input file can be shared among tasks [14]. In this work we assume that both the size of the input files, the dependencies among input files and tasks are known. We also consider that the master node has access to a storage system which serves as the repository of all input and output files. This paper is organized as follows: Section 2 discusses previous research work. Section 2 presents the model for the 2-D telegraph equation and introduces the IADE-DY and DS-MF scheme. Section 3 describes the input file affinity measure. Section 4 describes the parallel implementation of the algorithms. Section 5 presents the numerical and experimental result of the schemes under consideration. Finally, section 6 presents our conclusions.

1.1 Previous research work

Parallel performance of algorithms has been studied in several papers during the last years [7]. Bag of Task (BoT) applications composed of independent tasks with file sharing have appeared in papers [8, 14, 19]. Their relevance has motivated the development of specialized environments which aim to facilitate the execution of large BoT applications on computational grids and clusters, such as the AppLes Parameter-Sweep Template (APST) [4]. Giersch et al., [14] proved theoretical limits for the computational complexity associated to the scheduling problem. In [19], they proposed an iterative scheduling approach that produces effective and efficient schedules, compared to the previous works in [5, 14]. However, for homogeneous platforms, the algorithms proposed in [19] can be considerably simplified, in a way that it becomes equivalent to algorithms proposed previously. Fabricio [11] analyzes the scalability of BoT applications running on master-slave platforms and proposed a scalability related measure. Eric [2] presents a detailed study on the scheduling of tasks in the parareal algorithm. It proposes two algorithms, one which uses a manager-worker paradigm with overlap of sequential and parallel phases, and a second that is completely distributed. Hinde et al. [17], proposed a generic approach to embed the master-worker paradigm into software component models and describes how this generic approach can be implemented within an existing software component model. On the numerical computing part, [29] went in search of numerical consistency in parallel programming and presented methods that can drastically improve numerical consistency for parallel calculations across varying numbers of processors. In [9, 10], parallel implementation of 2-D telegraph equation using the SPMD technique and domain decomposition strategy have been researched and they presented a model that enhances overlap communication with computation to avoid unnecessary synchronization, thus yield significant speed up. On the parallel computing front, [28], has proposed a parallel ADI solver for linear array of processors. The ADI method in [23, 26] have been used for solving heat equations in 2D. Approach to solve the telegraphic equations using different numerical schemes has been treated in [8].

2. TELEGRAPH EQUATION

We consider the second order telegraph equation as given in [8]:

$$\frac{\partial^2 v}{\partial t^2}+a\frac{\partial v}{\partial t}=b\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)\quad 0\le x\le 1,\, 0\le y\le 1,\, t>0 \qquad (2.1)$$

extending the finite difference scheme on the telegraph equation of (3.1) becomes:

$$\frac{v_{i,j}^{n+1}-2v_{i,j}^{n}+v_{i,j}^{n-1}}{(\Delta t)^2}+a\frac{v_{i,j}^{n+1}-v_{i,j}^{n-1}}{2\Delta t}- b\left\{\frac{v_{i+1,j}^{n+1}-2v_{i,j}^{n+1}+v_{i-1,j}^{n+1}}{(\Delta x)^2}+\frac{v_{i,j+1}^{n+1}-2v_{i,j}^{n+1}+v_{i,j-1}^{n+1}}{(\Delta y)^2}\right\}=0 \qquad (2.2)$$

Although this simple implicit scheme is unconditionally stable, we need to solve a penta-diagonal system of algebraic equations at each time step. Therefore, the computational time is huge.

2.1 The 2-D ADI Method

In this section, we derive the 2-D ADI scheme of the simple implicit FDTD method by using a general ADI procedure extended to (2.1). Equation (2.4) can be rewritten as:

$$\left(I+\sum_{m=1}^{2}A_m\right)v_{i,j}^{n+1}-2C_o v_{i,j}^{n}+C_1 v_{i,j}^{n-1}=0 \qquad (2.3)$$

sub-iteration 1 is given by:

$$-\rho_x v_{i+1,j}^{n+1(1)}+(1+2\rho_x)v_{i,j}^{n+1(1)}-\rho_x v_{i-1,j}^{n+1(1)}= -(A_2)v_{i,j}^{n+1(*)}+(2C_o v_{i,j}^{n}-C_1 v_{i,j}^{n-1})\quad \forall i,j \qquad (2.4)$$

For various values of *i* and *j* , (2.4) can be written in a more compact matrix form at the (*k* + ½) time level as:

$$Av_j^{(k+1/2)}=f_k,\quad j=1,2,\ldots n. \qquad (2.5)$$

sub-iteration 2 is given by:

$$-\rho_y v_{i,j+1}^{n+1(2)}+(1+2\rho_y)v_{i,j}^{n+1(2)}-\rho_y v_{i,j-1}^{n+1(2)}= v_{i,j}^{n+1(1)}+A_2 v_{i,j}^{n+1(*)}.\quad \forall\, i,j \qquad (2.6)$$

For various values of *i* and *j* , (2.6) can be written in a more compact matrix form as:

$$Bv_i^{(k+1)}=g_{k+1/2},\qquad i=1,2,\ldots m \qquad (2.7)$$

this is a prediction of $v_{i,j}^{n+1}$ by the extrapolation method. Splitting by using an ADI procedure, we get a set of recursion relations as follows:

$$(I+A_1)v_{i,j}^{n+1(1)} = -(A_2)v_{i,j}^{n+1(*)} + (2C_o v_{i,j}^{n} - C_1 v_{i,j}^{n-1}) \quad (2.8)$$

$$(I+A_2)v_{i,j}^{n+1(2)} = v_{i,j}^{n+1(1)} + A_2 v_{i,j}^{n+1(*)} \quad (2.9)$$

where $v_{i,j}^{n+1(1)}$ is the intermediate solution and the desired solution is $v_{i,j}^{n+1} = v_{i,j}^{n+1(2)}$. Finally, expanding A_1 and A_2 on the left side of (2.8) and (2.9), we get the 2D ADI algorithm.

2.2 IADE-DY

The matrices derived from the discretization resulting to A (2.5) and B (2.7) is respectively tridiagonal of size (*mxm*) and (*nxn*). Hence, at each of the (*k* + ½) and (*k* + 1) time levels, these matrices can be decomposed into $G_1 + G_2 - \frac{1}{6}G_1G_2$, where G_1 *and* G_2 are lower and upper bidiagonal matrices. By carrying out the relevant multiplications, the following equations for computation at each of the intermediate levels are obtained:

(i) at the $(p+1/2)^{th}$ iterate,

$$u_1^{(p+1/2)} = \frac{1}{\hat{d}}(s_1 \hat{s} u_1^{(p)} + w_1 \hat{s} u_2^{(p)} + hf_1)$$

$$u_i^{(p+1/2)} = \frac{1}{\hat{d}}(-l_{i-1}u_{i-1}^{(p+1/2)} + v_{i-1}s_{i-1}u_{i-1}^{(p)} + (v_{i-1}w_{i-1} + s_i\hat{s})u_i^{(p)} + w_i\hat{s}u_{i+1}^{(p)} + hf_i),$$

$$i = 2,3,\ldots,m-1$$

$$u_m^{(p+1/2)} = \frac{1}{\hat{d}}(-l_{m-1}u_{m-1}^{(p+1/2)} + v_{m-1}s_{m-1}u_{m-1}^{(p)} + (v_{m-1}w_{m-1} + s_m\hat{s})u_m^{(p)} + hf_m)$$

(2.10)

(ii) at the $(p+1)^{th}$ iterate,

$$u_m^{(p+1)} = \frac{u_m^{(p+1/2)}}{d_m},$$

$$u_i^{(p+1)} = \frac{1}{d_i}(u_i^{(p+1/2)} - \hat{u}_i u_{i+1}^{(p+1)}), \quad (2.11)$$

$$where\ d_i = r + e_i,\ i = m-1, m-2, \ldots, 2, 1$$

The two-stage iterative procedure in the IADE-DY algorithm corresponds to sweeping through the mesh involving at each iterates the solution of an explicit equation.

2.3 DS-MF

The numerical representative of Eq. (2.8) and (2.9) using the Mitchell and Fairweather scheme is as follows:

$$\left(1-\frac{1}{2}\left(\rho_x-\frac{1}{6}\right)A_1\right)v_{i,j}^{n+1(1)} = \left(1+\frac{1}{2}\left(\rho_y+\frac{1}{6}\right)A_2\right)v_{i,j}^{n+1(*)} + \left(2C_o v_{i,j}^{n} - C_1 v_{i,j}^{n-1}\right) \quad (2.2)$$

$$\left(1-\frac{1}{2}\left(\rho_y-\frac{1}{6}\right)A_2\right)v_{i,j}^{n+1(2)} = \left(1+\frac{1}{2}\left(\rho_x+\frac{1}{6}\right)A_1\right)v_{i,j}^{n+1(1)} + A_2 v_{i,j}^{n+1(*)} \quad (2.13)$$

The horizontal sweep (2.12) and the vertical sweep (2.13) formulas can be manipulated and written in a compact matrix.

3 THE I$_{AFF}$

With reference to [11], we introduce the I$_{aff}$. First, we describe a simplified execution model that classifies several issues related with the execution of the telegraphic equation on the master-worker paradigm. Typically, each task goes through three phrases during execution of a parameter-sweep application: (1) an initialization phase, where the necessary files are sent from the master to the slave node and the task is started. The duration of this phase is equal to t$_{init}$. Note that this phase includes the overhead incurred by the master to initiate a data transfer to a slave, (2) a computational phase, where the task processes the parameter file at the slave node and produces an output file. The duration of this phase is equal to t$_{comp}$. Any additional overhead related to the reception of input files by a worker node is also included in this phase and (3) a completion phase, where the output file is sent back to the master and the master task is completed. The duration of this phase is equal to t$_{end}$. This phase may require some processing at the master, mainly related to writing out files to the repository. Since this writing may be deferred until the disk is available, we assume that this processing time is negligible. Therefore, the initialization phase of one slave can occur concurrently with the completion phase of another slave node. Given these three phases, the total execution time of a task is equal to

$$t_{total} = t_{init} + t_{comp} + t_{end} \quad (3.1)$$

as the machine model, we consider a cluster composed of *P + 1* processors. For the rest of this paper we assume that *T* » *P*. One processor is the master and the other processors are workers. Communication among master and workers is carried out through Ethernet and master can only send files through the network to a single worker at a given time. We assume the communication link is full-duplex, i.e., the master can receive an output file from a worker at the same time it sends an input file to another worker. We also assume that communication computation begins as soon as the input files are completely received. This assumption is coherent to the master-worker paradigm that is currently available in [14].

For the sake of simplicity and without loss of generality, we consider in this section that there is no contention related to the transmission of output files from worker nodes to the master. Indeed, it is possible to merge the computation phase with the completion phase without affecting the results of this section. Therefore, we merge both phases as t'_{comp} in the equations that follow. A worker is idle when it is not involved with the execution of any of the three phases of a task. For the results below, the task model is composed of *T* heterogeneous tasks. All tasks and files have the same size and each task depends upon a single non-shared file. Note that the problem of scheduling an application where each task depends upon a single non-shared file, all tasks and files have the same size and the master-worker paradigm is heterogeneous, has polynomial complexity [14]. These assumptions are considered in the analysis that follows. We define the effective number of processors P_{eff} as the maximum number of workers needed to run an application with no idle periods on any worker processor. Taking into account the task and platform models described in this section, a processor may have idle periods if:

$$t'_{comp} < (P-1)t_{init} \qquad (3.2)$$

P_{eff} is then given by the following equation:

$$P_{eff} = \left\lfloor \frac{t'_{comp}}{t_{init}} + 1 \right\rfloor \qquad (3.3)$$

the total number of tasks to be executed on a processor is at most

$$M = \left\lceil \frac{T}{P} \right\rceil \qquad (3.4)$$

for a platform with P_{eff} processors, the upper bound for the total execution time (makespan) will be

$$\lceil t_{makespan} \rceil = M(t_{init} + t'_{comp}) + (P-1)t_{init} \qquad (3.5)$$

the second term in the right hand side of Eq. (3.5) shows the time needed to start the first (P – 1) tasks in the other P – 1 processors. If we have a platform where the number of processors is larger than P_{eff} the overall makespan is dominated by communication times between the master and the workers. We then have:

$$\lceil t_{makespan} \rceil = MPt_{init} + t'_{comp} \qquad (3.6)$$

the set of Eqs. (3.2) – (3.6) will be considered in subsequent sections of this paper. It is worth noting that Eq. (3.5) is valid when workers are constantly busy, either performing computation or

communication. Eq. (3.6) is applicable when workers have idle periods, i.e., are not performing either computation or communication. Eq. (3.2) occurs mainly in two cases:

(1) For very large platforms (P large).

(2) For applications with small $\frac{t_{comp}}{t_{init}}$ ratio, such as fine-grain applications.

In order to measure the degree of affinity of a set of tasks concerning their input files, we introduce the concept of input file affinity. Given a set of *G* of tasks, composed of *K* tasks, $G = \{T_1, T_2, \ldots, T_k\}$, and the set *F* of the *Y* input files needed by the tasks belonging to group *G*, $F = \{f_1, f_2, \ldots, f_y\}$, we define I_{aff} as follows:

$$I_{aff}(G) = \frac{\sum_{i=1}^{y} (N-1)|f_i|}{\sum_{i=1}^{y} N_i |f_i|} \qquad (3.7)$$

where $|f_i|$ is the size in bytes of file f_i and $N_i (1 \le N_i \le K)$ is the number of tasks in group G which have file f_i as an input file. The term $(N_i - 1)$ in the numerator of the above equation can be explained as follows: if N_i tasks share an input file f_i, that file may be sent only once (instead of N_i times) when the group of tasks is executed on a worker node. The potential reduction of the number of bytes transferred from a master node to a worker node considering only input file f_i is then $(N_i - 1)|f_i|$. Therefore, the input file affinity indicates the overall reduction of the amount of data that needs to be transferred to a worker node, when all tasks of a group are sent to one node. Note that $0 \le I_{aff} \le 1$. For the special case where all tasks share the same input file $I_{aff} = \frac{k-1}{k}$, where k is the number of tasks of a group.

4 PARALLEL IMPLEMENTATION

The implementation is done on Geranium Cadcam Cluster consisting of 48 Intel Pentium at 1.73GHZ and 0.99GB RAM. Communication is through a fast Ethernet of 100 MBits per seconds connected and running Linux. The cluster performance has high memory bandwidth with a message passing supported by MPI [13]. The program written in C provided access to MPI through calling MPI library routines. At each time-step we have to evaluate v^{n+1} values at $'lm'$ grid points, where $'l'$ is the number of grid points along x-axis. Suppose we are implementing this method on $R\times S$ mesh connected computer. Denote the workers by $P_{i1,j1}: i1=1,2,\ldots,R \text{ and } R<l,\ j1=1,2,\ldots,S \text{ and } S<M$. The workers P_{i1j1}, are connected as shown in Fig.4. Let $L_1=\left[\frac{1}{R}\right]$ and $M_1=\left[\frac{M}{S}\right]$ where $[\ ]$ is the smallest integer part. Divide the $'lm'$ grid points into $'RS'$ groups so that each group contains at most $(L_1+1)(M_1+1)$ grid points and at least L_1M_1 grid points. Denote these groups by

$G_{i1j1}: i1=1,2,\ldots,R,\ j1=1,2,\ldots,S$. Design G_{i1j1}, such that it contains the following grid points

$$G_{i1j1}=\left\{\begin{array}{r}(X_{(i1-1)+1},Y_{(j1-1)+j}): i=1,2,\cdots,L_1 \text{ or } L_i+1\\ j=1,2,\cdots,M_1 \text{ or } M_1+1\end{array}\right\}$$

Assign the group G_{i1j1}, to the workers $P_{i1,j1}: i_1=1,2,\ldots,R, \text{For}, j_1=1,2,\ldots,S.$ Each worker computes its assigned group $v_{i,j}^{n+1}$ values in the required number of sweeps. At the $(p+1/2)^{th}$ sweep the workers compute $v_{i,j}^{(p+1/2)th}$ values of its assigned groups. For the $(p+1/2)^{th}$ level the worker P_{i1j1} requires one value from the worker P_{i1-1j1} or P_{i1+1j1}, worker. In the $(p+1/2)^{th}$ level the communication between the workers is done row-wise. After communication between the workers is completed then each worker P_{ij} computes the $v_{i,j}^{p+1/2}$ values. For the $(p+1)^{th}$ sweep each worker P_{i1j1} requires one value from the P_{i1-1j1} or P_{i1+1j1} worker.

4.1 MPI Communication Service Design

MPI like most other network-oriented middleware services communicates data from one worker to another across a network. However, MPI's higher level of abstraction provides an easy-to-use interface more appropriate for distributing parallel computing applications. We focus our evaluation on [13] because MPI serves as an important foundation for a large group of applications. Conversely, MPI provides a wide variety of communication operations including both blocking and non-blocking sends and receives and collective operations such as broadcast and global reductions. We concentrate on basic message operations: blocking send, blocking receives, non-blocking send, and non-blocking receive. Note that MPI provides a rather comprehensive set of messaging operations. A blocking receives (*MPI_Recv)* returns when a message that matches its specification has been copied to the buffer. However, an alternative to blocking communication operations MPI provides non-blocking communication to allow an application to overlap communication and computation. This overlap improves application performance. In non-blocking communication, initialization and completion of communication operations are distinct. A non-blocking send has both a send start call (*MPI_Isend)* initializes the send operation and it return before the message is copied from the send buffer. The send complete call (*MPI_Wait)* completes the non-blocking send by verifying that the data has been copied from the send buffer. It is this separation of send start and send complete that provides the application with the opportunity to perform computations.

4.2 Speedup, Efficiency and Effectiveness

Speedup and efficiency give a measure of the improvement of performance experienced by an application when executed on a parallel system [7]. In traditional parallel systems it is widely define as:

$$S(N)=T(s)/T(N),\qquad E(N)=S(N)/N \qquad (4.1)$$

The total efficiency is usually decomposed into the following equations.

$$E(N)=E_{num}(N)E_{par}(N)E_{load}(N), \qquad (4.2)$$

E_{par} is the parallel efficiency which is defined as the ratio of CPU time taken on one worker to that on *N* workers. The parallel efficiency and the corresponding speedup are commonly written as follow

$$S_{par}(N)=T(1)/T(N),\quad E_{par}(N)=S_{par}(N)/N \qquad (4.3)$$

The CPU time for the parallel computations with *N* workers can be written as follows:

$$T(N) = T_m(N) + T_{sd}(N) + T_{sc}(N) \qquad (4.4)$$

generally,

$$T_m(N) = T_m(1), \quad T_{sd}(N) = T_{sd}(1), \quad T_{sc}(N) = T_{sc}(1)/N, \qquad (4.5)$$

therefore, the speedup can be written as:

$$S_{par}(N) = \frac{T(1)}{T(N)} = \frac{T_{ser}(1) + T_{sc}(1)}{T_{ser}(1) + T_{sc}(1)/N} < \frac{T_{ser}(1) + T_{sc}(1)}{T_{ser}(1)} \qquad (4.6)$$

The corresponding efficiency is given by:

$$Epar(N) = \frac{T(1)}{nT(N)} = \frac{Tser(1) + Tsc(1)}{nTser(1) + Tsc(1)} < \frac{Tser(1) + Tsc(1)}{nTser(1)} \qquad (4.7)$$

The total efficiency can be decomposed as follows:

$$E(N) = \frac{T_s^{N_o}(1)}{n.T_{B=B}^{N_1 L}(N)} = \frac{T_s^{N_o}(1)}{T_{B=1}^{N_o}(1)} \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \frac{T_{B=B}^{N_0}(1)}{T_{B=B}^{N_1}(1)} \frac{T_{B=B}^{N_1}(1)}{T_{B=B}^{N_1}(N)} \frac{T_{B=B}^{N_1}(N)}{T_{B=B}^{N_1 L}(N)}, \qquad (4.8)$$

where $T_{B=B}^{N_1}(n)$ has the same meaning as $T_{B=B}^{N_1 L}(n)$ except the idle time is not included. Comparing (6.5) and (6.2), we obtain:

$$E_{load}(N) = \frac{T_{B=B}^{N_1}(N)}{T_{B=B}^{N_1 L}}, E_{par}(n) = \frac{T_{B=B}^{N_1}(1)}{n.T_{B=B}^{N_1}(N)},$$
$$Enum(N) = \frac{T_s^{N_o}(1)}{T_{B=B}^{N_1}(1)} = \frac{T_s^{N_o}(1)}{T_{B=1}^{N_o}(1)} \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \frac{T_{B=B}^{N_o}(1)}{T_{B=B}^{N_1}(1)}, \qquad (4.9)$$

Therefore,

$$E_{num}(N) = E_{dd} \frac{N_o}{N_1}, \quad E_{dd} = \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \qquad (4.10)$$

we call (4.10) domain decomposition efficiency (DD), which includes the increase of CPU time induced by grid overlap at interfaces and the CPU time variation generated by DD techniques. Effectiveness is given by:

$$L_n = S_n / C_n \qquad (4.11)$$

where $C_n = nT_n$, T_1 is the execution time on a serial machine and T_n is the computing time on parallel machine with N processors. Hence, effectiveness can be written as:

$$L_n = S_n/(nT_n) = E_n/T_n = E_n S_n / T_1 \qquad (4.12)$$

Table 1 The wall time T_W, the master time T_M, the worker data time T_{SD}, the worker computational time T_{SC}, the total time T, the parallel speed-up S_{par} and the efficiency E_{par} for a mesh of 300x300, with B = 50 blocks and $Niter$ = 100 for PVM and MPI.

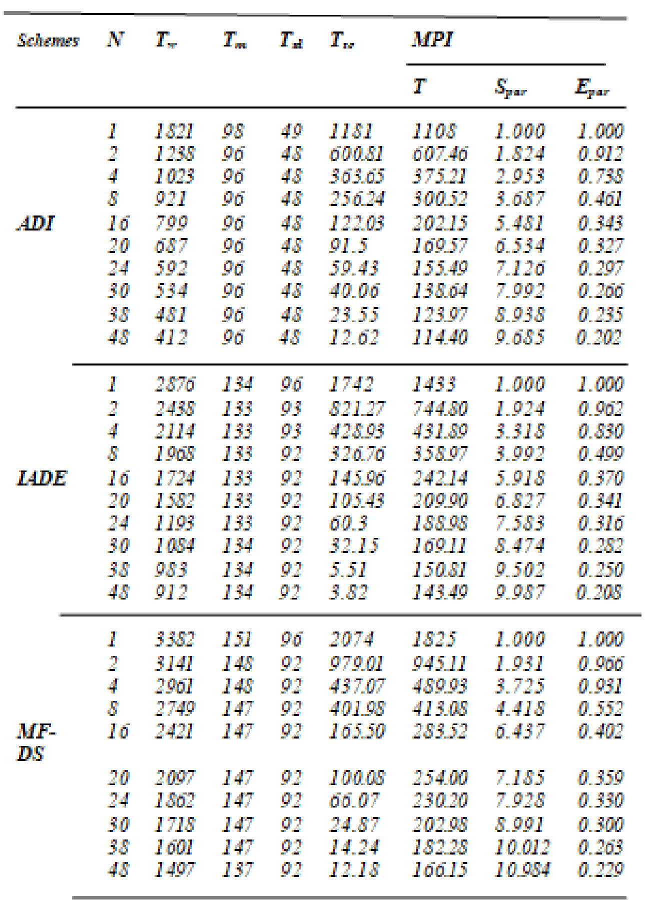
Schemes	*N*	T_w	T_m	T_{sd}	T_{sc}	*MPI*		
						T	S_{par}	E_{par}
	1	1345	82	21	829.31	824	1.000	1.000
	2	924	80	18	466.01	458.8	1.796	0.898
	4	836	80	18	299.07	313.55	2.628	0.657
	8	718	80	18	213.23	261.09	3.156	0.395
ADI	16	593	80	18	100.02	165.03	4.993	0.312
	20	526	80	18	80.84	139.24	5.918	0.296
	24	497	80	18	58.22	124.45	6.621	0.276
	30	438	80	18	42.75	109.6	7.518	0.251
	38	396	80	18	25.86	98.98	8.325	0.219
	48	357	80	18	8.77	92.45	8.913	0.186
	1	2341	98	40	1020	956	1.000	1.000
	2	2013	95	37	502.87	498.4	1.918	0.959
	4	1679	94	37	295.05	338.9	2.821	0.705
	8	1432	94	37	238.45	260.42	3.671	0.459
IADE	16	1391	94	37	103.94	165.03	5.793	0.362
	20	1135	94	37	75.31	156.26	6.118	0.306
	24	986	84	37	42	138.29	6.913	0.288
	30	821	84	37	20.19	123.82	7.721	0.257
	38	808	84	37	4.97	110.42	8.658	0.228
	48	775	84	37	2.4	101.42	9.426	0.196
	1	2968	116	53	1454	1328	1.000	1.000
	2	2575	114	51	700.6	689.87	1.925	0.963
	4	2388	114	51	377.63	439.59	3.021	0.755
	8	2011	114	51	308.04	340.95	3.895	0.487
MF-DS	16	1932	114	51	131.06	217.6	6.103	0.381
	20	1634	114	51	84.19	195.35	6.798	0.340
	24	1538	114	51	57.79	175.48	7.568	0.315
	30	1184	114	51	24.56	154.19	8.613	0.287
	38	1099	114	51	10.71	136.89	9.701	0.255
	48	987	114	31	0.11	123.43	10.759	0.224

5 Results and Discussion

5.1 Benchmark Problem

The application of the above mentioned algorithms were compared in terms of performance by simulating their executions in master-worker paradigm on several sizes. We assume a platform composed of variable number of heterogeneous processors. The solution domain was divided into rectangular blocks. The experiment is demonstrated on meshes of 200x200 and 300x300 for block sizes of 100 and 200 respectively, for MPI. Tables 1 - 5 show the various performance timing.

Table 2 The wall time T_W, the master time T_M, the worker data time T_{SD}, the worker computational time T_{SC}, the total time T, the parallel speed-up S_{par} and the efficiency E_{par} for a mesh of 300x300, with B = 100 blocks and $Niter$ = 100 MPI.



Schemes	N	T_w	T_m	T_{sd}	T_{sc}	MPI		
						T	S_{par}	E_{par}
ADI	1	1821	98	49	1181	1108	1.000	1.000
	2	1238	96	48	600.81	607.46	1.824	0.912
	4	1023	96	48	363.65	375.21	2.953	0.738
	8	921	96	48	256.24	300.52	3.687	0.461
	16	799	96	48	122.03	202.15	5.481	0.343
	20	687	96	48	91.5	169.57	6.534	0.327
	24	592	96	48	59.43	155.49	7.126	0.297
	30	534	96	48	40.06	138.64	7.992	0.266
	38	481	96	48	23.55	123.97	8.938	0.235
	48	412	96	48	12.62	114.40	9.685	0.202
IADE	1	2876	134	96	1742	1433	1.000	1.000
	2	2438	133	93	821.27	744.80	1.924	0.962
	4	2114	133	93	428.93	431.89	3.318	0.830
	8	1968	133	92	326.76	358.97	3.992	0.499
	16	1724	133	92	145.96	242.14	5.918	0.370
	20	1582	133	92	105.43	209.90	6.827	0.341
	24	1193	133	92	60.3	188.98	7.583	0.316
	30	1084	134	92	32.15	169.11	8.474	0.282
	38	983	134	92	5.51	150.81	9.502	0.250
	48	912	134	92	3.82	143.49	9.987	0.208
MF-DS	1	3382	151	96	2074	1825	1.000	1.000
	2	3141	148	92	979.01	945.11	1.931	0.966
	4	2961	148	92	437.07	489.93	3.725	0.931
	8	2749	147	92	401.98	413.08	4.418	0.552
	16	2421	147	92	165.50	283.52	6.437	0.402
	20	2097	147	92	100.08	254.00	7.185	0.359
	24	1862	147	92	66.07	230.20	7.928	0.330
	30	1718	147	92	24.87	202.98	8.991	0.300
	38	1601	147	92	14.24	182.28	10.012	0.263
	48	1497	137	92	12.18	166.15	10.984	0.229

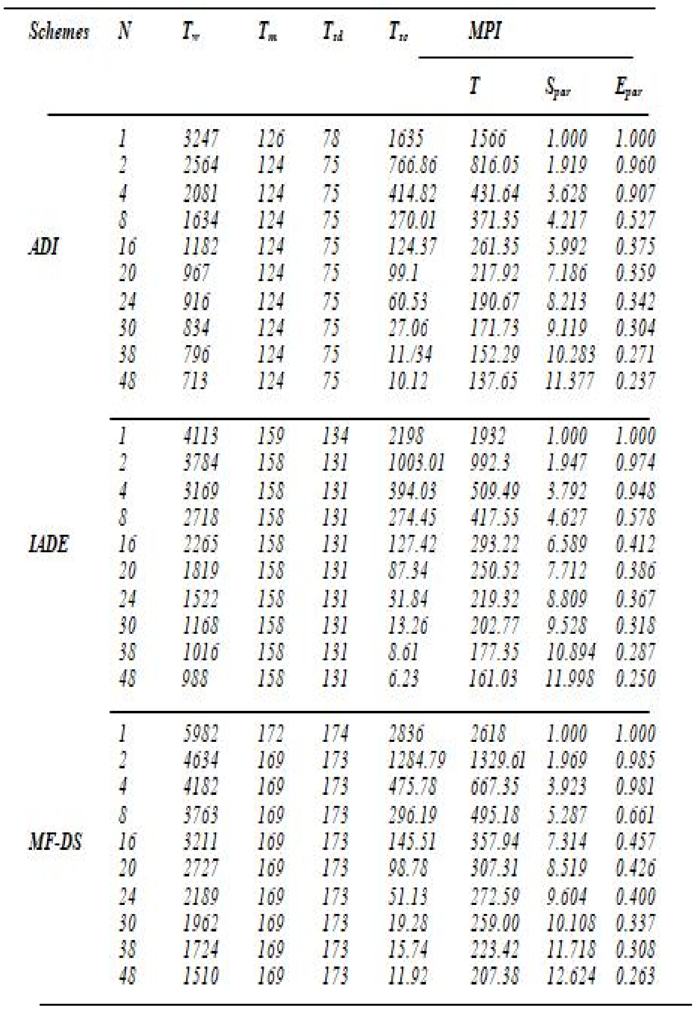
Consider the telegraph equation of the form:

$$\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}=\frac{\partial^2 U}{\partial t^2}+\frac{\partial U}{\partial t}+U \qquad (5.1)$$

5.2 Parallel Efficiency

To obtain a high efficiency, the worker computational time $T_{sc}(1)$ should be significantly larger than the serial time T_{ser}. The speed-up and efficiency obtained for various sizes of 200x200 to 300x300 are for various numbers of sub-domains; from *B* = 50 to 200 are listed in Tables 1 – 3 for MPI application. In these tables we listed the wall (elapsed) time for the master task, T_W, (this is necessarily greater than the maximum wall time returned by the slaves), the master CPU time, T_M, the average worker computational time, T_{SC} and the average worker data communication time T_{SD} all in seconds. The speed-up and efficiency versus the number of processors are shown in Fig.1(a,b) and Fig.2(a,b) respectively, with block number *B* as a parameter. The results show that the parallel efficiency increases with increasing grid size for given block number for MPI and decreases with the increasing block number for given grid size. Given other parameters the speed-up increases with the number of processors. When the number of processors is small the wall time decreases with the number of processors. As the number of processors become large the wall time increases with the number of processors as observed from the figures and table. Data communication at the end of every iteration is necessary in this strategy. Indeed, the updated values of the solution variables on full domain are multicast to all workers after each iteration since a worker can be assigned a different sub-domain under the pool-of-task paradigm. In Tables 1 – 3, the master time T_M is constant when the number of processors increases for a given grid size and number of sub-domains. The master program is responsible for (1) sending updated variables to worker (T_1), (2) assigning task to worker (T_2), (3) waiting for the worker to execute tasks (T_3), (4) receiving the results (T_4).

Table 3 The wall time T_W, the master time T_M, the worker data time T_{SD}, the worker computational time T_{SC}, the total time *T*, the parallel speed-up S_{par} and the efficiency E_{par} for a mesh of 300x300, with *B* = 200 blocks and *Niter* = 100 for MPI.



Schemes	N	T_w	T_m	T_{sd}	T_{sc}	MPI		
						T	S_{par}	E_{par}
ADI	1	3247	126	78	1635	1566	1.000	1.000
	2	2564	124	75	766.86	816.05	1.919	0.960
	4	2081	124	75	414.82	431.64	3.628	0.907
	8	1634	124	75	270.01	371.35	4.217	0.527
	16	1182	124	75	124.37	261.35	5.992	0.375
	20	967	124	75	99.1	217.92	7.186	0.359
	24	916	124	75	60.53	190.67	8.213	0.342
	30	834	124	75	27.06	171.73	9.119	0.304
	38	796	124	75	11./34	152.29	10.283	0.271
	48	713	124	75	10.12	137.65	11.377	0.237
IADE	1	4113	159	134	2198	1932	1.000	1.000
	2	3784	158	131	1003.01	992.3	1.947	0.974
	4	3169	158	131	394.03	509.49	3.792	0.948
	8	2718	158	131	274.45	417.55	4.627	0.578
	16	2265	158	131	127.42	293.22	6.589	0.412
	20	1819	158	131	87.34	250.52	7.712	0.386
	24	1522	158	131	31.84	219.32	8.809	0.367
	30	1168	158	131	13.26	202.77	9.528	0.318
	38	1016	158	131	8.61	177.35	10.894	0.287
	48	988	158	131	6.23	161.03	11.998	0.250
MF-DS	1	5982	172	174	2836	2618	1.000	1.000
	2	4634	169	173	1284.79	1329.61	1.969	0.985
	4	4182	169	173	475.78	667.35	3.923	0.981
	8	3763	169	173	296.19	495.18	5.287	0.661
	16	3211	169	173	145.51	357.94	7.314	0.457
	20	2727	169	173	98.78	307.31	8.519	0.426
	24	2189	169	173	51.13	272.59	9.604	0.400
	30	1962	169	173	19.28	259.00	10.108	0.337
	38	1724	169	173	15.74	223.42	11.718	0.308
	48	1510	169	173	11.92	207.38	12.624	0.263

Table4 Effectiveness of the various schemes with MPI for 300 x 300 mesh size

	N	*MPI T(s)*	*L_n*
	2	*816.05*	*0.118*
	8	*371.35*	*0.142*
ADI	*16*	*261.35*	*0.143*
	20	*217.92*	*0.165*
	30	*171.73*	*0.177*
	48	*137.65*	*0.172*
	2	*992.3*	*0.098*
IADE	*8*	*417.55*	*0.138*
	16	*293.22*	*0.140*
	20	*250.52*	*0.154*
	30	*202.77*	*0.157*
	48	*161.03*	*0.155*
	2	*1329.61*	*0.074*
MF-DS	*8*	*495.18*	*0.133*
	16	*357.94*	*0.128*
	20	*307.31*	*0.139*
	30	*259.00*	*0.130*
	48	*207.38*	*0.127*

Table5 Performance improvement of different schemes for 300x300 mesh size

N	***MPI***			***%***	***%***
	ADI	*IADE*	***MF-DS***	***ADI+ IADE***	***IADE+ MF-DS***
2	*816.05*	*992.3*	*1329.6*	*17.76*	*25.37*
8	*371.35*	*417.55*	*495.18*	*11.06*	*15.68*
16	*261.35*	*293.22*	*357.94*	*10.87*	*18.08*
20	*217.92*	*250.52*	*307.31*	*13.01*	*18.48*
30	*171.73*	*202.77*	*259.00*	*15.31*	*21.71*
48	*137.65*	*161.03*	*207.38*	*14.52*	*22.35*

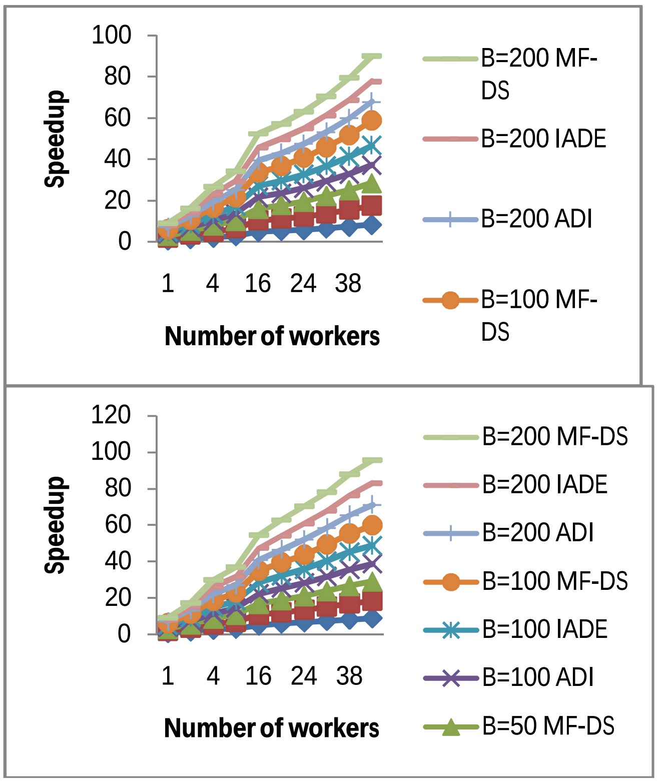


Fig.1 Speed-up versus the number of workers for various block sizes. **a** mesh 200x200 MPI, **b.** mesh 300x300 MPI

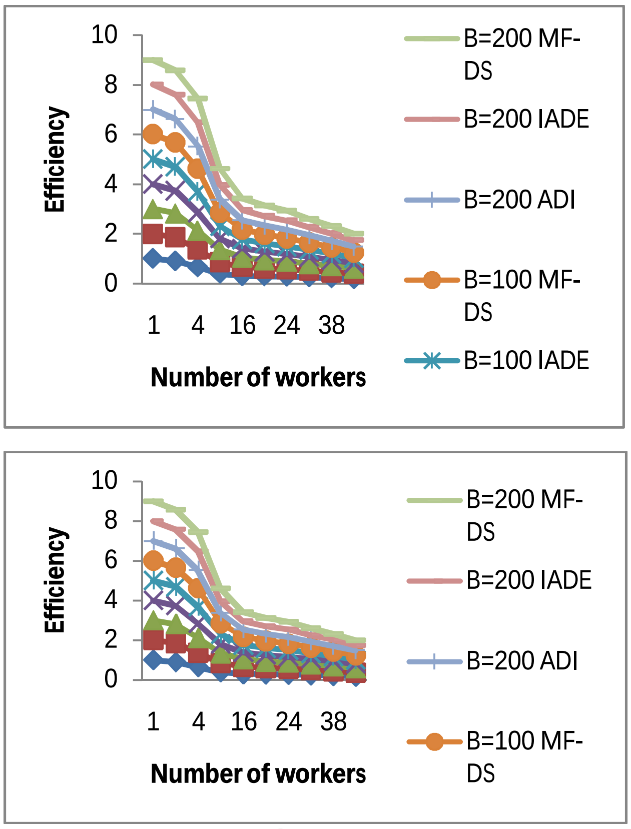


Fig.2 Parallel efficiency versus the number of workers for various block sizes. a mesh 200x200 MPI, b mesh 300x300 MPI

Table 7 shows the effectiveness of the various schemes with MPI. As the number of processor increases, the effectiveness of MF-DS scheme performs significantly better than the ADI. As the total number of processor increases, the bottleneck of parallel computers appears and the global reduction consumes a large part of time, we anticipate that the improvement in Table 5, will move to be significant. Fig.3 and Fig. 4 show the efficiency using MPI with varying block sizes, converges using the I_{aff} measure.

5.3 Numerical Efficiency

The numerical efficiency E_{num} includes the Domain Decomposition efficiency E_{DD} and convergence rate behavior N_o / N_1, as defined in Eq. (6.10). The DD efficiency $E_{dd} = T_{B=1}^{N_o}(1) / T_{B=B}^{N_o}(1)$ includes the increase of floating point operations induced by grid overlap at interfaces and the CPU time variation generated by DD techniques. In Tables 2 and 3, we listed the total CPU time distribution over various grid sizes and block numbers running on different processors for MPI. The DD efficiency E_{DD} is calculated and the result is shown in Fig3. Note that the DD efficiency can be greater than one, even with one processor. Fig.3 and Fig.4 show that the optimum number of sub-domains, which maximizes the DD efficiency E_{DD} increases with the grid size. The convergence rate behavior N_o / N_1, the ratio of the iteration number for the best sequential CPU time on one processor and the iteration number for the parallel CPU time on N processor describe the increase in the number of iterations required by the parallel method to achieve a specified accuracy as compared to the serial method. This increase is caused mainly by the deterioration in the rate of convergence with increasing number of processors and sub-domains. Because the best serial algorithm is not known generally, we take the existing parallel program running on one processor to replace it. Now the problem is that how the decomposition strategy affects the convergence rate? The results are summarized in Table 5 with Fig.5

It can be seen that N_o / N_1 decreases with increasing block number and increasing number of processors for given grid size. The larger the grid size, the higher the convergence rate. For a given block number, a higher convergence rate is obtained with less processors. This is because one processor may be responsible for a few sub-domains at each iteration. If some of this sub-domains share some common interfaces, the subsequent blocks to be computed will use the new updated boundary values, and therefore, an improved convergence rate results. The convergence rate is reduced when the block number is large. The reason for this is evident: the boundary conditions propagate to the interior domain in the serial computation after one iteration, but this is delayed in the parallel computation. In addition, the values of variables at the interfaces used in the current iteration are the previous values obtained in the last iteration. Therefore, the parallel algorithm is less "implicit" than the serial one. Despite these inherent short comes. A high efficiency is obtained for large scale problems.

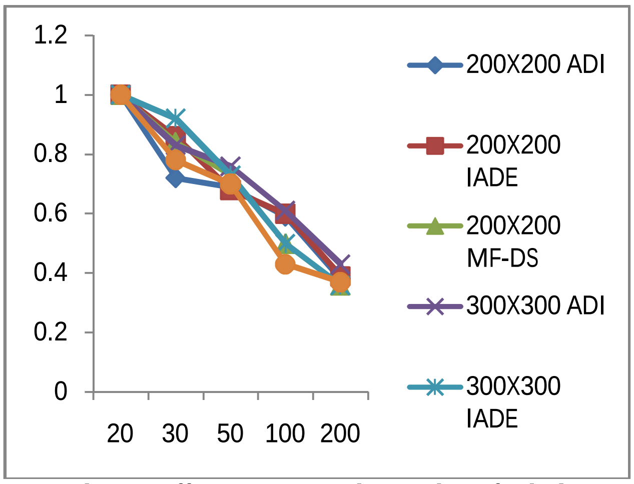


Fig.3 The DD efficiency versus the number of sub-domains for various blocks of MPI

Table6 The number of iteration to achieve given tolerance of 10^{-3} for a grid of 300x300 for MPI

Scheme	N	B=20	B=50	B=100	B=200
ADI	2	1818	3249	3611	1521
	4	1818	3568	3823	1768
	8	1818	3992	4094	1904
	16	1818	4327	4468	2112
	30	1818	4509	4713	2307
	48	1818	4813	4994	2548
IADE	2	2751	3976	4418	2154
	4	2751	4132	4632	2294
	8	2751	4378	4895	2478
	16	2751	4599	5109	2653
	30	2751	4708	5343	2801
	48	2751	4996	5599	2994
MF-DS	2	3101	4321	4852	3211
	4	3101	4582	5074	3528
	8	3101	4718	5362	3769
	16	3101	4992	5621	3928
	30	3101	5284	5895	4325
	48	3101	5476	6122	4518

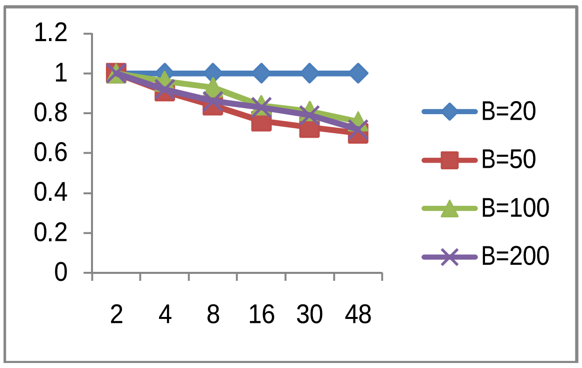


Fig.4 Convergence behavior with domain decomposition for mesh 200x200 MF-DS MPI

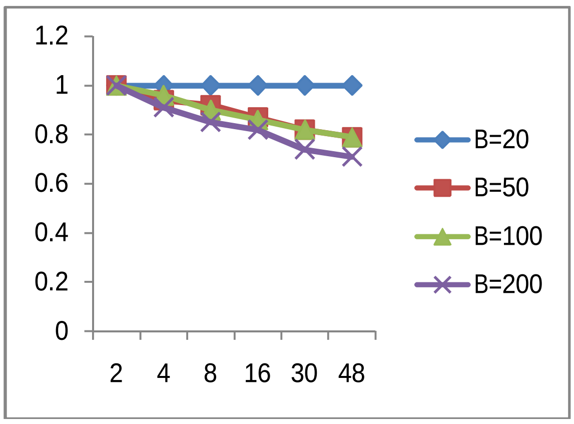


Fig.5 Convergence behavior with domain decomposition for mesh 300x300 MF-DS MPI

5.4 Total Efficiency

We implemented the serial computations on one of the processors, and calculated the total efficiencies. The total efficiency *E(N)* for grid sizes 200x200 and 300x300 have been showed respectively. From Eq. (6.8), we know that the total efficiency depend on N_o / N_1, E_{par} and DD efficiency E_{DD} since the load balancing is not the real problem here. For a given grid size and block number, the DD efficiency is constant. Thus, the variation of *E*(N) with processor number *n* is governed by E_{par} and N_o / N_1. When the processor number becomes large, *E*(N) decreases with *n* due to the effect of both the convergence rate and the parallel efficiency. With the MPI version we were able to achieve good implementation for the efficiency of different mesh sizes with different block numbers as observed in Fig.4 and Fig.5. We observed that the implementation with MPI achieved conformity to unity

6. CONCLUSION

We have implemented the solution of the IADE-DY scheme on 2-D Bio-Heat equation on a parallel platform of MPI/PVM cluster via domain decomposition method. We have reported a detailed study on the computational efficiency of a parallel finite difference iterative alternating direction explicit method under a distributed environment with PVM and MPI. Computational results obtained have clearly shown the benefits of using parallel algorithms. We have come to some conclusions that: (1) the parallel efficiency is strongly dependent on the problem size, block numbers and the number of processors as observed in Figures both for PVM and MPI. (2) A high parallel efficiency can be obtained with large scale problems. (3) The decomposition of domain greatly influences the performance of the parallel computation (Fig. 4 – 5). (4) The convergence rate depends upon the block numbers and the number of processors for a given grid. For a given number of blocks, the convergence rate increases with decreasing number of processors and for a given number of processors it decreases with increasing block number for both MPI and PVM (Fig.6 – 7). On the basis of the current parallelization strategy, more sophisticated models can be attacked efficiently.

References

[1.] W. Barry, A. Michael, 2003. Parallel Programming Techniques and Application using Networked Workstation and Parallel Computers. Prentice Hall, New Jersy
[2.] A. Beverly, et al., 2005. The Algorithmic Structure Design Space in Parallel Programming. Wesley Professional
[3.] D. Callahan, K. Kennedy. 1988. Compiling Programs for Distributed Memory Multiprocessors. Journal of Supercomputer 2, pp 151 – 169
[4.] J.C Chato, 1998. Fundamentals of Bio-Heat Transfer. Springer-Verlag, Berlin
[5.] H. Chi-Chung, G. Ka-Kaung, et. al., 1994. Solving Partial Differential Equations on a Network of Workstations. IEEE, pp 194 – 200.
[6.] M. Chinmay, 2005. Bio-Heat Transfer Modeling. Infrared Imagine, pp 15 – 31
[7.] R. Chypher, A. Ho, et al., 1993. Architectural Requirements of Parallel Scientific Applications with Explicit Communications. Computer Architecture, pp 2 – 13
[8.] P.J Coelho, M.G Carvalho, 1993. Application of a Domain Decomposition Technique to the Mathematical Modeling of Utility Boiler. Journal of Numerical Methods in Eng., 36 pp 3401 – 3419
[9.] D'Ambra P., M. Danelutto, Daniela S., L. Marco, 2002. Advance Environments for Parallel and Distributed Applications: a view of current status. Parallel Computing 28, pp 1637 – 1662.
[10.]Z.S Deng, J. Liu, 2002. Analytic Study on Bio-Heat Transfer Problems with Spatial Heating on Skin Surface or Inside Biological Bodies. ASME Journal of Biomechanics Eng., 124 pp 638 – 649
[11.]H.S Dou, Phan-Thien. 1997. A Domain Decomposition Implementation of the Simple Method with PVM. Computational Mechanics 20 pp 347 – 358
[12.]F. Durst, M. Perie, D. Chafer, E. Schreck, 1993. Parallelization of Efficient Numerical Methods for Flows in Complex Geometries. Flow Simulation with High Performance Computing I, pp 79 – 92, Vieweg, Braunschelweig
[13.]Eduardo J.H., Yero M.A, Amaral H. (2007). Speedup and Scalability Analysis of Master-Slave Applications on Large Heterogeneous Clusters. Journal of Parallel & Distributed Computing, Vol. 67 Issue 11, 1155 – 1167.
[14.]Fan C., Jiannong C., Yudong S. 2003. High Abstractions for Message Passing Parallel Programming. Parallel Computing 29, 1589 – 1621.
[15.]I. Foster, J. Geist, W. Groop, E. Lust, 1998. Wide-Area Implementations of the MPI. Parallel Computing 24 pp 1735 – 1749.

[16.] A. Geist A. Beguelin, J. Dongarra, 1994. Parallel Virtual Machine (PVM). Cambridge, MIT Press
[17.] G.A Geist, V.M Sunderami, 1992. Network Based Concurrent Computing on the PVM System. Concurrency Practice and Experience, pp 293 – 311
[18.] W. Groop, E. Lusk, A. Skjellum, 1999. Using MPI, Portable and Parallel Programming with the Message Passing Interface, 2nd Ed., Cambridge MA, MIT Press
[19.] Guang-Wei Y., Long-Jun S., Yu-Lin Z. 2001. Unconditional Stability of Parallel Alternating Difference Schemes for Semilinear parabolic Systems. Applied Mathematics and Computation 117, pp 267 – 283
[20.] M. Gupta, P. Banerjee, 1992a. Demonstration of Automatic Data Partitioning for Parallelizing Compilers on Multi-Computers. IEEE Trans. Parallel Distributed System, 3, vol. 2, pp 179 – 193
[21.] K. Jaris, D.G. Alan, 2003. A High-Performance Communication Service for Parallel Computing on Distributed Systems. Parallel Computing 29, pp 851 – 878
[22.] J. Z. Jennifer, et al., 2002. A Two Level Finite Difference Scheme for 1-D Penne's Bio-Heat Equation.
[23.] J. Liu, L.X Xu, 2001. Estimation of Blood Perfusion Using Phase Shift Temperature Response to Sinusoidal Heating at Skin Surfaces. IEEE Trans. Biomed. Eng., Vol. 46, pp 1037 – 1043
[24.] Mitchell, A.R., Fairweather, G. (1964). Improved forms of the Alternating direction methods of Douglas, Peaceman and Rachford for solving parabolic and elliptic equations, Numer. Maths, 6, 285 – 292.
[25.] J. Noye, 1996. Finite Difference Methods for Partial Differential Equations. Numerical Solutions of Partial Differential Equations. North-Hilland Publishing Company.
[26.] D.W Peaceman, H.H Rachford, 1955. The Numerical Solution of Parabolic and Elliptic Differential Equations. Journal of Soc. Indust. Applied Math. 8 (1) pp 28 – 41
[27.] Peizong L., Z. Kedem, 2002. Automatic Data and Computation Decomposition on Distributed Memory Parallel Computers. ACM Transactions on Programming Languages and Systems, vol. 24, number 1, pp 1 – 50
[28.] H.H Pennes, 1948. Analysis of Tissue and Arterial Blood Temperature in the Resting Human Forearm. Journal of Applied Physiol. I, pp 93 – 122
[29.] M.J Quinn, 2001. Parallel Programming in C. MC-Graw Hill Higher education New York.
[30.] R. Rajamony, A. L. Cox, 1997. Performance Debugging Shared Memory Parallel Programs Using Run-Time Dependence Analysis. Performance Review 25 (1), pp 75 – 87
[31.] J. Rantakokko, 2000. Partitioning Strategies for Structuring Multi Blocks Grids. Parallel Computing 26 pp 166 – 1680
[32.] B. V Rathish Kumar, et al., 2001. A Parallel MIMD Cell Partitioned ADI Solver for Parabolic Partial Differential Equations on VPP 700. Parallel Computing 42, pp 324 – 340
[33.] B. Rubinsky, et al., 1976. Analysis of a Steufen-Like Problem in a Biological Tissue Around a Cryosurgical Problem. ASME J. Biomech. Eng., vol. 98, pp 514 – 519
[34.] Sahimi, M.S., Sundararajan, E., Subramaniam, M. and Hamid, N.A.A. (2001). The D'Yakonov fully explicit variant of the iterative decomposition method, *International Journal of Computers and Mathematics with Applications*, 42, 1485 – 1496
[35.] V. T Sahni, 1996. Performance Metrics: Keeping the Focus in Routine. IEEE Parallel and Distributed Technology, Spring pp 43 – 56.
[36.] G.D Smith, 1985. Numerical Solution of Partial Differential Equations: Finite Difference Methods 3rd Ed., Oxford University Press New York.
[37.] M. Snir, S. Otto et. al., 1998. MPI the Complete Reference, 2nd Ed., Cambridge, MA: MIT Press.
[38.] X.H Sun, J. Gustafson, 1991. Toward a Better Parallel Performance Metric. Parallel Computing 17.
[39.] M. Tian, D. Yang, 2007. Parallel Finite-Difference Schemes for Heat Equation based upon Overlapping Domain Decomposition. Applied Maths and Computation, 186, pp 1276 – 1292

AUTHOR

Ewedafe Simon Uzezi received the B.Sc. and M.Sc. degrees in Industrial-Mathematics and Mathematics from Delta Stae University and The University of Lagos in 1998 and 2003 respectively. He further obtained his Ph.D. in Numerical Parallel Computing 2010 from the Universityof Malaya, Malaysia. In 2011 he joined UTAR in Malaysia as an Assistant Professor and later lectured in Oman and Nigeria as Senior Lecturer in Computing. Currently he is a Senior Lecturer in the Department of Computing UWI Jamaica.