On the Parallel Design and Analysis for 3-D ADI Telegraph Problem with MPI

Simon Uzezi Ewedafe
Department of Computing
The University of the West Indies,
Mona Kingston 7, Jamaica

Rio Hirowati Shariffudin
Institute of Mathematical Sciences
Universiti Malaya
Kuala Lumpur, Nigeria

***Abstract*—In this paper we describe the 3-D Telegraph Equation (3-DTEL) with the use of Alternating Direction Implicit (ADI) method on Geranium Cadcam Cluster (GCC) with Message Passing Interface (MPI) parallel software. The algorithm is presented by the use of Single Program Multiple Data (SPMD) technique. The implementation is discussed by means of Parallel Design and Analysis with the use of Domain Decomposition (DD) strategy. The 3-DTEL with ADI scheme is implemented on the GCC cluster, with an objective to evaluate the overhead it introduces, with ability to exploit the inherent parallelism of the computation. Results of the parallel experiments are presented. The Speedup and Efficiency from the experiments on different block sizes agree with the theoretical analysis.**

Keywords—3-DTEL; ADI; MPI; SPMD; DD; Parallel Design

I. INTRODUCTION

Parallel computing has greatly motivated the research works on the parallel design and analysis of the 3-DTEL in parallel cluster system. Cluster applications have more processor cores to manage and exploit the computational capacity of high-end machines providing effective and efficient means of parallelism even as the challenges of providing effective resources management grows. It is a known fact that high capacity computing platform are expensive, and are characterized by long-running, high processor-count jobs. The performance of message-passing programs depends on the parallel target machine, and the parallel programming model to be applied to achieve parallelism. In a cluster machines having large number of processing units' scalability becomes an important issue. Many programs from scientific computing have a large potential for parallelism that is exploited best in such a programming model for mixed fast and data parallelism where the parallelism can be structured in the form of concurrent multi-processor tasks [21].

Developing parallel applications have its own challenges in the field of parallel computing. With reference to [11], there are theoretical challenges such as task decomposition, dependence analysis, and task scheduling. Then they are practical challenges such as portability, synchronization, and debugging. However, there exist an alternative and cost effective way of achieving performance through the use of loosely connected system of processors with a local area network [3]. Hence, for a global task with other processors relevant data needs to be passed from processor to processor through a message-passing mechanism [20, 15], since there is greater demand for computational speed and the computations must be completed within reasonable time period. A multi-processor task can be implemented on a subset of processors, and one of the advantages is based on the fact that for many message-passing machines communication costs are affected by the number of participating processors.

Design and analysis for finite difference DD for 2-D heat equation has been discussed in [23], and the parallelization for 3-DTEL on parallel virtual machine with DD [8] show effective load scheduling over various mesh sizes, which produce the expected inherent speedups. Parallel algorithms have been implemented for the finite difference method by [12], and [21, 13] use the discrete eigen functions method with the AGE method on telegraph equation problem.

The theoretical properties of the 3-D ADI algorithm with the parallel design approach employing SPMD model with DD are promising, achieving good performance as to what was done by [7] in practice can be challenging. There is a tradeoff between the reduction of the time required for an inherently sequential part of an algorithm, and an increase in the number of the iterations required to converge [2]. Previous work on 3-D ADI scheme did not consider the parallel design approach on parallelism and improvement on scalability. To write SPMD programs using one of the standard message-passing software like MPI [13] requires the explicit administration of processors with a large user group. In this paper, we present a support for the implementation of parallel design and analysis with the use of DD strategy. Our programming style allows the application programmer to specify the program organization in a clear and readable program code.

We presented a detailed study of using parallel design and analysis on 3-DTEL, and solved by the use of ADI method on a GCC cluster MPI. The SPMD model is employed with DD to enhance overlapping communication with computation that resulted in significant improved speedup, effectiveness, and efficiency across varying mesh sizes as compared to [7].

Our results demonstrated the overlap communication with computation, and the ability to arbitrary use of varying mesh sizes distribution on GCC to reduce memory pressure while preserving parallel efficiency. On the other hand, the advantage of our platform is to have somewhat specification mechanism through a static distribution, and an execution implementation.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 introduces the model for the 3-DTEL and the 3-D ADI scheme. Section 4 introduces the parallel design and analysis. Section 5 introduces the results of several experiments, which illustrate and evaluate the parallelization possible with our platform. Section 6 gives the conclusion.

II. Related Work

A work by [16] achieved configuration of MPI-based message passing programs, and various other platforms for the application of telegraph and heat equations have been done in [7, 8]. Description of application aware job scheduler that dynamically controls resource allocation among concurrently executing jobs was done by [22]. A framework called 'Gridway' for adaptive execution of applications in Grids was described by [14]. Parallelization by time decomposition was first proposed by [18] with motivation to achieve parallel real-time solutions, and even the importance of loop parallelism, loop scheduling have been extensively studied [1]. The ADI method for the Partial Differential Equations (PDE) proposed by [19] has been widely used for solving algebraic systems resulting from finite difference method analysis of PDE in several scientific and engineering applications. Works on parallel implementation of 2-D Telegraph problem on cluster systems have been done in [10, 12].

In [12] the unconditional stability of the alternating difference schemes has similarity to our scheme and shows that the unconditional stability application is useful to its speedup and efficiency as studied. Our implementation in the GCC platform has several aspects that differentiate it from the above. GCC is designed for application running on distributed memory clusters, which can dynamically and statically calculate partition sizes based on the run-time performance of the application. We use an efficient algorithm with stability which maps data using message passing over the GCC cluster. We evaluated our system using experimental results from speedup and efficiency for the system utilization. Our approach is best suited to applications where data and computations are uniformly distributed across processors.

III. The Model Problem

We consider the second order telegraph equation in 3-D:

$$\frac{\partial^2 v}{\partial t^2} + a\frac{\partial v}{\partial t} - \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) = 0 \qquad (3.1)$$

where $a = RC + GL$, let $\Delta x, \Delta y \; and \; \Delta z$ be the grid spacing in the x, y, z and t directions, where $\Delta x = \Delta y = \Delta z = 1/m$, m is a positive integer. We can solve (3.1) by extending the 1-D simple implicit finite difference method [21] of the telegraph equation to the above 3-D telegraph equation, (3.1) becomes:

$$\frac{v_{i,j,k}^{n+1} - 2v_{i,j,k}^{n} + v_{i,j,k}^{n-1}}{(\Delta t)^2} + a\frac{v_{i,j,k}^{n+1} - v_{i,j,k}^{n-1}}{2\Delta t} - \left\{ \begin{array}{l} \dfrac{v_{i+1,j,k}^{n+1} - 2v_{i,j,k}^{n+1} + v_{i-1,j,k}^{n+1}}{(\Delta x)^2} \\ + \dfrac{v_{i,j+1,k}^{n+1} - 2v_{i,j,k}^{n+1} + v_{i,j-1,k}^{n+1}}{(\Delta y)^2} \\ + \dfrac{v_{i,j,k+1}^{n+1} - 2v_{i,j,k}^{n+1} + v_{i,j,k-1}^{n+1}}{(\Delta z)^2} \end{array} \right\} = 0 \qquad (3.2)$$

although this simple implicit scheme is unconditionally stable, therefore, the computational time is extremely huge.

A. ADI Method on 3-DTEL

We derive the ADI method for 3-DTEL of the simple implicit finite difference method by using a general ADI procedure [6] extended to (3.1). The ADI method is a well-known method for solving the PDE. The main feature of ADI is to sweep directions alternatively. In contrast to the standard finite-difference formulation with only one iteration to advance from the *nth* to (*n + 1*)*th* time step, the formulation of the ADI method requires multilevel intermediate steps to advance from the *nth* to (*n + 1*)*th* time step. Equation (3.2) can be rewritten as:

$$\left(I + \sum_{m=1}^{3} A_m\right) v_{i,j,k}^{n+1} - 2C_o v_{i,j,k}^{n} + C_1 v_{i,j,k}^{n-1} = 0 \qquad (3.3)$$

where the operators of I, $A_m s$, and the constants of C_o, C_1 are define as:

$$Iv_{i,j,k}^{n} \equiv v_{i,j,k}^{n} \qquad (3.4)$$

$$A_1 v_{i,j,k}^{n} \equiv -\rho_x \left(v_{i+1,j,k}^{n} - 2v_{i,j,k}^{n} + v_{i-1,j,k}^{n}\right) \qquad (3.5)$$

$$A_2 v_{i,j,k}^{n} \equiv -\rho_y \left(v_{i,j+1,k}^{n} - 2v_{i,j,k}^{n} + v_{i,j-1,k}^{n}\right) \qquad (3.6)$$

$$A_3 v_{i,j,k}^{n} \equiv -\rho_z \left(v_{i,j,k+1}^{n} - 2v_{i,j,k}^{n} + v_{i,j,k-1}^{n}\right) \qquad (3.7)$$

$$C_o \equiv \frac{1}{(\Delta t)^2} \Big/ \left(\frac{1}{(\Delta t)^2} + \frac{a}{2\Delta t}\right) \qquad (3.8)$$

$$C_1 \equiv \left(\frac{1}{(\Delta t)^2} - \frac{a}{2\Delta t}\right) \Big/ \left(\frac{1}{(\Delta t)^2} + \frac{a}{2\Delta t}\right) \qquad (3.9)$$

the constant of ρ_x, ρ_y and ρ_z are:

$$\rho_x = \frac{b}{(\Delta x)^2} \Big/ \left(\frac{1}{(\Delta t)^2} + \frac{a}{2\Delta t}\right) \qquad (3.10)$$

$$\rho_y = \frac{b}{(\Delta y)^2} \Big/ \left(\frac{1}{(\Delta t)^2} + \frac{a}{2\Delta t} \right) \tag{3.11}$$

$$\rho_y = \frac{b}{(\Delta y)^2} \Big/ \left(\frac{1}{(\Delta t)^2} + \frac{a}{2\Delta t} \right) \tag{3.12}$$

TABLE I. THE ADI 3-DTELALGORITHM

The ADI 3-DTEL Algorithm

Input = $v_{i,j,k}^{n}, v_{i,j,k}^{n-1} \ \forall i, j, k$

Output = $v_{i,j,k}^{n+1} \ \forall i, j, k$

Begin

Sub-Iteration 1:

$$-\rho_x v_{i+1,j,k}^{n+1(1)} + (1+2\rho_x) v_{i,j,k}^{n+1(1)} - \rho_x v_{i-1,j}^{n+1(1)} =$$
$$-(A_2 + A_3) v_{i,j,k}^{n+1(*)} + (2C_o v_{i,j,k}^{n} - C_1 v_{i,j,k}^{n-1})$$
$$\forall i, j, k$$

Sub-Iteration 2:

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

Sub-Iteration 3:

$$-\rho_z v_{i,j,k+1}^{n+1(3)} + (1+2\rho_z) v_{i,j,k}^{n+1(3)} - \rho_z v_{i,j,k-1}^{n+1(3)}$$
$$= v_{i,j,k}^{n+1(2)} + A_3 v_{i,j,k}^{n+1(*)}. \quad \forall i, j, k$$

End

and set

$$v_{i,j,k}^{n+1(*)} = 2v_{i,j,k}^{n} - v_{i,j,k}^{n-1} \tag{3.13}$$

which is a prediction of $v_{i,j,k}^{n+1}$ by the extrapolation method.

Then splitting (3.3) by using an ADI procedure as in [17], we get a set of recursion relations as follows:

$$(I + A_1) v_{i,j,k}^{n+1(1)} = -(A_2 + A_3) v_{i,j,k}^{n+1(*)} + (2C_0 v_{i,j,k}^{n} - C_1 v_{i,j,k}^{n-1}) \tag{3.14}$$

$$(I + A_2) v_{i,j,k}^{n+1(2)} = v_{i,j,k}^{n+1(1)} + A_2 v_{i,j,k}^{n+1(*)} \tag{3.15}$$

$$(I + A_3) v_{i,j,k}^{n+1(3)} = v_{i,j,k}^{n+1(2)} + A_3 v_{i,j,k}^{n+1(*)} \tag{3.16}$$

where $v_{i,j,k}^{n+1(1)}, v_{i,j,k}^{n+1(2)}$ are the intermediate solutions and the desired solution is $v_{i,j,k}^{n+1} = v_{i,j,k}^{n+1(3)}$. Finally, expanding A_1, A_2 and A_3 on the left side of (3.14) and (3.16), we get the 3-D ADI algorithm as in Table 1.

IV. PARALLEL IMPLEMENTATION, DESIGN AND ANALYSIS

A. The Parallel Platform

The Geranium Cadcam Cluster consist of 32 Intel Pentium dual core processor at 1.73GHZ and 0.99GB RAM. Communication is through a fast Ethernet of 100 MBits per seconds running Linux, located at the University of Malaya. The cluster performance has high memory bandwidth with a message passing supported by MPI [13]. The program is written in C and provides access to MPI through calling MPI library routines. The platform contains more computations on varying set of mesh sizes. Performance in the platform concerns the resource assessment and code placement on computing resources [5]. The 3-DTEL with ADI scheme is implemented on the GCC cluster, with an objective to evaluate the overhead it introduces with ability to exploit the inherent parallelism of the computation. We observed the scalability across the varying number of processors and mesh sizes, to enable the speedup we need convergence in fewer than N iterations.

B. Domain Decomposition

The parallelization of the computations is implemented by means of grid partitioning technique. The computing domain is decomposed into many blocks with reasonable geometries. Along the block interfaces, auxiliary control volumes containing the corresponding boundary values of the neighboring block are introduced, so that the grids of neighboring blocks are overlapped at the boundary. When the domain is split, each block is given an I-D number by a "master" task, which assigns these sub-domains to "slave" tasks running in individual processors. In order to couple the sub-domains' calculations, the boundary data of neighboring blocks have to be interchanged after each iteration. The calculations in the sub-domains use the old values at the sub-domains' boundaries as boundary conditions. This may affect the convergence rate; however, because the algorithm is implicit, the blocks strategy can preserve nearly same accuracy as the sequential program.

The DD is used to distribute data between different processors; the static load balancing is used to maintain same computational points for each processor. The partitioning and load balancing is done in the pre-processing stage giving no room for extra storage when the parallel program is executed. Data parallelism originated the SPMD [17], thus, the finite difference approximation used in this paper can be treated as an SPMD problem. Same computation is performed for multiple data sets, and the multiple data are different parts of the overall grid.

C. Parallel ADI with MPI

We focus on computational domain partitions in implementing the parallel 3-DTEL ADI scheme on GCC platform. We need divide the dimensions into sub-domains with no unique way of partitioning the domain of computation. The case of making a balance between the implementation of the algorithm and the communication efficiency is paramount to balance. The partitioning considered is the orientation of slices changing with the sweeps according to [4].

After *x*-sweeps, the orientation changes to the *y* or the *z* direction. In this process each processor owns three data domains, one for each direction. Implementing the parallel algorithm for solving (3.1) is based on: indication of sweeping direction for each sub-domain. Sweeping direction of each sub-domain must be in opposite direction of its neighbors. For example, we must use left right direction for odd sub-domains and right left direction for even sub-domains. Updating start node of each sub-domain with (3.14) and (3.16), each processor of the parallel machine works only on its specific portion of the grid and when processor needs information from the nearest neighbor a message is passed through the MPI message passing library. For the best parallel performance, one would like to have optimal load balancing and as little communication between processors as possible. Considering load balancing first, one would like each processor to do exactly the same amount of work, hence, each processor is not idle. For the finite difference code, the basic computational element usually is the node; it makes sense to partition the grid such that each processor gets an equal number of nodes to work on. The second criterion is that the amount of communication between processors be made as small as possible. To minimize communication, the program must divide the domain in a way that minimizes the length of the touching faces in the different sub-domains. The number of processors that one processor has to communicate with also contributes to additional communication time, because of the latency penalty for starting the new message. At first step, we divide the spatial computational domain to $P = P_1 \times P_2 \times P_3$. We can use the non-blocking message passing for this communication stage to reduce computing time by allowing work to be done while communication is in progress.

D. Load Balancing

With static load balancing, the computation time of parallel subtasks should be relatively uniform across processors; otherwise, some processors will be idle waiting for others to finish their subtasks. Therefore, the domain decomposition should be reasonably uniform. A better load balancing is achieved with the pool of tasks strategy, which is often used in master – slave programming [2]: the master task keeps track of idle slaves in the distributed pool and sends out the next task to the first available idle slave. With this strategy, the processors are kept busy until there is no further task in the pool. If the tasks vary in complexity, the most complex tasks are sent out to the most powerful processor first. With this strategy, the number of sub-domains should be relatively large compared to the number of processors.

Otherwise, the slave solving the last sent block will force others to wait for the completion of this task; this is especially true if this processor happens to be the least powerful in the distributed system. The block size should not be too small either, since the overlap of nodes at the interfaces of the sub-domains become significant. This results in a doubling of the computations of some variables on the interfacial nodes, leading to a reduced efficiency. Increasing the block number also lengthens the execution time of the master program, which leads to a reduced efficiency.

E. Speedup and Efficiency

A simple speedup analysis with reference to [2] produces the following:

$$\varphi = \frac{N}{Nr(K+1)+K}, \tag{4.1}$$

where r is the ration of the time taken by coarse propagation to fine propagation over the same time interval, K is the number of iterations required for convergence, and communication overhead is ignored. In the limit $r \to 0,\ \varphi \to \frac{N}{K}$, therefore, the efficiency will be $\frac{1}{K}$. The algorithm for the scheme is performed on a distributed memory system of p processors, assumes that each processors initially stores $n = N/p$ objects distributed over the entire physical domain. In the first iteration of the algorithm, the domain is decomposed into two sub-domains so that the difference between the sums of the weight of the sub-domain is as small as possible. Then the same process is applied to two sub-domains in parallel, and process is repeated recursively, for log p iteration. In other words, during iteration i, $1 \le i \le \log p$, the p processors are group into 2^{i-1} groups of $p/2^{i-1}$ processors each. At the beginning of the iteration, the problem domain is already partitioned into 2^{i-1} sub-domains and the objects in each sub-domain are stored in single group of processors. At the end of the iteration, each processor group is divided into two groups, and the corresponding sub-domain is divided into two sub-groups with the object in one sub-domain residing in one half the processors and the other objects in the other sub-domain residing in the other half of processor

V. Results and Discussion

Consider the Telegraph Equation of the form:

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial v}{\partial z^2} = \frac{\partial^2 v}{\partial t^2} + \frac{\partial v}{\partial t} + v \tag{5.1}$$

the boundary condition and initial condition posed are:

$$\left.\begin{aligned} v(0,y,z) &= 0 \\ v(1,y,z) &= 100 \\ v(x,0,z) &= 0 \\ v(x,1,z) &= 100 \\ v(x,y,0) &= 0 \\ v(x,y,1) &= 100 \end{aligned}\right\} \quad t \ge 0 \tag{5.2}$$

$$v(x,y,z) = e^{xyz}, \tag{5.3}$$

A. Parallel Efficiency

The speedup and efficiency obtained for various sizes, for 70x70x6 to 210x210x6, are for various numbers of sub-domains, for B = 50 are listed in Tables 2 – 4. In the Tables

we also 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 slave computational time, T_{SC}, and the average slave data communication time, T_{SD}, all in seconds. The speedup and efficiency versus the number of processors are shown in Fig. 1 and Fig. 2, respectively, with block number B as a parameter.

The results in the Tables show that the parallel efficiency increases with increasing grid size for given block number, and decreases with the increasing block number for given grid size. As the number of processors increase, though this leads to a decrease in execution time, but a point is reached when the increased processors will not have much impact on total execution time. Hence, when the numbers of processors increase, balancing the number of computational cells per processors will become a difficult task due to significant load imbalance. The gain in increasing execution time for certain mess sizes is due to uneven distribution of the computational cell, and the execution time has a very small change due to DD influence on performance in parallel computation.

The total CPU time is composed of three parts: the CPU time for the master task, the average slave CPU time for data communication and the average slave CPU time for computation, $T = T_M + T_{SD} + T_{SC}$.

TABLE II. THE WALL TIME T_W, THE MASTER TIME T_M, THE SLAVE DATA TIME T_{SD}, THE SLAVE COMPUTATIONAL TIME T_{SC}, THE TOTAL TIME T, THE PARALLEL SPEED-UP S_{PAR} AND THE EFFICIENCY E_{PAR} FOR A MESH OF 70X70X6, WITH B = 50 BLOCKS AND $NITER$ = 100.

N	T_w	T_m	T_{sd}	T_{sc}	T	S_{par}	E_{par}
1	1245	38	4	522	564	1.000	1.000
2	621	36	3	281	320	1.761	0.881
3	318	36	3	188	227	2.482	0.827
4	257	36	3	158	197	2.865	0.716
5	238	36	3	131	170	3.324	0.665
6	219	36	3	107	146	3.864	0.644
7	206	36	3	92	131	4.321	0.617
8	205	36	3	76	115	4.918	0.615
12	183	36	3	60	96	5.921	0.493
16	176	36	3	44	83	6.824	0.427
20	155	36	3	28	67	8.211	0.411
24	138	36	3	25	64	8.926	0.372
28	125	36	3	21	60	9.412	0.336
32	112	36	3	14	53	10.896	0.341

TABLE III. THE WALL TIME T_W, THE MASTER TIME T_M, THE SLAVE DATA TIME T_{SD}, THE SLAVE COMPUTATIONAL TIME T_{SC}, THE TOTAL TIME T, THE PARALLEL SPEED-UP S_{PAR} AND THE EFFICIENCY E_{PAR} FOR A MESH OF 120X120X6, WITH B = 50 BLOCKS AND $NITER$ = 100.

N	T_w	T_m	T_{sd}	T_{sc}	T	S_{par}	E_{par}
1	2721	119	13	1589	1721	1.000	1.000
2	1292	113	13	822	948	1.818	0.909
3	694	113	13	497	623	2.764	0.921
4	482	113	13	347	473	3.641	0.910
5	449	113	13	325	451	3.817	0.763
6	408	113	13	243	369	4.663	0.777
7	396	113	13	225	351	4.912	0.702
8	385	113	13	167	293	5.873	0.734
12	371	113	13	135	261	6.618	0.552
16	372	113	13	97	223	7.738	0.484
20	348	113	13	59	185	9.328	0.466
24	322	113	13	37	163	10.611	0.442
28	308	113	13	28	154	11.322	0.404
32	284	113	13	12	138	12.589	0.393

TABLE IV. THE WALL TIME T_W, THE MASTER TIME T_M, THE SLAVE DATA TIME T_{SD}, THE SLAVE COMPUTATIONAL TIME T_{SC}, THE TOTAL TIME T, THE PARALLEL SPEED-UP S_{PAR} AND THE EFFICIENCY E_{PAR} FOR A MESH OF 210X210X6, WITH B = 50 BLOCKS AND $NITER$ = 100.

N	T_w	T_m	T_{sd}	T_{sc}	T	S_{par}	E_{par}
1	13825	378	55	8086	8519	1.000	1.000
2	6439	374	54	4189	4617.3	1.845	0.923
3	3427	374	54	2662	3090	2.757	0.919
4	2718	374	54	1909	2337	3.646	0.914
5	2589	373	54	1548	1975	4.315	0.863
6	2443	373	54	1286	1713	4.974	0.829
7	2094	373	54	1124	1551	5.495	0.785
8	2019	373	54	970	1398	6.184	0.773
12	1924	373	54	562	989	8.616	0.718
16	1918	373	54	396	823	10.352	0.647
20	1710	373	54	278	705	12.1	0.605
24	1621	373	54	230	656.6	12.984	0.541
28	1597	373	54	163	591	14.448	0.516
32	1481	373	54	132	558	15.264	0.477

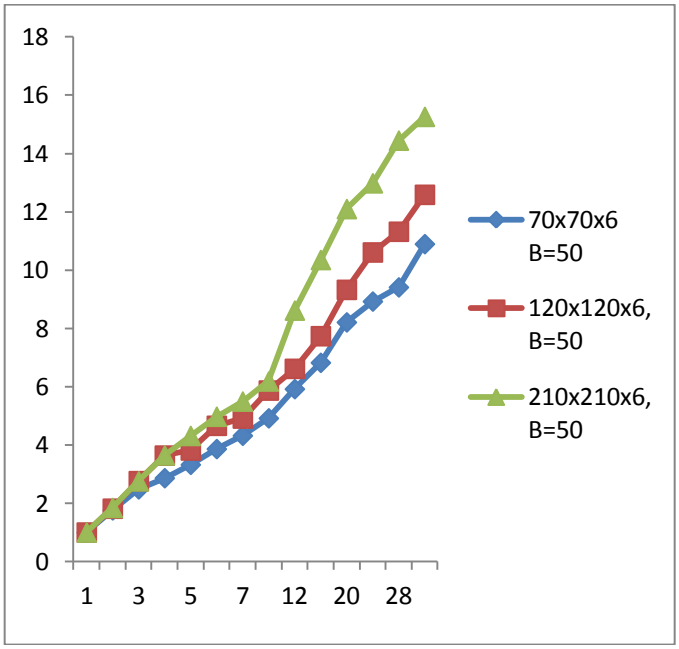

Fig. 1. Speedup versus the number of processors for mesh 70x70x6,120x120x6 and 210x210x6

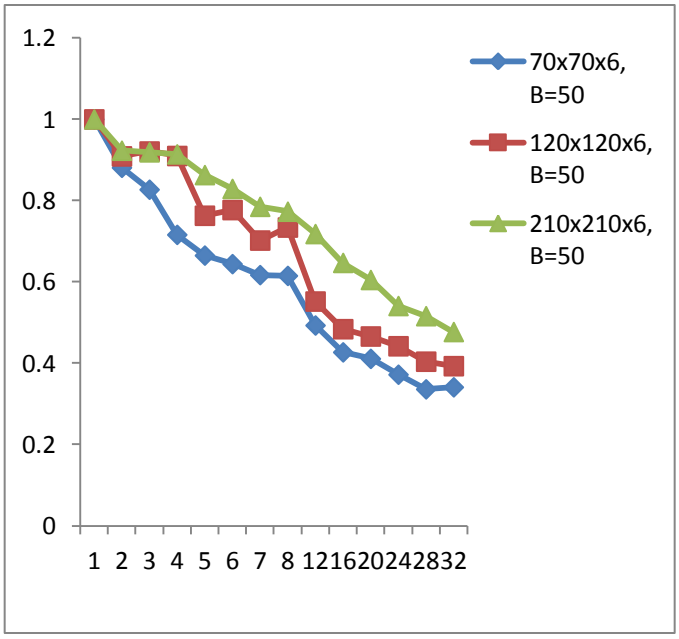

Fig. 2. Parallel efficiency versus the number of processors for mesh 70x70x6, 120x120x6 and 210x210x6

B. Numerical Efficiency

The numerical efficiency includes the DD efficiency and convergence rate behavior. The DD efficiency includes the increase of floating point operations induced by grid overlap at interfaces and the CPU time variation generated by DD techniques. In Table 5, we listed the total CPU time distribution over various grid sizes and block numbers running with only one processor. In Table, the DD efficiency can be calculated, and the result as shown in Fig. 3. Note that the DD efficiency can be greater than one, even with one processor. Fig. 3 also shows that the optimum number of sub-domains, which maximizes the DD efficiency E_{DD}, increases with the grid size. The convergence rate behavior, 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, describes 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 6 and Fig. 4, and Table 7 and Fig. 5.

It can be seen that the convergence rate decreases with increasing block number and increasing number of processors for given grid size. The larger the grid size, the higher the convergence rate.

TABLE V. THE TOTAL COMPUTATIONAL TIME *T* FOR 100 ITERATIONS AS A FUNCTION OF VARIOUS BLOCK NUMBERS

	B = 1	*B = 8*	*B = 16*	*B = 24*	*B = 50*
70x70x6	*411*	*437*	*481*	*509*	*564*
120x120x6	*572*	*641*	*987*	*1394*	*1721*
210x210x6	*3493*	*4168*	*4928*	*6294*	*8519*

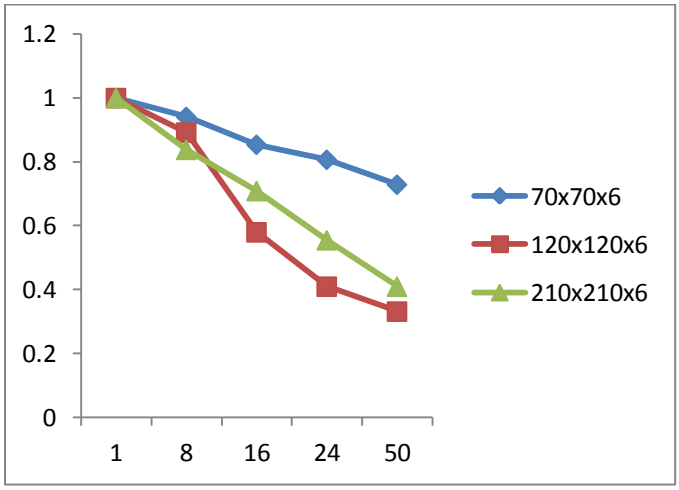

Fig. 3. The DD efficiency versus the number of sub-domains for various meshes.

TABLE VI. THE NUMBER OF ITERATION TO ACHIEVE A GIVEN TOLERANCE OF 10^{-3} FOR A GRID OF 70X70X6

N	*B = 1*	*B = 16*	*B = 50*
1	*1796*	*1987*	*2129*
2	*1796*	*2206*	*2346*
4	*1796*	*2293*	*2492*
8	*1796*	*2371*	*2524*

12	*1796*	*2396*	*2598*
16	*1796*	*2417*	*2609*
28	*1796*	*3214*	*4968*
32	*1796*	*3486*	*5291*

TABLE VII. THE NUMBER OF ITERATION TO ACHIEVE A GIVEN TOLERANCE OF 10^{-2} FOR A GRID OF 120X120X6

N	*B = 1*	*B = 16*	*B = 50*
1	*2138*	*2313*	*2434*
2	*2138*	*2329*	*2518*
4	*2138*	*2348*	*2531*
8	*2138*	*2461*	*2687*
12	*2138*	*2461*	*2692*
16	*2138*	*2518*	*2698*
28	*2138*	*3763*	*5321*
32	*2138*	*3775*	*5711*

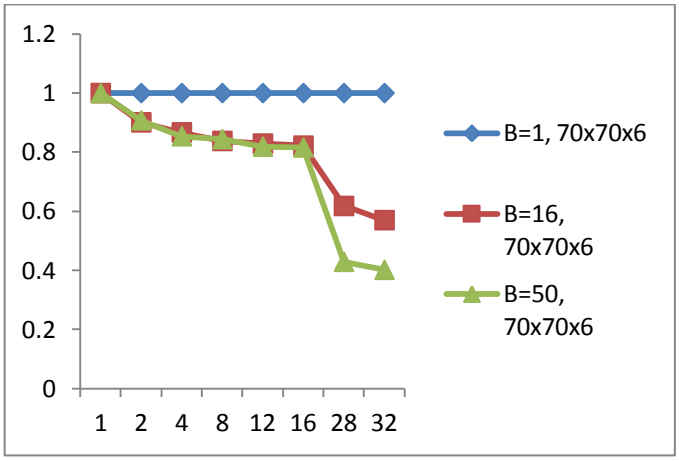

Fig. 4. Convergence behavior with domain decomposition for mesh 70x70x6

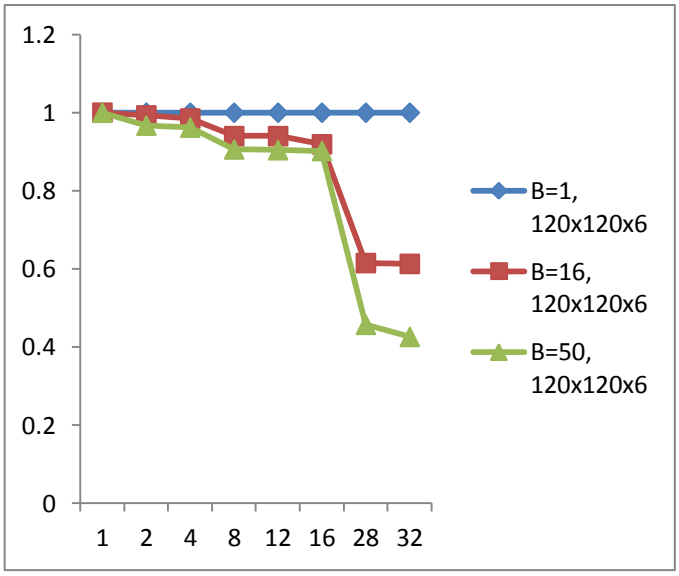

Fig. 5. Convergence behavior with domain decomposition for mesh 120x120x6

VI. CONCLUSION

The results presented in this paper show the study on the parallel design and analysis for 3-D TEL ADI scheme with MPI. The objective is to present a design for the GCC for distributed computation, because they depend on empirical concern. The system allows a parallel collection of overlapping communication to avoid unnecessary synchronization and to have the impact of parallel convergence. In addition to the use of ease of our platform, compared to other approaches show negligible overhead with effective load scheduling over various mesh sizes, which produce the expected inherent speedups. It was also confirmed that flexible scheduling for the overlapping communication are important, and this is easy on with SPMD model as seen from the Tables and Figures. Computational results obtained have clearly shown the benefits of parallelization. The DD greatly influences the performance of the 3-DTEL ADI scheme on the parallel computers. On the basis of the current parallelization strategy, more sophisticated models can be attacked efficiently. Similarly, we are interested in improving our algorithms and testing implementations on additional architectures.

VII. FUTURE WORK

The description of 3-DTEL with the use of ADI method on GCC Cluster System with MPI employing the SPMD technique has been carried out. This paper allows a parallel collection of overlapping communication to avoid unnecessary synchronization and to have the impact of parallel convergence. We suggest future work to be carried out on the 3-DTEL employing the used of Iterative Alternating Direction Implicit (IADE) method. Parallel implementation for the scheme could use the Input File Affinity Measure on a tightly coupled distributed environment with dynamic allocation of task with varying mesh sizes.

REFERENCES

[1] J. Aguilar, E. Leiss, 'Parallel Loop Scheduling Approaches for Distributed and Shared Memory System', Parallel Process Letter 15 (1 – 2), 2005, pp. 131 – 152

[2] E. Aubanel, 'Scheduling of tasks in the parareal algorithm' Parallel Computing 37 (3), 2011, 172 – 182

[3] W. Barry, A. Michael, '*Parallel Programming Techniques and Application using Networked Workstation and Parallel Computers*' 2003, Prentice Hall, New Jersy

[4] G. Baolai. On the Performance of Parallel Implementation of an ADI Scheme for Parabolic PDEs on Shared and Distributed Memory. Shared Hierarchical Research Computing Network, The University of Western Ontario.

[5] D. Cyril, M. Fabrice, 'Jacobi computation using mobile agent' Int'l Journal of Computer Science & Information Technologies, 1 (5), 2010, 392 – 401

[6] D.J Evans, B. Hassan, 'Numerical Solution of the Telegraph Equation by the AGE Method', Int'l Journal of Computer Mathematics Vol. 80, number 10, 2003, pp 1289 – 1297

[7] S. U. Ewedafe, H. S. Rio, 'Parallelization of 2-D IADE-DY Scheme on Geranium Cadcam Cluster for Heat Equation' Int'l Jour. Of Advanced Research in Artificial Intelligence, 2 (6), 2013, pp. 27 – 33

[8] S. U. Ewedafe, H. S. Rio, 'Parallelization of 3-D ADI Scheme on Telegraph Problem using Domain Decomposition with PVM' Int'l Jour. Of Applied Information Systems, 4 (11), 2012, pp. 12 – 24

[9] S. U. Ewedafe, H. S. Rio, 'Parallel Implementation of 2-D Telegraph Equation on MPI/PVM Cluster' Int'l Jour. of Parallel Programming, 39, Issue 2, 2011, 202 – 231

[10] S. U. Ewedafe, H. S. Rio, 'Armadillo Generation Distributed Systems & Geranium Cadcam Cluster for solving 2-D Telegraph Equation' Int'l Jour. of Computer Mathematics, 88, Issue 3, 2011, 589 – 609

[11] N. Giacaman, O. Sinnen, 'Parallel iterator for parallelizing object-oriented applications' Intl journal of parallel programming, 39 (2), 2011, 223 – 269, 2011.

[12] Y. Guang-Wei, Long-Jun S., Yu-Lin Z., 'Unconditional Stability of Parallel Alternating Difference Schemes for Semilinear parabolic Systems' Applied Mathematics and Computation 117, 2001, pp 267 – 283

[13] W. Groop, E. Lusk, A. Skjellum, 'Using MPI, portable and parallel programming with the message passing interface,' 1999, 2nd Ed., Cambridge MA, MIT Press

[14] E. Huedo, R. Montero, I. Llorente, 'A Framework for Adaptive Execution in Grids' Software Practice & Experiences 34 (7), 2004, pp. 631 - 651

[15] K. Jaris, D.G. Alan, 'A High-Performance Communication Service for Parallel Computing on Distributed System', Parallel Computing 29, 2003, pp 851 – 878

[16] L, Kale, S. Kumar, J. DeSouza, 'A malleable-Job System for Time-Shared Parallel Machines' Proceedings of the second IEEE/ACM Int'l Symposium on Cluster Computing, IEEE Computer Society, 2002, Washington DC, U.S.A,

[17] H. Laurant, 'A method for automatic placement of communications in SPMD parallelization' Parallel computing 27, 2001, 1655 – 1664

[18] J. L. Lions., Y. Maday, G. Turinki, 'Parareal in time discretization of PDE' Comptes, rendus de lacadimie des sciences – series 1 – mathematics 332 (7), 2011, 661 – 668

[19] D.W Peaceman, H.H Rachford, 'The Numerical Solution of Parabolic and Elliptic Differential Equations' Journal of Soc. Indust. Applied Math. 8 (1), 1955, pp 28 – 41

[20] Peizong L., Z. Kedem, 'Automatic Data and Computation Decomposition on Distributed Memory Parallel Computers' ACM Transactions on Programming Languages and Systems, vol. 24, number 1, 2002, pp 1 – 50

[21] T. Rauber, G. Runger, 'A Transformation Approach to Derive Efficient Parallel Implementations', IEEE Transactions on Software Engineering, 26 (4), 2000, pp. 315 - 399

[22] J. Weissman, L. Rao, D. England, 'Integrated Scheduling: The Best of both Worlds', Jour. Parallel & Distri. Computing 63 (6), 2003, pp. 631 - 651

[23] W. Zheng-Su, Z. Baolin, C. Guang-Nan, 'Design and analysis for finite difference DD for 2-D heat equation', ICA3PP – 02, IEEE Computer Society