International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 7, Issue 1, January - February 2018 ISSN 2278-6856

Discretization of 2-D Active Thermal Control Problem on Parallel Virtual Machine (PVM) with Stationary Iterative Methods

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

[1]The University of the West Indies, Department of Computing, Mona Campus, Kingston, Jamaica

[2]University of Malaya, Institute of Mathematical Sciences, Kuala Lumpur, Malaysia

Abstract

We present the discretization of 2-D Active Thermal Control Process (2D-ATCP) running on a Master-Slave platform. The parallel implementation was carried out using Parallel Virtual Machine (PVM). The 2-D ATCP was discretized using stationary iterative methods. Parallel compare of these methods ith parallel algorithms were experimentally evaluated and numerical results show their effectiveness. We show that the schemes have scalability performance with less information to schedule tasks. The results show that various block sizes on various domain of decomposition can significantly improve system throughput and the platform can be used for more sophisticated cluster scheduling on different dimensions of ATCP.

Keywords:2D-ATCP, Stationary Methods, PVM, Parallel Implementation.

1. INTRODUCTION

The current developments in parallelism and distributed parallel simulations have shown that parallel computer systems are viable to overcome scientific and computationally intense challenging computations relating to thermal constraints to offering computational performance. Heterogeneous multi-core systems comprise different processing units specialized for specific computational tasks, which can achieve superior performance compared to some platform of multi-core architectures as in Aubanel (2011).On the basis of the above, parallel programs are written efficiently to execute on multiple clusters to create parallelism in the program. These method calls are transferred into jobs that are executed efficiently on grids (Barry, 2003). Developments in computer architectures have shown that parallel computer systems are a viable way to overcome major hardware design challenges relating to energy-consumption, and thermal constraints, while offering high computational peak performance Martin, Benkner and Pllans (2012). Many applications are also sensitive to message or to message transfer bandwidth, but the effect is less pronounced than with overhead. It is a known fact that high capacity computing platform is expensive and is characterized by long-running, high processor-count jobs Barry (2003). Developing parallel applications have its own challenges in the field of parallel computing. According to Giacaman and Sinnen, (2011), there are theoretical challenges such as task decomposition, dependence analysis, and task scheduling.

In the loosely-coupled style, the assembly of separate parallel modules in a single application only requires interfaces, rather than the deeper structural changes or non-overlapped time, or processor division that might be required in the Single Program Multiple Data (SPMD) models Miller, Becker and Kale (2012). For a global task with other processors relevant data needs to be passed from processor to a processor through a message-passing mechanism Peizong, and Kedem (2002), Jaris and Alan (2003), since there is greater demand for computational speed and the computations must be completed within reasonable time period. This chapter concerns the estimated peak junction temperature of semiconductor devices, which is part of the thermal control systems Zarith et al., (2008) that treats the sequential algorithm of Parabolic Equation in solving thermal control process on printed circuit board. The theoretical properties of the 2-D stationary iterative finite difference discretization with the Domain Decomposition (DD) parallel communication approach using PVM to enable better flexibility in parallel execution and greater ease of parallel implementation across the different domain of block sizes**,** andemploying SPMD technique is emphasized in this chapter. The 2-D Active Thermal Control Process (2-D ATCP) is implemented on Geo Cluster with the ability to exploit inherent parallelism. We employ the message passing Master-Slave construction for the parallel platform application. The paper is organized as follows: Section2 introduces the finite difference method for the stationary iterative methods to be solved. Section 3 introduces the parallel overheads implementations. Section 4 introduces the numerical experiment and performance analysis. Section 5 gives the conclusion.

1.1 Related Work

The implementation of sequential algorithm in solving ATCP on printed circuit board with numerical finite difference method to design the discretization of the Partial Differential Equations (PDE) was implemented by Zarith et al., (2008). The implementation design approaches are either passive or active controlled. The passive controlled design is suitable for low to medium power dissipation, while the active thermal control system is suitable for industrial processing, and testing of die. In testing package

high-power integrated circuits, active thermal control is useful in providing die-level temperature stability Christopher and Lienhard (2005). They considered active test power sequences that contain many frequencies at various phase angles, each contributing to the temperature of the die. They developed a method of temperature variation of the die, and a method of controlling multiple frequency test sequences subject to a finite temperature tolerance. Sweetland et al., (2003) presented an ATCP of distributed-parameter system; with application to testing of packaged integrated circuit devices, requiring the die-level temperature be modulated to account for the distributed thermal capacitance and resistance of the device packaging. They also demonstrated fundamental limits of temperature control for typical devices under test conditions. Parallelization by time decomposition was first proposed by Lions, Madaya and Turinki, (2011) with motivation to achieve parallel real-time solutions, and even the importance of loop parallelism, loop scheduling have been extensively studied Aguilar and Leiss, (2005). Programming on heterogeneous many-core systems using explicit platform description to support programming by Martin, Banker and Pllans, (2012) constituted a viable approach for coping with power constraints in modern computer architectures, and can be formed across the whole computing landscape to high-end supercomputers and large-scale data centers. Chi-chung et al., (1994) used a network of workstations as a single unit for speeding up computationally intensive applications as a cost-effective alternative to traditional parallel computers. The platform provided a general and efficient parallel solution for time-dependent PDE. However, Tian and Yang, (2007) proposed an efficient parallel finite-difference scheme for solving Heat Equations numerically. They based it upon the overlapping DD method. Ewedafe and Rio, (2014) parallelized a 3-D ADI scheme using DD method with the SPMD technique on a Message Passing Interface (MPI) platform of clusters. On the same investigation, Ewedafe and Rio, (2014) used a numerical iterative scheme to solve 2-D Bio-Heat on MPI/PVM cluster systems with the SPMD technique. The Geo cluster are designed for application running on distributed memory clusters which can dynamically and statically calculate partition sizes based on the run-time performance of applications. We use the stationary iterative techniques (Jacobi and Gauss-Seidel) on the resulted matrices from the discretization of the 1-D ATCP model. Parallelization of the problem is carried out using DD parallel communication approach with PVM. The parallelization strategy and performance are discussed, and results of the parallel experiments are presented.

2. THE MODEL PROBLEM FOR 2-D ATCP

The mathematical model for the 2-D ATCP follows a similar model of Sweetland and Lienhard, (2003). For the transient response, the physical model of the device is reduced to the form:

$$\frac{\partial^2 v(x,t)}{\partial x^2}+\frac{\partial^2 v(y,t)}{\partial y^2}=\frac{1}{(b_t)}\frac{\partial v(x,y,t)}{\partial t} \qquad (2.1)$$

where b_t is the thermal diffusivity (that measure the ability of a material to conduct thermal energy relative to its ability to store thermal energy), and $b_t=\alpha=\frac{k}{\rho c_p}$, and k is the thermal conductivity $(W/(m.k))$, ρ is the density (kg/m^3), and c_ρ is the specific heat capacity $(J/(kgk))$ while ρc_p together can be considered the volumetric heat capacity $(J/(m^3k))$. Hence,

$$b_t\left(\frac{\partial^2 v(x,t)}{\partial x^2}+\frac{\partial^2 v(y,t)}{\partial^2 y}\right)=\frac{\partial v(x,y,t)}{\partial t}$$

$$\text{let} v(x,t)=v(y,t)=v(x,y,t)=V, \text{then we have} \qquad (2.2)$$

$$b_t\left(\frac{\partial^2 V}{\partial x^2}+\frac{\partial^2 V}{\partial y^2}\right)=\frac{\partial V}{\partial t}$$

we can then solve (2.2) by extending the 2-D explicit finite difference method to become:

$$\frac{\partial V}{\partial t}=V_t=\frac{V_{i,j+1}-V_{i,j}}{\Delta t}$$
$$\frac{\partial^2 V}{\partial x^2}=V_{xx}=\frac{V_{i+1,j}-2V_{i,j}+V_{i-1,j}}{(\Delta x)^2} \qquad (2.3)$$
$$\frac{\partial^2 V}{\partial y^2}=V_{yy}=\frac{V_{i,j+1}-2V_{i,j}+V_{i,j-1}}{(\Delta y)^2}$$

applyingEq. (2.3) on Eq. (2.2), the temperature of the explicit node is given by:

$$\frac{V_{i,j+1}-V_{i,j}}{\Delta t}=b_t\left(\begin{array}{l}\frac{V_{i+1,j}-2V_{i,j}+V_{i-1,j}}{(\Delta x)^2}+\\ \frac{V_{i,j+1}-2V_{i,j}+V^n_{i,j-1}}{(\Delta y)^2}\end{array}\right) \qquad (2.4)$$

2.1 Jacobi Scheme

If we applied the central difference and forward difference from the above into Eq. (2.2) we have:

$$V^{n+1}_{i,j}=V^n_{i,j}+\frac{b_t\Delta t}{\Delta^2}(V^n_{i+1,j}+V^n_{i-1,j}+V^n_{i,j+1}+$$
$$V^n_{i,j-1}-4V^n_{i,j})\ \ i=1\ldots,n,\ \ j=1\ldots,m \qquad (2.5)$$

it is stable in one spatial dimension only if $\Delta t/\Delta^2 \le 1/2$. In two dimensions this becomes $\Delta t/\Delta^2 \le 1/4$. Suppose we try to take the largest possible time step, and set $\Delta t = \Delta^2/4$. Then equation (4.3.5) becomes:

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^{n} + V_{i-1,j}^{n} + V_{i,j+1}^{n} + V_{i,j-1}^{n}) \quad (2.6)$$

thus the algorithm consists of using the average of V at its four nearest neighbor points on the grid (plus contribution from the source). This procedure is then iterated until convergence. This method is in fact a classical method with origins dating back to the last century, called "Jacobi's method".

2.2 Gauss-Seidel Scheme

Here we make use of updated values of V on the right hand of side of (2.6) as soon as they are available. The averaging is done in place instead of being copied from an earlier time step to a later one. If we proceed along the rows, incrementing j for fixed I, we write the computing formula as:

$$V_{i,j}^{n+1} = \frac{1}{4}(V_{i+1,j}^{n} + V_{i-1,j}^{n+1} + V_{i,j+1}^{n} + V_{i,j-1}^{n+1}) \quad (2.7)$$

if we had the approximate values of the unknowns at each grid points, this equation can be used to generate new values.

2.3 Gauss-Seidel Scheme

From (2.7), adding superscripts to show that a new value is computed from previous iterates, (x_i, t_j), If, instead of adding just the bracketed term, we add a larger value (thus "overrelaxing"), we get the new iterating relation

$$U_{i,j}^{(n+1)} = U_{i,j}^{(n)} + \omega\left[\frac{U_{i+1,j}^{(n)} + U_{i-1,j}^{(n+1)} + U_{i,j+1}^{(n)} + U_{i,j-1}^{(n+1)} - 4U_{i,j}^{(n)}}{4}\right]. \quad (2.8)$$

maximum acceleration is obtained for some optimum value of ω. This optimum value will always lie between 1.0 and 2.0. We define a scalar $\omega_n (0 < \omega_n < 2)$ and apply Eq. (2.7) to all interior points (i, j) and call it $U_{i,j}^{/}$. Hence, we have:

$$U_{i,j}^{n+1} = \omega_n U_{i,j}^{/} + (1-\omega_n) U_{i,j}^{n}$$

3.PARALLEL IMPLEMENTATION WITH PVM

The parallelization of the computations is implemented by means of grid partitioning technique (Coelho &Carvalho, 1993). 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 implementation was done on Geo Cluster consisting of 16 Intel Celeron CPU J 1900 at 1.99GHz quad core, and 495.7GB of Disk type. PVM Geist and Dongarra, (1998) is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource PVM is ideally suited for concurrent applications composed of many interrelated parts, and is very useful for the study of large-scale parallel computation. Partitioning strategy simply divides the problem into parts. Most partitioning formulations, however, require the results of the parts to be combined to obtain the desired result. Partitioning can be applied to the program data i.e. dividing the data and operating upon the divided data concurrently. This is called data partitioning or DD. When the domain is splinted, each block is given an identification 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. DD is used to distribute data between different processors; the static load balancing is used to maintain same computational points for each processor. Data parallelism originated the SPMD Laurant, (2001), thus, the finite difference approximation used in this paper can be treated as an SPMD problem. The SPMD model contains only a single program with multiple data and each process will execute the same code. To facilitate this within a single program, statements need to be inserted to select which portions of the code will be executed by each processor. The copy of the program is started by checking pvm_parent, it then spawns multiple copies of itself and passes then the array of tids. At this point, each copy is equal and can work on its partition of the data in collaboration with other processes. In the master model, the master program spawns and direct a number of slave program which perform computations. Any pvm task can initiate processes on the machine. The master calls pvm_mytid, which as the first pvm call, enrolls this task in the pvmsystem. It then calls pvm_spawn to execute a given number of slave programs on other machines in pvm. Each slave calls pvm_tid to determine its task id in the virtual machine, and then uses the data broadcast from the master to create a unique ordering from 0 to nproc minus 1. Subsequently, pvm_send and pvm_recv are used to pass messages between processors. When finished, all pvm programs call pvm_exit to allow pvm to disconnect any sockets to the process and keep track of which processes are currently running. A master program wakes up worker programs, assign initial data to the workers and let them work, receive results from workers, update and display them. The worker program works like this: receive initial data from master, exchange the edges data with the next-

door workers, and send the result to the master.Speed-up and efficiency are commonly used to measure the performance of a parallel code. The runtime of the original serial code is used as a measured of the runtime on one processor. In this context, runtime can be defined as the time that has elapsed from the first moment when the first processor actually begins execution of the program to the moment when the last process executes its last statement. In this present code, time is measured after the initialization of PVM, and before the domain decomposition. The time measurement ends after the writing of the last result, just before finalizing PVM. Only the timing of the day is considered. The parallel Speed-up (Sp) is the ratio of the runtime on one processor t1 to the runtime on P processor tp.The performance metric most commonly used is the speedup and efficiency which gives a measure of the improvement of performance experienced by an application when executed on a parallel system. Speedup is the ratio of the serial time to the parallel version run on N processors. Efficiency is the ability to judge how effective the parallel algorithm is expressed as the ratio of the speedup to N processors. In traditional parallel systems it is widely define as:

$$S(n) = {T(s)}/{T(n)}, \quad E(n) = {S(n)}/{n} \quad (3.1)$$

where*S(n)* is the speedup factor for the parallel computation, *T(s)* is the CPU time for the best serial algorithm, *T(n)* is the CPU time for the parallel algorithm using N processors, *E(n)* is the total efficiency for the parallel algorithm. However, this simple definition has been focused on constant improvements. A generalized speedup formula is the ratio of parallel to sequential execution speed. A thorough study of speedup models with their advantages and disadvantages. A different approach known as relative speedup, considers the parallel and sequential algorithm to be the same. While the absolute speedup calculates the performance gain for a particular problem using any algorithm, relative speedup focuses on the performance gain for a specific algorithm that solves the problem. The parallel efficiency and the corresponding speedup are commonly written as follows:

$$S_{par}(n) = {T(1)}/{T(n)}, \; E_{par}(n) = {S_{par}(n)}/{n} \quad (3.2)$$

The parallel efficiency takes into account the loss of efficiency due to data communication and data management owing to domain decomposition. The CPU time for the parallel computations with N processors can be written as follows:

$$T(n) = T_m(n) + T_{sd}(n) + T_{sc}(n) \quad (3.3)$$

where$T_m(n)$ is the CPU time taken by the master program, $T_{sd}(n)$ is the average slave CPU time spent in data communication in slaves, $T_{sc}(n)$ is the average CPU time expressed in computation in slaves. Generally,

$$T_m(n) = T_m(1), T_{sd}(n) = T_{sd}(1), T_{sc}(n) = {T_{sc}(1)}/{n}, \quad (3.4)$$

Therefore, the speedup can be written as:

$$S_{par}(n) = \frac{T(1)}{T(n)} = \frac{T_{ser}(1)+T_{sc}(1)}{T_{ser}(1)+T_{sc}(1)/n} < \frac{T_{ser}(1)+T_{sc}(1)}{T_{ser}(1)} \quad (3.5)$$

Where $T_{ser}(1) = T_m(1) + T_{sd}(1)$, which is the part that cannot be parallelized. This is called Amdahl's law, showing that there is a limiting value on the speedup for a given problem. The corresponding efficiency is given by:

$$Epar(n) = \frac{T(1)}{nT(n)} = \frac{Tser(1)+Tsc(1)}{nTser(1)+Tsc(1)} < \frac{Tser(1)+Tsc(1)}{nTser(1)} \quad (3.6)$$

the parallel efficiency represents the effectiveness of the parallel program running on *N* processors relative to a single processor. However, it is the total efficiency that is of real significance when comparing the performance of a parallel program to the corresponding serial version. Let $T_s^{No}(1)$ denotes the CPU time of the corresponding serial program to reach a prescribed accuracy with *No* iterations, $T_{B=B}^{N_1 L}(n)$ denotes the total CPU time of the parallel version of the program with B blocks run on *N* processors to reach the same prescribed accuracy with N_i iterations including any idle time. The superscript *L* acknowledges degradation in performance due to the load balancing problem. The total efficiency in (3.1) can be decomposed as follows:

$$E(n) = \frac{T_s^{N_o}(1)}{n.T_{B=B}^{N_1 L}(n)} = \frac{T_s^{N_o}(1)}{T_{B=1}^{N_o}(1)} \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \frac{T_{B=B}^{N_0}(1)}{T_{B=B}^{N_1}(1)} \frac{T_{B=B}^{N_1}(1)}{T_{B=B}^{N_1}(n)} \frac{T_{B=B}^{N_1}(n)}{T_{B=B}^{N_1 L}(n)}, \quad (3.7)$$

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 (3.4) and (3.1), we obtain:

$$E_{load}(n) = \frac{T_{B=B}^{N_1}(n)}{T_{B=B}^{N_1 L}}, E_{par}(n) = \frac{T_{B=B}^{N_1}(1)}{n.T_{B=B}^{N_1}(n)}, \quad Enum(n) = \frac{T_s^{N_o}(1)}{T_{B=B}^{N_1}(1)} = \frac{T_s^{N_o}(1)}{T_{B=1}^{N_o}(1)} \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \frac{T_{B=B}^{N_o}(1)}{T_{B=B}^{N_1}(1)}, \quad (3.8)$$

when*B=1* and *n = 1*, $T_m(1) + T_{sd}(1) << T_{sc}(1)$, then $T_{B=1}^{N_o}(1)/T_s^{N_o}(1) \approx 1.0$. We note that $T_{B=B}^{N_o}(1)/T_{B=B}^{N_1}(1) = N_o/N_1$. Therefore,

$$E_{num}(n) = E_{dd}\frac{N_o}{N_1}, \quad E_{dd} = \frac{T_{B=1}^{N_o}(1)}{T_{B=B}^{N_o}(1)} \quad (3.9)$$

we call (3.9) 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. The second term N_o / N_1 in the right-hand side of (3.9) represents the increase in the number of iterations required by the parallel method to achieve a specified accuracy compared to the serial method. 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. The activities of communication are slow compare to computation, often by order of magnitude. The communication costs are measured in terms of latency and bandwidth. The communication costs comprised of the physical transmission costs and the protocol overhead. The overhead is high in heterogeneous networks, where data may have to be converted to another format. The communication cost can be reduced but not avoided. The non-blocking communication is employed across SPMD to reduce the problem of blocking. Factors considered in communication include; Cost of communications which implies overhead and required synchronization between tasks, latency versus bandwidth, synchronization communication, and efficiency of communication. We have our grid distributed in a block fashion across the processors, the values for the ghost cells are calculated on neighboring processors and sent using PVM class. Synchronization is the coordination of parallel tasks associated with communication. A routine that returns when the transfer has been completed is a synchronous message passing and performs two actions: transferring of data then synchronous processes through send / receive operations.

4. NUMERICAL EXPERIMENTS AND PERFORMANCE ANALYSIS

Consider the 2D-ATCP of the form in Eq. (4.1) with boundary and initial conditions as:

$$U(x, y, 0) = F(x, y), (x, y, t) \in R \times \{0\}, \quad (4.1)$$

and $U(x, y, t)$ is specified on the boundary of $R, \partial R$ by

$$U(x, y, t) = G(x, y, t), (x, y, t) \in \partial R \times (0, T], \quad (4.2)$$

where for simplicity we assume that the region *R* of the *xy*-plane is a rectangle. Consider the 2-D ATCP (4.1) with the auxiliary conditions (4.2) and (4.3). The region R is a rectangle defined by

$$R = \{(x, y) : 0 \le x \le L, 0 \le y \le M\}.$$

At the point $P(x_i, y_j, t_k)$ in the solution domain, the value of $U(x, y, t)$ is denoted by $U_{i,j,k}$ where $x_i = i\Delta x,\ y_j = j\Delta y$ *for* $0 \le i \le (m+1), 0 \le j \le (n+1)$ *and* $\Delta x = L/(m+1),\ \Delta y = M/(n+1)$. The increment in the time t, Δt is chosen such that $t_k = k\Delta t$ *for* $k = 0,1,2,\ldots$ for simplicity of presentation, we assume that m and n are chosen so that $\Delta x = \Delta y$ and consequently the mesh ratio is defined. We obtain speedup and efficiency for various mesh sizes i.e. 100x100 mesh size to 1000x1000 mesh size are listed in Tables 1 to 3. The Tables show the parallel time decreasing as the number of processors increase for using the stationary iterating schemes. The speedup and efficiency versus the number of processors are shown in Fig. 3 and Fig. 4, respectively. The results in the Tables show that the parallel efficiency increases with increasing grid size, 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. Performance begins to degrade with an effect caused by the increase in communication overhead as the mesh increases. 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.To obtain a high efficiency, the slave computational time $T_{sc}(1)$ should be significantly larger than the serial time T_{ser}. In this present program, the CPU time for the master task and the data communication is constant for a given grid size and sub-domain. Therefore, the task in the inner loop should be made as large as possible to maximize the efficiency. The speed-up and efficiency obtained for various sizes of 100x100 to 1000x1000 are for various numbers of sub-domains;

Table 1: Execution time and Speed-up for the JacobiScheme on the 2-D ATCP

P	mxn	T1	T2	T3	T4	Sp1	Sp2	Sp3	Sp4
1	100x100	0.52	0.48	0.45	0.44	1	1.08	1.15	1.18
2	200x200	2.97	1.95	1.83	1.82	1	1.52	1.62	1.63
3	300x300	6.66	6.26	4.39	4.16	1	1.06	1.52	1.60
4	500x500	14.78	12.98	12.17	12.13	1	1.14	1.21	1.22
5	1000x1000	62.46	50.83	46.58	46.74	1	1.23	1.34	1.34

Table 2: Execution time and Speed-up for the Gauss-Seidel Scheme on the 2-D ATCP

P	mxn	T1	T2	T3	T4	Sp1	Sp2	Sp3	Sp4
1	100x100	0.52	0.51	0.45	0.45	1	1.02	1.16	1.16
2	200x200	2.48	2.31	1.99	1.83	1	1.07	1.25	1.36
3	300x300	6.22	5.13	4.44	4.16	1	1.21	1.40	1.50
4	500x500	16.94	13.95	12.28	12.21	1	1.21	1.38	1.39
5	1000x1000	65.92	51.25	46.61	46.43	1	1.29	1.41	1.42

Table 3: Execution time and Speed-up for the SOR Scheme on the 2-D ATCP

P	mxn	T1	T2	T3	T4	Sp1	Sp2	Sp3	Sp4
1	100x100	2.14	0.58	0.45	0.42	1	3.69	4.76	5.10
2	200x200	9.51	2.35	1.80	1.73	1	4.05	5.28	5.50
3	300x300	22.64	5.34	4.08	3.93	1	4.24	5.55	5.76
4	500x500	69.08	15.66	12.73	10.95	1	4.41	5.43	6.31
5	1000x1000	78.9	60.64	44.42	44.33	1	5.13	7	7.01

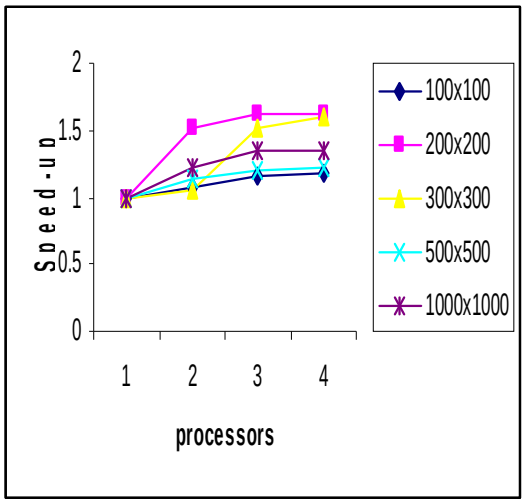

Fig. 3: Sp for Jacobi Scheme

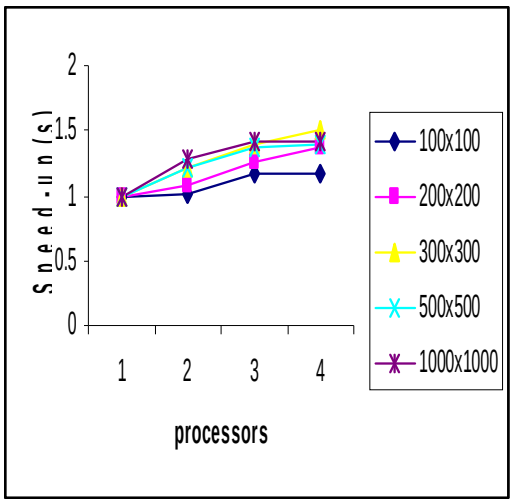

Fig. 4: Sp for Gauss-Seidel Scheme

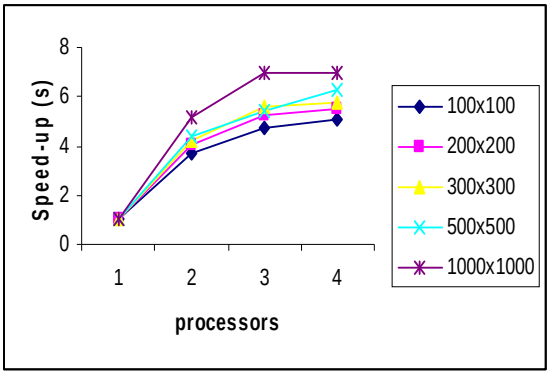

Fig. 3: Speed-up for SOR Scheme

5. CONCLUSIONS

In this paper, we consider the 2-D ATCP with PVM and using DD parallel implementation with stationary methods on parallel platform on the resulted matrices of the 2-D ATCP model with the flexibility of a parallel platform on the general-purpose message-driven execution. The algorithms presented show significant improvement when implemented on the above number of processors. In addition to the ease of use compared to other common approaches, the results show negligible overhead with effective load scheduling which produce the expected inherent speedups. Different speedups and efficiencies were recorded for the different schemes and we observed that the schemes exhibited convergence to stability when accessed with various mesh sizes and do conform to unity.

References

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

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

[3]. Barry, W. Michael, A. (2003).Parallel Programming Techniques and Application using Networked Workstation and Parallel Computers. Prentice Hall, New Jersey

[4]. Burden, R. L., Faires, J. D. (1993).Numerical Analysis 5th ed.' PWS Publishing Company, Boston, 1993

[5]. Chi-chung, H., Ka-keung, C., Man-Sheung Yuen, G., Hamdi, M. (1994). Solving PDE on Network of Workstation. IEEE, pp194 – 200

[6]. Christopher, C. R., Lienhard, J. H. (2005). Active Thermal Control of Distributed Parameter Systems Excited at Multiple Frequencies. Journal of Heat Transfer

[7]. Coelho, P. J., Carvalho, M. G. (1993). Application of a Domain Decomposition Techniqueto the Mathematical Modeling of Utility Boiler.Journal of Numerical Methods in Eng., 36 pp 3401 – 3419

[8]. Ewedafe, S. U., Rio, H. S. (2014). On the Parallel Design and Analysis for 3-D ADI Telegraph Problem with MPI.Int'l Jour. Of Advanced Compt. Sci. and Applications, 5 (4), 2014

[9]. Ewedafe, S. U., Rio, H. S. (2014). Domain Decomposition of 2-D Bio- Heat Equationon MPI/PVM Clusters. Int'l Journal of Emerging Trends of Technology in Compt. Science, 3 (5), 2014

[10]. Fahnl, P. Lienhard A., Slocum A. (1999). Thermal management and Control in Testing Packaged Integrated Circuit (IC) Devices' Proc. 34th Inter Society Energy Conversion Conference

[11]. Geist, A. Beguelin, J. Dongarra, (1998). Parallel Virtual Machine (PVM). Cambridge MIT Press

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

[13]. Gupta, M, Banerjee, P. (1992). Demonstration of Automatic Data Partitioning for Parallelizing Compilers on Multi-Computers. IEEE Trans. Parallel Distributed System, 3 (2), pp 179 – 193

[14]. Jaris, K., Alan, D. (2003). A High-Performance Communication Service for Parallel Computing on Distributed System, Parallel Computing, 29, 851 – 878

[15]. Lauran, H. (2001). A method for automatic placement of communications in SPMD parallelization.Parallel Computing,27, 1655 – 1664

[16]. Lions, J. L., Maday, Y., Turinki, G. (2011).Parareal in time discretization of PDE.Computes, rendus de lacadimie des sciences – series 1 – mathematics,332 (7), 2011, 661 – 668
[17]. Martin, S., Benkner, S., Pllans, D. (2012).Using Explicit Parallel Description to Support Programming of Heterogeneous Many-Core Systems.Parallel Computing,38, pp 52 – 65
[18]. Miller, P., Becker, L. Kale, E. (2012). Using Shared Arrays in Message-Driven ParallelPrograms. Parallel Computing,38, 66 - 74
[19]. Peizong L., Kedem, Z. (2002). Automatic Data and Computation Decomposition on Distributed Memory Parallel Computers. ACM Transactions on Programming Languages and Systems, 24,1 – 50.
[20]. Sweetland, M., Lienhard, H. (2003).Active Thermal Control of Distributed Parameter System with Application to Testing of Packaged (IC) Devices.ASME Journal of Heat Transfer 165 – 174
[21]. Tian, M., Yang, D. (2007).Parallel Finite Difference Schemes for Heat Equations Based on Overlapping Domain Decomposition.Applied Maths&Compt, 18, 2007, pp 1276 – 1292
[22]. Zarith, S., Ghaffar, A., Alias, N., Sham, F., Hassan, A., Hassan, H. (2008).Sequential Algorithms of Parabolic Equation in Solving Thermal Control Process on Printed Circuit Board.Jour. Fundamental Science,4, 379 - 385

AUTHOR

Ewedafe Simon Uzezi received the B.Sc.and M.Sc. degrees in Industrial-Mathematics and Mathematics from DeltaStae University and The University ofLagos in 1998 and 2003 respectively. Hefurther obtained his PhD in Numerical Parallel Computing in 2010 from the Universityof Malaya, Malaysia. In 2011 he joined UTAR in Malaysia as an Assistant Professor, andlater lectured in Oman and Nigeria as Senior Lecturer inComputing. Currently, he is a Senior Lecturer in UWIJamaica in the Department of Computing.