Parallel Numerical Solution of Linear PDEs Using Implicit and Explicit Finite Difference Methods

Shakeel Ahmed Kamboh, Jane Labadin, Khuda Bux Amur, Muhammad Afzal Soomro, and Syed Muhammad Saeed Ahmed

***Abstract*—This paper presents the parallel computation of numerical solution of 3D linear PDEs discretized by using the Finite Difference Method (FDM). The parallel computation of numerical solution is carried out by two different solution methods. First, a typical discretized PDE is expressed as an implicit function of unknowns that lead to a system of linear algebraic equations represented in standard matrix form as AX=B. Next, the discretized PDE is explicitly defined for the grid points lying on the computational domain. Both the implicit and explicit forms of the problem are parallelized by using data parallelism technique. The main objective of this study is to analyze the computational time (sec) and memory (Mb) requirements for the implicit (matrix based) and explicit (matrix free) methods. For the testing and implementation purpose a typical 3D Poisson's equation with Dirichlet boundary conditions is used. The parallel methods are executed on MATLAB parallel computing environment. The results revealed that the implicit solution method uses huge amount of computer memory than the explicit approach. Also, the parallel computational time is higher in former than later. This concludes that the explicit method offers good performance as compared to implicit method implemented on parallel system.**

***Index Terms*—parallel computing, numerical solution of PDEs, finite difference method, implicit and explicit methods.**

I. Introduction

Linear Partial Differential Equations (PDEs) are extensively used to simulate many real world problems in various fields of science, engineering and technology. The Poisson's equation, Laplace's equation, wave equation, heat equation, Helmholtz equation and Klein–Gordon equation are some of the well- known examples of linear PDEs. To solve the problems represented in terms of linear PDEs the different analytical and numerical methods have been used. However, the numerical methods are preferred than the analytical methods because they can be efficiently implemented on computers and use less storage memory than the analytical methods [1]. Among the different numerical methods the finite difference method (FDM) is commonly used to solve the linear PDEs due to its simplicity and ease of implementation [2-4]. The finite difference discretization schemes often come out as implicit or explicit functions of the values of dependent variable related to a given PDE. In an implicit scheme, the computation of the values of dependent variable are defined by a set of algebraic linear equations expressed in matrix form and solved by any iterative method. Whereas, in explicit scheme the unknown values of dependent variables are directly computed in terms of known values. The solution process of explicit scheme is repeated iteratively for each unknown. Often, it is possible to obtain implicit scheme from given explicit scheme and vice versa.

Shakeel Ahmed Kamboh, Khuda Bux Amur, Muhammad Afzal Soomro, Muhammad Saeed Ahmed Syed and Jane Labadin, Department of Mathematics and Statistics, Quaid-e-Awam University of Engineering, Science and Technology, Nawabshah, Pakistan
Email: shakeel.maths@yahoo.com

In practice, most of the problems formulated by using linear PDEs are attempted to solve by FDM exist in three dimensions (3D) and in large scales. Consequently, require high computing efforts. Even some times it becomes quite difficult to achieve the numerical solution on sequential computers with limited resources. In such conditions, the parallelization of the usual sequential algorithms provides the benefits of High Performance Computing (HPC) and reduces computational complexity. The parallelism may be achieved by the utilization of different computing architectures such as multi-core systems, GPUs, clusters of sequential computers, remote clusters, etc. However, before using any computing architecture the art of parallelization of sequential algorithms is considered to be one of the most influential steps towards the HPC.

This study is devoted to the parallel computation of numerical solution algorithms arising from the explicit and implicit finite difference discretization of linear PDEs. The paper begins with the Introduction Section providing the motivation for the research. Next Section presents the brief background of some related works and highlights the main objective of the current study. The methodology is presented in Section-III, where a typical 3D linear PDE is discretized using FDM. The parallelization and implementation of solution algorithms based on implicit and explicit FDM is addressed. Section-IV exhibits the numerical results and compares the parallel performance of implicit and explicit FDM in terms of computational time and memory requirements. Finally, Section-V draws conclusion and offers some directions for future work.

II. Background

For the computationally intensive problems solved by FDM the large number of unknowns is computed on the parallel systems. The parallelization of FDM is based on the choice of the implicit or explicit solution schemes. In this context, many studies can be found in literature. For example, [5-9] and [10-16] used implicit and explicit finite difference schemes respectively to solve 1D, 2D and 3D

 ISSN – 1999-4974

linear PDEs in parallel. They used different parallel computing architectures and parallel programming languages to solve the large scale problems in fluid flow, electromagnetism, weather and climate modeling, etc. Most of the parallel implicit and explicit schemes have been implemented independently but their comparison is rarely found in literature. Also, their implementation is often done on expensive parallel architectures using FORTRAN, C, C++ and CUDA. However, the MATALB provides more flexibility to implement the parallel applications on parallel/distributed computing environment. Therefore, this study is aimed at the parallelization and implemetation of implicit and explicit finite difference methods on easily available architectures using MATLAB. The main objective is to investigate the storage memory (Mb) requirements and the computing time (sec) on parallel system for such numerical solution schemes.

III. METHODOLOGY

The main objective of this study is to investigate the memory (Mb) requirements and the computing time (sec) on parallel systems for implicit and explicit solution schemes of linear PDEs. Consequently, for the testing and implementation purpose a typical linear PDE with Dirchilet boundary conditions called Poisson's equation is chosen. This equation is widely used to simulate various physical phenomena. Yet, the methodology steps to be presented may also be used for other kinds of linear PDEs with minor changes. The general 3D Poisson's equation in Cartesian coordinates is given as follows:

$$\frac{\partial^2 u(x,y,z)}{\partial x^2}+\frac{\partial^2 u(x,y,z)}{\partial y^2}+\frac{\partial^2 u(x,y,z)}{\partial z^2}=f(x,y,z) \quad , \qquad (1)$$

where u is the dependent variable representing any physical phenomenon, (x, y, z) are the space coordinates and f is the given forcing function. It is considered that the Eq. (1) is applied to simulate u numerically with given initial and boundary conditions imposed on the computational domain D as shown in Fig.1.

To obtain the numerical solution by FDM Eq. (1) is discretized by using 7-point central finite difference scheme and is expressed as given below,

$$\frac{u_{i+1,j,k}-2u_{i,j,k}+u_{i-1,j,k}}{h_1^2}+\frac{u_{i,j+1,k}-2u_{i,j,k}+u_{i,j-1,k}}{h_2^2}+\frac{u_{i,j,k+1}-2u_{i,j,k}+u_{i,j,k-1}}{h_3^2}=f_{i,j,k} \quad , \qquad (2)$$

where $i=1, 2, \ldots, m+1$, $j=1, 2, \ldots, n+1$ and $k=1, 2, \ldots, s+1$ are the indices of the grid points and $h_1=(x_m-x_o)/m$, $h_2=(y_m-y_o)/n$ and $h_3=(z_m-z_o)/s$ are the increments along x, y, and z directions respectively. While $m=G_{rf}.c_x$, $n=G_{rf}.c_y$, and $s=G_{rf}.c_z$ are the total number of finite difference cells with initial number of cells c_x, c_y, and c_z along x, y and z axis respectively, refined by a grid refinement factor G_{rf}. A schematic of discretized computational domain with $c_x=c_y=c_z=10$ and the $G_{rf}=2$ is illustrated by Fig. 2. Note that for the current problem the computational domain D is composed of $G_p=(m+1)(n+1)(s+1)$ grid points. Where $N=(m-1)(n-1)(s-1)$ are the unknown interior points and $K=G_p-N$ are the known boundary points at outer edges of the domain.

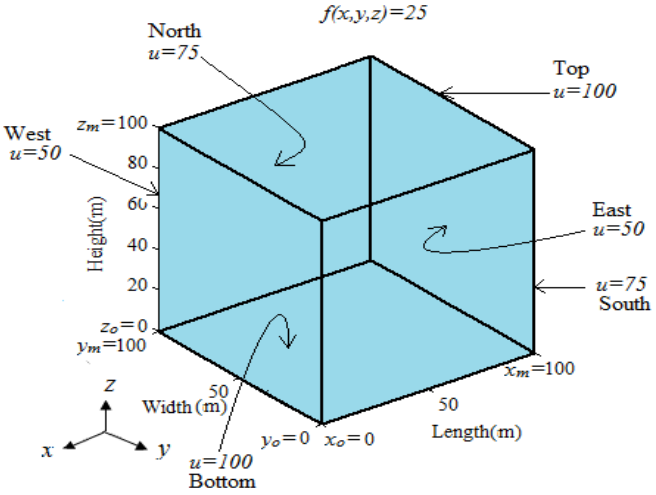


Fig. 1. Computational domain D with boundary conditions.

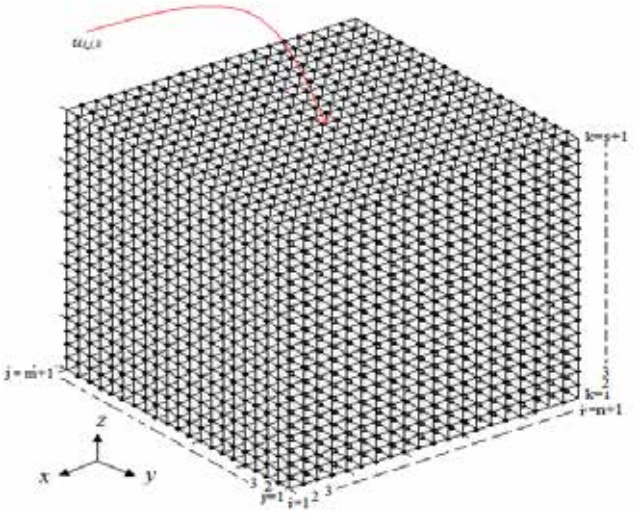


Fig. 2. Schematic of discretized computational domain (3D mesh) with $c_x=c_y=c_z=10$ and $G_{rf}=2$.

To come up with an implicit or an explicit scheme Eq. (2) is further simplified and arranged as;

$$c_1u_{i,j,k}-c_2u_{i-1,j,k}-c_2u_{i+1,j,k}-c_3u_{i,j-1,k}-c_3u_{i,j+1,k}-c_4u_{i,j,k-1}-c_4u_{i,j,k+1}=c_5f_{i,j,k} \quad , \qquad (3)$$

or Eq. (3) is divided by c_1 to normalize the constants, thus we have;

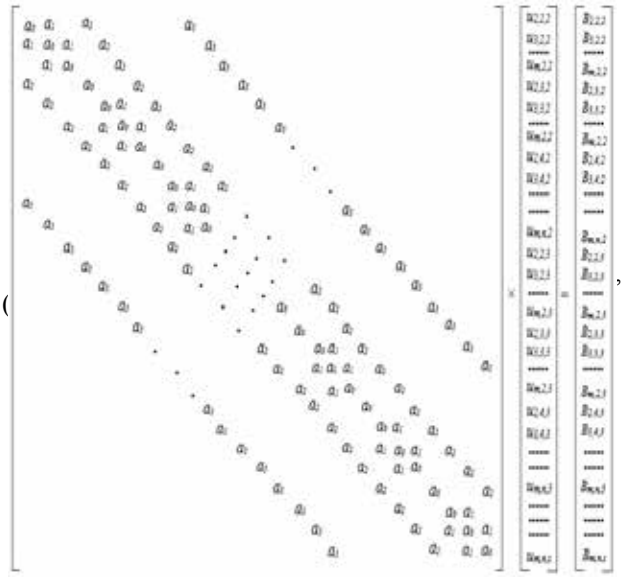
$$a_0u_{i,j,k}-a_1u_{i-1,j,k}-a_1u_{i+1,j,k}-a_2u_{i,j-1,k}-a_2u_{i,j+1,k}-a_3u_{i,j,k-1}-a_3u_{i,j,k+1}=a_4f_{i,j,k} \quad , \qquad (4)$$

where,

ISSN – 1999-4974

$c_1 = 2\left((h_1h_2)^2 + (h_1h_3)^2 + (h_2h_3)^2\right), c_2 = (h_2h_3)^2,$
$c_3 = (h_1h_3)^2,\ c_4 = (h_1h_2)^2, c_5 = (h_1h_2h_3)^2,$
$a_0 = c_1 / c_1 = 1,\ a_1 = c_2 / c_1,\ a_2 = c_3 / c_1,$
$a_3 = c_4 / c_1,\ a_4 = c_5 / c_1.$

Eq. (4) is the implicit finite difference scheme and is considered as a system of linear equations in N unknown values of u. This system can be transformed into standard matrix form as $AX=B$. Since the problem in hand is three dimensional and contains N unknowns for specific G_{rf}. To transform 3D data points into a 2D matrix of order $N \times N$ the unknown interior points are arranged into a natural linear order and then reshaped into a coefficient matrix A, a column vector X and the right hand side column vector B as given in the following matrix equation;



In Eq. (5) the coefficient matrix A is diagonally dominant thus the system can be solved iteratively. However, the implicit scheme requires much matrix manipulation. Therefore, it is worth to compute all unknowns directly without using matrix form. It can be done by defining $u_{i,j,k}$ explicitly; thus by rearranging Eq. (4) the explicit finite difference scheme is obtained as;

$$u_{i,j,k} = \frac{a_1(u_{i+1,j,k} + u_{i-1,j,k}) + a_2(u_{i,j+1,k} + u_{i,j-1,k}) + a_3(u_{i,j,k+1} + u_{i,j,k-1}) + a_4 f_{i,j,k}}{a_0}, \quad (6)$$

Eq. (6) can be solved iteratively; thus for each of the iterations γ the iterative explicit solution scheme is expressed as;

$$u^{\gamma+1}_{i,j,k} = \frac{a_1(u^{\gamma}_{i+1,j,k} + u^{\gamma}_{i-1,j,k}) + a_2(u^{\gamma}_{i,j+1,k} + u^{\gamma}_{i,j-1,k}) + a_3(u^{\gamma}_{i,j,k+1} + u^{\gamma}_{i,j,k-1}) + a_4 f_{i,j,k}}{a_0}. \quad (7)$$

In many practical problems the numerical solution of the above schemes is required at fine meshes because of the numerical accuracy or computation at very small increments. Though, refined meshes make the problem computationally intensive in terms of processing time. In order to reduce computational complexity such problems need to parallelize. The following sections demonstrate the parallelization of the above implicit and explicit schemes to be solved by perfectly parallel Jacobi iterations.

A. Parallelization of implicit scheme

Refering to Eq. (5) the sequential version of Jacobi algorithm is defined by the following formula;

$$x_i^{\gamma+1} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i}^{N} a_{ij} x_j^{\gamma}) \quad . \quad (8)$$

This algorithm is perfectly parallel and can be parallelized by using the data parallelism technique [17]. Only few important steps are required to parallelize Eq. (8) since all the solution values in any iteration depend completely on the values of preceding iteration. Thus the parallel process can be run independently on all the parallel workers (processors). The most significant step requires the availability of initial solution vector on all workers. To achieve this goal the mpi gather need to apply. For the sake of brevity the important steps are listed as below:

Step 1: Partition the matrices *A, X, B* and the column index *j* into *P* sub-matrices and define their local part as a_L, x_L, b_L and j_L respectively.

Step 2: Set the initial solution vector $x_0 = 0$, and the error tolerance ε.

Step 3: Set the iterations γ then compute the solution vector locally by the equivalent parallel formula as;

$$x_{L i_L}^{\gamma+1} = \frac{1}{a_{i_L j_L}}(b_{L i_L} - \sum_{j \neq i_L}^{N} a_{i_L j} x_0^{\gamma}) \quad (9)$$

Step 4: Gather the local solutions x_L from each worker and define it by variable x_1.

Step 5: Check the convergence; if $\max(|x_1 - x_0|) < \varepsilon$ then stop the process, otherwise continue.

Step 6: Set $x_0 = x_1$ and repeat the steps from 3 to 5.

B. Parallelization of explicit FDM

In contrast to matrix based Jacobi algorithm in the explict Jacobi algorithm the solution at any interior point depends on its six neighboring points. Therefore, the parallelization of explicit method requires more steps. Referring to Eq. (7) the most important steps involve the domain partitioning and message paasing to exchange the neighboring points. The domain *D,* can be partitioned into subdomains along any axis. A schematic of 1-dimensional domain partitioning of the present problem is illustrated in Fig. 3. After the domain is partitioned and distributed to parallel workers the

 ISSN – 1999-4974

neighboring data points between any two workers need to exchange in each iteration. Then the computations need to operate locally on parallel workers.

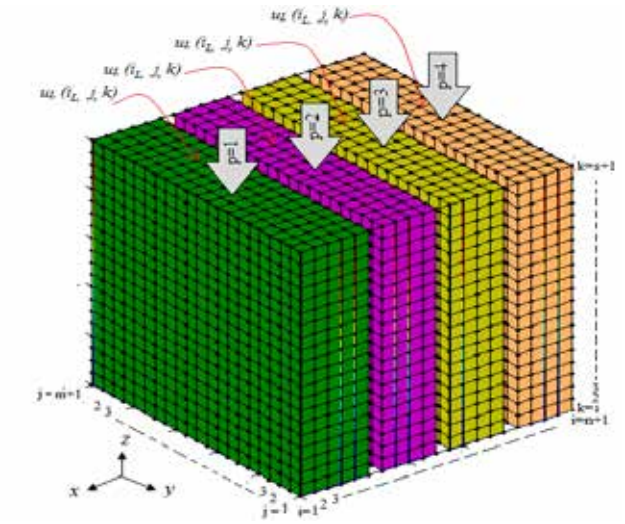


Fig. 3. A schematic of 1-dimensional domain partitioning along x-axis distributed to *P=4* workers.

The major steps required for the parallelization of explicit scheme are listed below:

Step 1: Partition the domain containing u and f along any axis (say x-axis) and define its local parts as u_L and f_L on parallel workers.

Step 2: Set the initial solution array $u0_L$ and the error tolerance ε.

Step 3: Start the iterations γ.

Step 4: Exchange and share the neighborhood data points of $u0_L$ then compute the solution locally by the equivalent parallel formula as;

$$u_{L_{i,j,k}}^{\gamma+1} = \frac{a_1(u0_{L_{i_L+1,j,k}}^{\gamma} + u0_{L_{i_L-1,j,k}}^{\gamma}) + a_2(u0_{L_{i_L,j+1,k}}^{\gamma} + u0_{L_{i_L,j-1,k}}^{\gamma}) + a_3(u0_{L_{i_L,j,k+1}}^{\gamma} + u0_{L_{i_L,j,k-1}}^{\gamma}) + a_4 f_{L_{i_L,j,}}}{a_0} \quad (9)$$

Step 5: Obtain the local error $\varepsilon_L = \max(\max(\max(|u_L - u0_L|)))$.

Step 6: Find the global error $\varepsilon_g = \max(\varepsilon_L)$ and check for convergence, if $\varepsilon < \varepsilon_g$ then stop the process, otherwise continue.

Step 7: Set $u0_L = u_L$ and repeat the steps from 3 to 6.

C. Implementation of parallel implicit and explicit FDM

Both the implicit and explicit solution schemes are implemented on MATLAB parallel computing environment using local scheduler with maximum *P=8* workers (or labs). A shared memory type parallel computing system with 2.80 GHz intel processor, 4GB RAM, 150 GB hard drive and 32-bit Windows 7 operating system is utilized. The parallel algorithms are tested for different problem sizes depending on the grid refinement factor as given in Table I. It is noticeable from the Table I that the implicit method uses huge amount of memory to store coefficient matrix *A* than to store *u* in explicit method. As far as the problem size increases the implicit scheme appears to be impractical in terms of memory requirement. However, if the matrix *A* is stored using sparse matrix function *S(A)* then less memory is utilized as compared by defining matrix *A* in common . Yet, the explicit method remains dominant over implicit due to very small memory usage. The algorithms are run for each grid size with predefined error tolerance of $\varepsilon = 0.0001$. First, the sequential algorithms are tested on *P=1* and then the corresponding parallel algorithms are run at *P=2, 4, 6,* and *8* workers. The convergence iterations for both the methods are noted and are given in Table II. It can be seen that for each data size the methods converge in about same number of iterations with small difference. The computational time (sec) and the data received D_p (Mb) per worker (lab) for each grid refinement factor is noted. The parallel performance results of implicit and explicit schemes are listed in Table III and Table IV respectively.

TABLE I.
PROBLEM SIZE AND THE REQUIRED MEMORY FOR IMPLICIT AND EXPLICIT SOLUTION METHOD

	Problem Size			**Memory Storage (Mb)**		
G_{rf}	G_p	K	N	*Explicit, u*	*Implicit, A*	*Implicit, S(A)*
1	1331	602	729	0.0102	4.0546	0.0411
2	9261	2402	6859	0.0687	358.9316	0.4029
3	29791	5402	24389	0.2273	4538.1418	5.0943
4	68921	9602	59319	0.5258	26845.8844	30.1359
5	132651	15002	117649	1.0100	105600.6409	118.5424
6	226981	21602	205379	1.7300	321811.9327	361.2511
7	357911	29402	328509	2.7300	823350.2432	924.2547
8	531441	38402	493039	4.0500	1854610.1038	2081.8991
9	753571	48602	704969	5.7500	3791666.3434	4256.3485
10	1030301	30002	970299	7.8600	7182923.5031	8063.2162

TABLE II.
PROBLEM SIZE AND THE ITERATIONS FOR IMPLICIT AND EXPLICIT CONVERGED SOLUTION

	Problem Size			**Number of Iterations, γ**	
G_{rf}	G_p	K	N	*Explicit, u*	*Implicit, u*
1	1331	602	729	226	230
2	9261	2402	6859	798	811
3	29791	5402	24389	1649	1661
4	68921	9602	59319	2746	2756
5	132651	15002	117649	4065	4077
6	226981	21602	205379	5587	5599
7	357911	29402	328509	7299	7312
8	531441	38402	493039	9188	9196
9	753571	48602	704969	11242	11261
10	1030301	30002	970299	13452	13464

ISSN – 1999-4974

TABLE III.
SEQUENTIAL AND PARALLEL PERFORMANCE USING IMPLICIT SOLUTION METHOD

G_{rf}	Sequential Time (sec)	Parallel Computing Time (sec)				D_p (Mb)
	P=1	*P=2*	*P=4*	*P=6*	*P=8*	
1	25.21	37.82	58.24	133.94	468.79	1.47
2	21.94	30.71	38.39	86.38	280.73	20.82
3	127.03	165.14	180.83	388.78	1166.33	120.76
4	500.22	600.27	510.23	1048.01	2882.02	473.38
5	1694.52	1863.97	1491.17	2833.23	6374.77	1584.24
6	3564.49	3564.49	2762.48	4972.46	9944.91	3270.15
7	9037.63	8585.75	6439.31	11268.79	19720.39	8268.36
8	17091.01	15381.91	10767.33	17766.10	31090.68	15753.5
9	30532.71	25952.81	16869.32	26990.92	44535.02	28194.0
10	49824.94	39859.95	23915.97	35873.95	57398.33	45541.98

TABLE IV.
SEQUENTIAL AND PARALLEL PERFORMANCE USING EXPLICIT SOLUTION METHOD

G_{rf}	Sequential Time (sec)	Parallel Computing Time (sec)				D_p (Mb)
	P=1	*P=2*	*P=4*	*P=6*	*P=8*	
1	0.11	0.58	16.25	38.75	82.50	0.23
2	2.24	3.26	53.83	121.32	207.66	2.75
3	15.93	13.40	116.60	214.47	333.20	12.23
4	70.47	47.19	220.33	460.00	690.66	35.44
5	253.98	141.28	459.44	978.32	1518.88	81.00
6	787.95	338.55	903.61	1710.83	2907.22	159.06
7	1702.00	1173.64	1320.91	2962.72	4141.82	281.31
8	2934.34	1832.00	2125.03	4379.05	6250.06	460.66
9	5015.61	3583.00	4381.36	6144.32	8762.72	711.16
10	9365.14	5231.00	7901.00	8418.45	12802.00	1002.00

IV. RESULTS AND ANALYSIS

This section evaluates the numerical results and the performance of parallel system tested for the implicit and explicit finite difference methods. The numerical solution profiles of dependent variable *u* and its gradient are shown in Fig. 4 and Fig 5. respectively. The numerical profiles appear to be realistic and exhibit the transition of solution from high values to low values. For more refined discretization the numerical solution improves significantly. From the results as given in Table III and Table IV it can be noted that both the computing time and data exchange per lab in implicit method is much higher than the explicit method. It is because of the complexity of assigning and storing of large number of elements of matrix *A*. However, the parallel system scales well for *P=2* and *P=4* workers as shown in Fig. 6 and Fig. 7. Speedup in computing time can be observed for higher G_{rf}. Parallel explicit method appears best at *P=2* while implicit parallel method scales well at *P=4*. However, like sequential time the parallel time in explicit schemes is much better than the implicit method. Overall behavior of parallel system for both the methods shows that increasing the number of workers will not increase the performance of parallel system on the architecture used in this study. However, it is assumed that the parallel algorithms may produce better performance on other distributed memory or GPU type architectures.

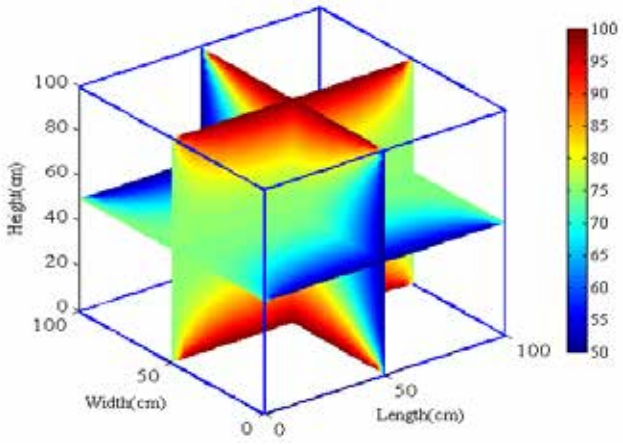


Fig. 4. Numerical solution profiles of *u* with $c_x = c_y = c_z = 10$ and $G_{rf} = 2$.

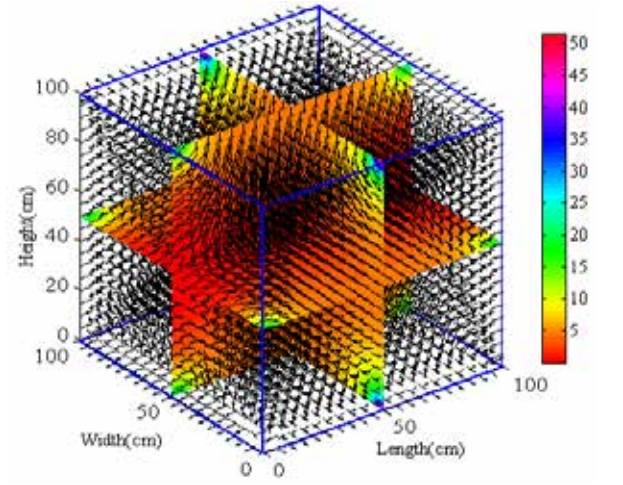


Fig. 5. Gradient profiles of *u* at each grid point with $c_x = c_y = c_z = 10$ and $G_{rf} = 2$.

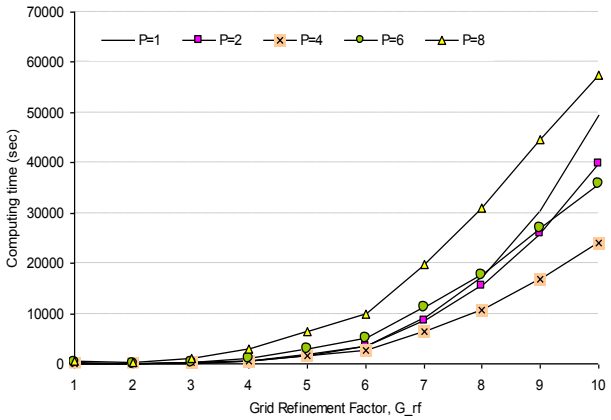


Fig.6.

ISSN – 1999-4974

Fig. 6. Sequential (at P=1 worker) and parallel (at P=2, 4, 6, 8) computing time (sec) versus grid refinement factor using implicit solution method.

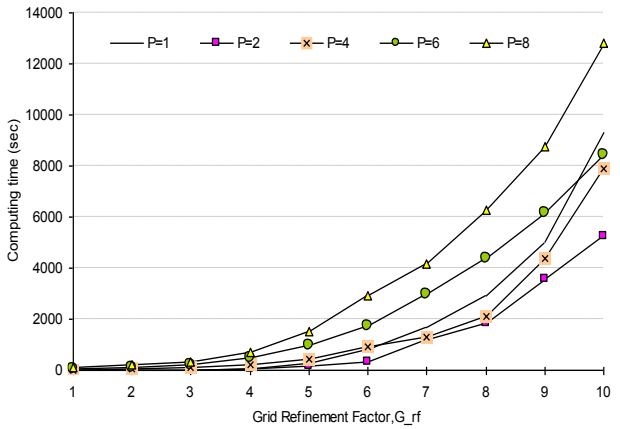


Fig.7.

Fig. 7. Sequential (at P=1 worker) and parallel (at P=2, 4, 6, 8) computing time (sec) versus grid refinement factor using explicit solution method.

In order to understand the message passing among the parallel workers the mpi profiles are obtained for both the methods. A schematic of the message passing profiles obtained for $G_{rf} = 1$ using parallel implicit and explicit methods on $P=8$ local workers are shown in Fig. 8. The amount of data exchanged among the workers is exhibited in Fig. 8 (a). Since in the implicit scheme much communication is required by mpi gather function thus the red pattern reveals the pair wise comparison of workers and maximum amount of data received per lab for this function. Other than red patterns illustrate the amount of data used during local distribution of matrix A, B and X to all workers. Similarly, in explicit scheme the each neighboring worker sends and receives a constant amount of data from its left and/or right worker simultaneously as shown in Fig. 8 (d). The non-blocking mpi send and receive functions are employed for this task. It is revealed that the amount of data exchanged in implicit scheme is higher than the explicit scheme. It is because of the reason that in every iteration N data points are gathered in implicit scheme while $(m+1)(s+1)$ data points are sent or received by neighboring workers in explicit scheme where $N > (m+1)(s+1)$. Moreover, the maximum communication time (sec) consumed by both the methods during data exchange process is illustrated by Fig. 8 (b and e). The maximum communication time in implicit method is about six times greater than explicit method. The pair wise communication patterns in explicit method are almost symmetric while the implicit method shows non-symmetric communication patterns. Finally, the maximum waiting time (sec) used during the message passing is also exhibited through Fig. 8 (c and f). Just like the maximum communication time the maximum waiting time in implicit method is higher than explicit method. Thus from the comparison of performance indicators it appears that for the parallel numerical solution of the problems similar to that is used in this study the explicit FDM provides better performance as compared to implicit FDM.

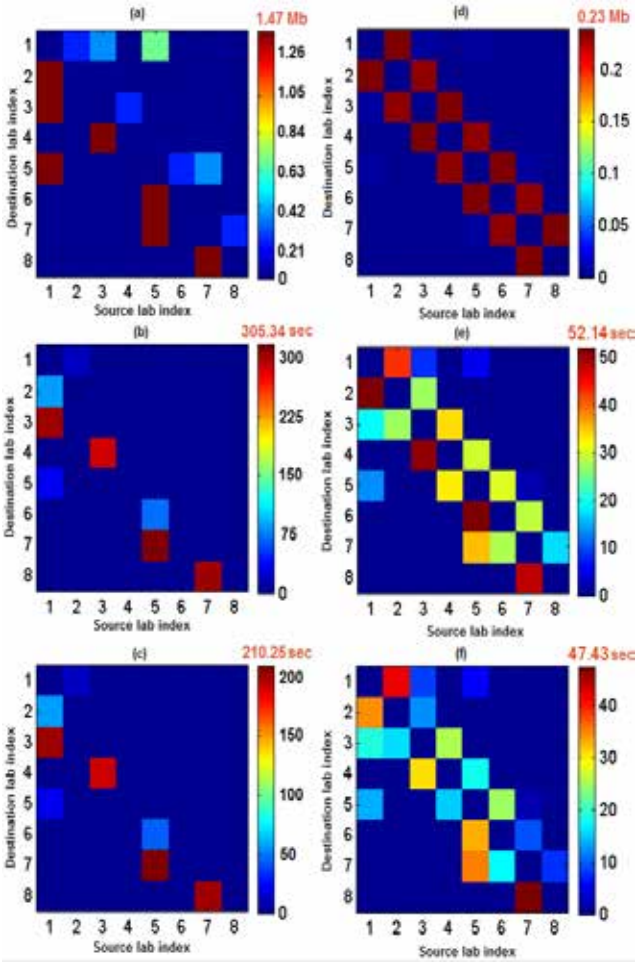


Fig.8.

Fig. 8. The message passing profiles obtained for $G_{rf}=1$ using parallel implicit and explicit schemes on P=8 local workers.: (a and d) the maximum data received per lab, (b and e) maximum communication time (sec) per lab, (c and f) maximum waiting time (sec) per lab

V. CONCLUSION

The parallel numerical solution of a typical 3D linear PDE was implemented by using the implicit and explicit finite difference solution methods. It was investigated that the implicit solution method uses significantly huge amount of memory than that of used by explicit solution method. Also the more data is required to pass among the parallel workers using implicit method. Consequently, the parallel performance of explicit method is much better than implicit method. The parallel system scales well up to P=4 workers on shared memory architecture. It is expected that the parallel performance may improve on distributed memory or GPU type architectures. The paper concludes that the explicit finite solution schemes are more efficient when implemented on parallel systems.

ISSN – 1999-4974

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support from Universiti Malaysia Sarawak and the Quaid-e-Awam University of Engineering Science and Technology, Nawabshah, Pakistan.

REFERENCES

[1] H. M. Anita. "Numerical Methods for Scientist and Engineers," Birkhauser-verlag, 2002.

[2] J. Stoer and R. Bulirsch, "Introduction to Numerical Analysis," Springer-Verlag, New York, 2002.

[3] M. .K. Jain, "Numerical Solution of Differential Equations," New Age International Ltd., New Delhi, 1984.

[4] G. D. Smith, "Numerical Solutions of Partial Differential Equations: Finite Difference Methods," Oxford University Press, New York, 1985.

[5] M. J. Hsieh and R. Luo ¨Exploring a coarse-grained distributive strategy for finite-difference Poisson–Boltzmann calculations," J Mol Model. 17(8), 2011, pp. 1985–1996.

[6] Y. Lu and C.A Shen, "Domain decomposition finite-difference method for parallel numerical implementation of time-dependent Maxwell's equations," Antennas and Propagation, IEEE Transactions on, 45, 1997, pp. 556-562.

[7] A. Fijany, D. Weinberger, R. Roosta, S. Gulati, "Massively Parallel Solution of Poisson Equation on Coarse Grain MIMD, "JPL TRS, 1998, pp. 1-21.

[8] S. H. Zainud Deen, E. Hassan, M. S. Ibrahim, K. H. Awadalla, and A. Z. Botros, "Electromagnetic scattering using GPU-based finite difference frequency domain method," Progress In Electromagnetics Research B, 16, 2009, pp. 351-369.

[9] Y. Wang, M. Baboulin, J. Dongarra, J. Falcou, Y. Fraigneau and O. L. Maitre, "A parallel solver for incompressible fluid flows," International Conference on Computational Science, ICCS 2013, pp.1-10.

[10] O. Ceylan and O. Kalenderli, "Parallel Computation of Two Dimensional Electric Field Distribution Using PETSC," Lecture Series on Computer and Computational Sciences, Volume 1, 2004, pp. 1-4.

[11] M. Vu Pham, F. Plourde and S.D Kim, "Strip Decomposition Parallelization of Fast Direct Poisson Solver on a 3D Cartesian Staggered Grid," International Journal of Mathematical, Computational, Physical and Quantum Engineering Vol.1 No.3, 2007, pp.183-192.

[12] A. M. Adams1, E. Sifakis and J. Teran, "A parallel multigrid Poisson solver for fluids simulation on large grids," Eurographics, ACM SIGGRAPH Symposium on Computer Animation (2010), pp.1-10.

[13] Q. Xu and W. Wang, "A new parallel iterative algorithm for solving 2D poisson equation," Vol. 27, Issue 4, 2011, pp.829–853.

[14] T. Tanga, W. Liua and J. M. McDonough, "Parallelization of linear iterative methods for solving the 3-D pressure Poisson equation using various programming languages," Procedia Engineering 61 (2013) 136 – 143.

[15] Bollig, Evan F., Natasha Flyer, and Gordon Erlebacher,"Solution to PDEs using radial basis function finite-differences (RBF-FD) on multiple GPUs," *Journal of Computational Physics,* vol. 231, pp. 7133-7151, 2012.

[16] Gong, Chunye, Weimin Bao, and Guojian Tang, "A parallel algorithm for the Riesz fractional reaction-diffusion equation with explicit finite difference method," *Fractional Calculus and Applied Analysis,* vol. 16, pp. 654-669, 2013.

[17] A. Margaris, S. Souravlas, and M. Roumeliotis, "Parallel implementations of the Jacobi linear algebraic system solver," In Proceedings of the 3rd Balkan Conference in Informatics, 2007, 161–172.

 ISSN – 1999-4974