PARALLEL IMPLEMENTATION OF ITERATIVE METHODS FOR SOLVING PDEs

Muhammad Naveed Akhtar, Muhammad Hanif Durad, Anila Usman
Department of Computer and Information Science (DCIS),
Pakistan Institute of Engineering & Applied Sciences (PIEAS)
Corresponding Author: naveed@pieas.edu.pk

ABSTRACT*: Partial differential equations (PDEs) can be used to model physical systems. Using direct methods to find solutions of PDEs are computationally expensive for large systems. However, Numerical solution of PDEs can be efficiently approximated by parallel computational techniques using iterative methods. In this paper a number of parallel iterative methods have been implemented and analyzed for Laplace and Poisson's equations in higher dimensions using message passing interface (MPI) library. Furthermore, two large scale linear systems have also been analyzed using said methods. Some suggestions regarding selection of appropriate parallel iterative methods have been made based on computational experiments derived from scale ability and timing analysis.*

Keywords: Partial differential equations (PDEs); Parallel Iterative method; Laplace's Equation; Poisson's equation; Message Passing Interface (MPI)

INTRODUCTION

Several large scale applications in field of scientific computing require simulating some physical phenomena whose behavior is governed by a set of partial differential equations (PDEs). These phenomenons are modeled by evaluating variables over grid covering the region of interest using discrete finite differences, resulting in large scale linear systems. While finding numerical solutions of partial differential equations the number of unknown variables may be 10^5 or more [1] and matrices involved are usually sparse with certain symmetry. Using direct methods to solve such problems is computationally expensive and hence iterative methods are utilized to solve both independent and PDE resultant linear systems.

Iterative methods are un-avoidable tools to find some approximation of any exact solution. Most of the Iterative methods have a starting value set called vector. Then there exists a sequences of such vectors computed on base of initial vector. This sequence finally converges to the exact solution. There may be a check to stop the computation if adequate precision is achieved. In many cases iterative methods performs faster than direct methods. Parallel implementation of these methods is also not very exaggerated. Iterative methods also demand some condition or mathematical properties to be fulfilled by linear system of equations for guaranteed convergence to exact solution.[2]

Iterative methods can be subdivided in two classes named stationary iterative methods (SIM) and non-stationary (also known as Krylov subspace methods) [3]. SIM methods evolve as a part of engineering and mathematics. These were popular in 1960 and now a day also used sometimes. These are not the best methods lately but are used as pre conditioners for Krylov subspace methods. In this paper three SIMs namely the Jacobi method, the Gauss–Seidel method and the successive over-relaxation method (SOR), along with their parallel implementation are being discussed which can overcome the sluggishness of these methods. Among many Krylov subspace methods we consider only one most important method, the Conjugate Gradient (CG) method for solving symmetric positive definite systems of equations which develop while modeling PDEs used in this paper. The parallel implementation of the CG method is based on the algorithm [2]. All algorithms discussed above have been implemented using Massage Passing Interface (MPI) library. With the advent of multi-core machines the performance of MPI programs is very difficult to predict. However, a statistical estimation may be used to foresee performance pattern of certain algorithms.

The rest of this paper is structured as follows: Section II summarizes related work; section III describes the target architectures and execution environment used for performance evaluation; section IV reviews the basic concepts behind the individual iterative methods. Finally Section V presents experimental results for selected PDEs and large scale linear systems, then for the two large scale systems. Section V concludes the paper.

RELATED WORK

Parallel implementations of iterative methods have been studied extensively in the last three decades. Almost all textbooks on parallel computing discuss the iterative methods in detail. The classical references include [1-8]. Each of these texts usually presents a selected number of iterative methods, thus getting a unified picture about the performance of these algorithms is a bit difficult. A few other researchers have also discussed these topics, but they have highlighted different aspects of iterative methods such as:

Ortega, James M *et al.* [9] presented the status of numerical methods for partial differential equations on vector and parallel computers. They discuss both direct and iterative methods for elliptic equations for various available architectures at the time of writing. They have presented some experimental results.

Mathew, Tarek Poonithara Abraham [10] have studied all iterative algorithms for the solution of partial differential equations, techniques for the discretization of partial differential equations on non-matching grids, and techniques for the heterogeneous approximation of partial differential equations of heterogeneous character. The divide and conquer methodology used is based on a decomposition of the domain of the partial differential equation into smaller sub-domains, and by design is suited for implementation on parallel computer architectures. However, even on serial computers, these methods can provide flexibility in the treatment of complex geometry and heterogeneity in a partial differential equation.

Balay, Satish et al. presents an overview of background for the numerical solution of PDEs, explains the challenges in parallel computations for PDE-based models. They have recommended use of PETSc software library and provided a brief overview of PDE-related software in high-performance computing community. They have additional recommendations for application scientists regarding software choices.

Korfgen, Bernd and Inge Gutheil [12] solve two problems of PDEs in parallel computer architecture using numerical methods. These problems include Poisson's equation in 2-D and the physical process of vibration of a membrane. Simple Jacobi algorithm is used to solve the Poisson equation, also a suitable parallelization scheme is proposed. 2nd case uses the ScaLAPACK library for calculation and the issue regarding data distribution is addressed.

In summary, the existing papers or texts have following limitations:

- It is difficult to get a unified picture of the performance of various parallel iterative algorithms using MPI.
- There are only a few iterative algorithms have been implemented using MPI.
- Analysis of comparatively modest size vectors and matrices is present in literature.

In the nutshell, we believe that the paper will provide some extensions in theoretical background and practical implementation of parallel iterative algorithms.

TARGET ARCHITECTURES AND EXECUTION ENVIORNMENT

The two machines, a standalone machine and other computing cluster were used in experiments having the following architectures:

SGI Virtue: 2 x Intel Xeon Processor E5440 @ 2.83 GHz with 12 MB cache, 4 cores and 4 GB memory.

Computing Cluster:

Head Node: 2 x Intel Xeon Processors E5504 @ 2.00 GHz with 4 MB cache, 4 cores and 16 GB memory,

Cluster-Workers: 6 x Intel Core i5 Processors @ 2.67GHz with 8MB Cache, 4 cores and 4 GB memory each.

These algorithms were executed repeatedly on above systems using MPI. SGI VIRTUE system has 8 processing cores on a single board, while the computing cluster system has 40 processing cores out of which 32 utilized in the computational experiments carried out in this paper. These processing elements communicate with each other over Gigabit Ethernet network.

REVIEWOF PARALLEL ITERATIVE METHODS FOR PDES

As stated earlier that iterative methods are classified as SIM (stationary iterative methods) and Krylov subspace methods. Jacobi method, the successive over-relaxation method (SOR), the Gauss–Seidel method and symmetric successive over-relaxation method (SSOR) come under the umbrella of stationary iterative methods. Whereas Non-stationary Methods include Conjugate Gradient (CG), Generalized Minimal Residual (GMRES), Minimum Residual (MINRES), Quasi-Minimal Residual (QMR), Bi-Conjugate Gradient (BiCG) and Bi-conjugate Gradient Stabilized (Bi-CGSTAB) etc [3]. Parallel implementation of three stationary iterative methods including Jacobi method, the Gauss–Seidel method, and the successive over-relaxation method (SOR) and one non-stationary iterative method the Conjugate Gradient (CG) is discussed in this paper.

It should be noted that the basic model implemented to solve all these PDEs is the solution of the system of linear equations represented as:

$$Ax = b \tag{1}$$

Assume $\overset{(k)}{x}$ is an approximation to the solution x of equation (1), Then

$$x = \overset{(k)}{x} + \overset{(k)}{e} \tag{2}$$

Here $\overset{(k)}{e}$ is called the error. Our objective in each iteration is to reduce the error according to some set criterion.

Jacobi Method

Jacobi method is also known as the simultaneous displacement method as it treats each equation independently for finding the values. It can be represented in the matrix form as:

$$\overset{(k)}{x} = \mathrm{D}^{-1}(L+U)\overset{(k-1)}{x} + \mathrm{D}^{-1}\mathrm{b} \tag{3}$$

Where the matrices D, L and U represent the diagonal, the strictly lower-triangular, and the strictly upper-triangular parts of A, respectively.[13]

For implementing Jacobi method in MPI environment it takes the input from a file and allocates the memory using memory allocation functions, initializes processes and sends data to them. Processes then compute the given set of data and perform their computations independently. After they have finished first iteration, they again broadcast their result so that each process gets the updated values. When any of the stopping criteria is met, iterations stop and the result are displayed.

Gauss Seidel Method

Gauss- Seidel method is obtained by a little modification in the Jacobi method. The difference is that the equations are examined at a time in sequence, and the obtained results are used in next steps as soon as they become available. This method is also called as a method of successive displacements. In matrix form this method can be represented as:

$$\overset{(k)}{x} = (\mathrm{D}-L)^{-1}(U\overset{(k-1)}{x} + \mathrm{b}) \tag{4}$$

In above relation D, L and U are matrices representing diagonal, the strictly lower-triangular, and the strictly upper-triangular parts of system matrix A, respectively [13].

For implementing Gauss Seidel Method in MPI environment it takes the input from a file and allocates the memory using memory allocation functions, initializes processes and sends data to them. Processes then compute the given set of data and perform their computations independently. After they have finished first iteration, they again broadcast their result so that each process gets the updated values. When any of the stopping criteria is met, iterations stop and the result are displayed. In this method, processes depend for their computation on the data from the other processes. Therefore, there is a lot of data dependency. Due to this reason the parallel execution time is more than the serial execution time.

Successive Over-Relaxation Method (SOR)
Gauss-Seidel converges more rapidly by using successive over-relaxation (SOR). It uses step to next Gauss-Seidel iteration as search direction with a fixed search parameter called w.
In matrix form Gauss Seidel method can be represented as:

$$\overset{(k)}{x} = (1-w)\overset{(k-1)}{x} + w\overset{k}{x}_{GS} \quad (5)$$

Here the $\overset{k}{x}_{GS}$ represents Gauss Seidel approximation. Value of w determines the convergence rate. $w > 1$ means over-relaxation; $w < 1$ shows under-relaxation and $w = 1$ means Gauss-Seidel method [13].

MPI implementation of SOR method is the same as Gauss-Seidel method except that relaxation parameter is pre-calculated for specific system.

Conjugate Gradient (CG) Method
The Conjugate Gradient method performs better for symmetric positive definite systems. This method generate vectors with successive approximations leading to the solution, residuals corresponding to the each iteration performed and the search directions which are used as base of further iterations and residuals computation [13].

Updated scalars are computed in each iteration, by using two inner products. These scalars are used to make the sequences satisfy certain orthogonal conditions.

The parallel implementation of the CG method is based on the algorithm [2] each iteration step of this algorithm is based on the basic vector and matrix operations but in an efficient way.

EXPERIMENTAL RESULTS
A unified performance analysis for said parallel iterative methods has been performed for two PDEs namely Laplace's equation, Poisson's equations; and two large scale systems using target architectures. The results are presented in this section. Similar trends were observed for both hardware systems; however, we present results for our computing cluster.

Laplace 2D
The following equation has been taken from [14]. Consider the problem of determining the steady-state heat distribution in a thin square metal plate using Laplace's equation model is as:

$$u_{xx} + u_{yy} = 0$$
$$0 \le x, y \le 1 \quad (6)$$

The boundary conditions are:

$$u(0, y) = u(x, 0) = 0$$
$$u(1, y) = 400y;\ u(x,1) = 400x$$

And the exact solution is:

$$u(x, y) = 400xy$$

Fig. 1shows the execution time for said iterative methods using eight processing elements.

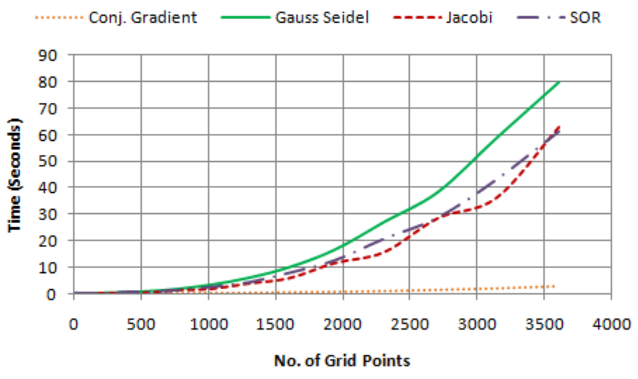

Fig. 1Execution time for iterative methods

Fig. 1 depicts that the conjugate gradient is the fastest and Gauss–Seidel method is the slowest. The results are excepted as conjugate gradient uses proper search direction while Gauss–Seidel method involves heavy communication between processing elements. The Jacobi method is second in speed since it involves much less communication between processing elements from present iteration to next iteration.
Fig. 2 shows no. of processing elements versus execution time keeping the no. of grid points constant = 3600.

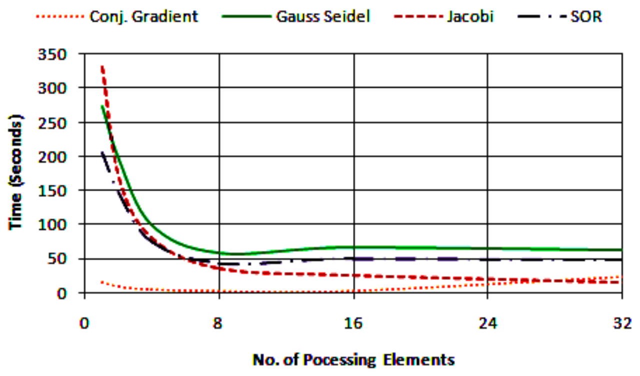

Fig. 2No. of processing elements versus execution time

Fig. 2 shows execution time decreases with increasing processing elements until processor count reaches to eight. After that execution time starts increasing gradually because more network communication is involved.

Laplace 3D
The following equation has been extended to 3-D by the authors. Consider the problem of determining the steady-state heat distribution in a thin cube metal plate using Laplace's equation model is as:

$$u_{xx} + u_{yy} + u_{zz} = 0$$
$$0 \le x, y, z \le 1 \quad (7)$$

The boundary conditions are:

$$u(0,y,z)=u(x,0,z)=u(x,y,0)=0$$
$$u(1,y,z)=800yz$$
$$u(x,1,z)=800xz$$
$$u(x,y,1)=800xy$$

And the exact solution is:

$$u(x,y,z)=800xyz$$

Fig. 3 shows the execution time for said iterative methods using eight processing elements.

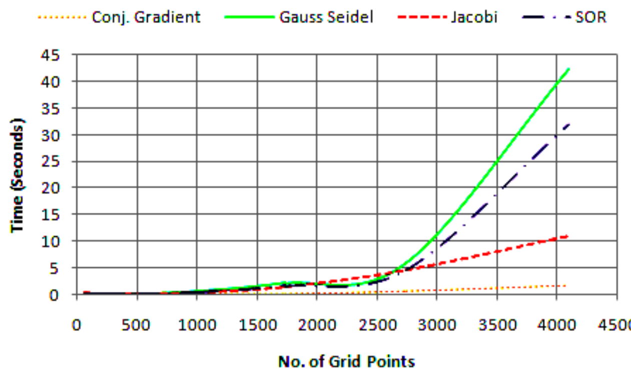

Fig. 3 Execution time for iterative methods

Fig. 3for 3D Laplace depicts almost same trends as shown in Fig. 1.

Fig. 4 shows execution time versus no. of processing elements keeping the no. of grid points constant=3600.

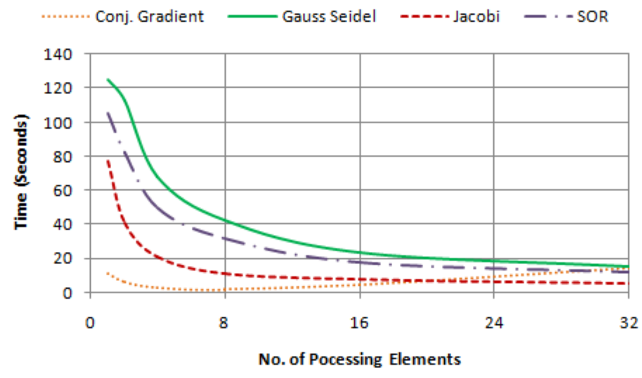

Fig. 4 No. of processing elements versus execution time

It can be seen from Fig. 4, the trend is similar to Fig. 2 though problem dimension has increased.

Poisson 2D

The following equation has been taken from [14]. Consider the problem of determining the steady-state heat distribution in a thin square metal plate using Poisson's equation model is as:

$$u_{xx}+u_{yy}=xe^{y} \qquad (8)$$
$$0\le x,y\le 1$$

The boundary conditions are:

$$u(0,y)=0 \quad , \quad u(1,y)=e^{y}$$
$$u(x,0)=x \quad , \quad u(x,1)=e^{x}$$

And the exact solution is:

$$u(x,y)=xe^{y}$$

Fig. 5shows the execution time for said iterative methods using eight processing elements.

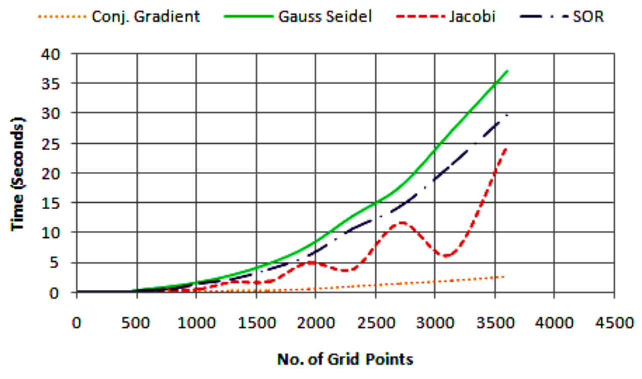

Fig. 5 Execution time for iterative methods

Fig. 5 depicts the same trend as for Fig. 1 showing no difference between the results of Poisson and Laplace equations.

Fig. 6shows no. of processing elements versus execution time keeping the no. of grid points constant = 3600.

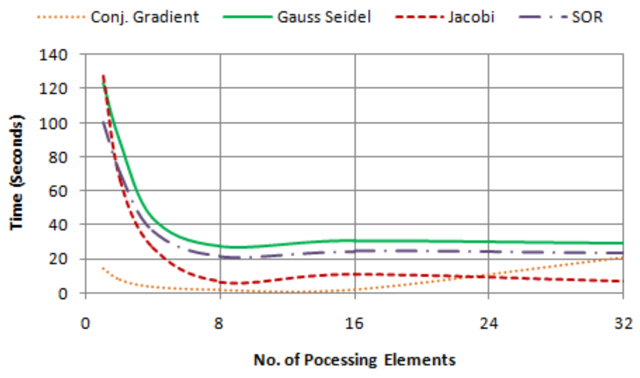

Fig. 6 No. of processing elements versus execution time

Fig. 6 shows the similar trend as Fig. 2.

Poisson 3D

The following equation has been extended to 3-D by the authors. Consider the problem of determining the steady-state heat distribution in a thin cube metal plate using Poisson's equation model is as:

$$u_{xx}+u_{yy}+u_{zz}=3e^{x}e^{y}e^{z} \qquad (9)$$
$$0\le x,y,z\le 1$$

The boundary conditions are:

$$u(0,y,z)=e^{y}e^{z} \quad , \quad u(1,y,z)=e.e^{y}e^{z}$$
$$u(x,0,z)=e^{x}e^{z} \quad , \quad u(x,1,z)=e.e^{x}e^{z}$$
$$u(x,y,0)=e^{x}e^{y} \quad , \quad u(x,y,1)=e.e^{x}e^{y}$$

And the analytical solution is:

$$u(x,y,z)=e^x e^y e^z$$

Fig. 7 shows the execution time for said iterative methods using eight processing elements.

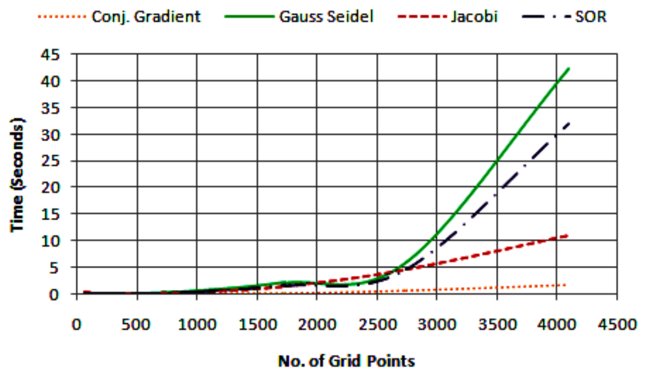

Fig. 7 Execution time for iterative methods

Fig. 7 depicts the same trend as for Fig. 1 showing no change in overall trend with increase in problem dimension.

Fig. 8shows execution time versus no. of processing elements keeping the no. of grid points constant.

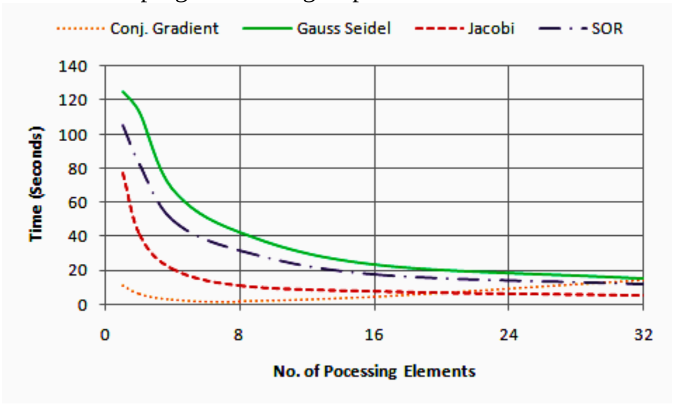

Fig. 8 No. of processing elements versus execution time

It can be seen from Fig. 8, the trend is similar to Fig. 2 though problem dimension has increased.

Large Scale System I

This system was derived by authors to compare the results; we need to have a standard problem set through which we can verify the results effectively and efficiently. For this reason following matrix was chosen due to its high degree of stability and convergence.

Let n be an even integer and consider the $n \times n$ matrix A with '3' on main diagonal, '-1' on the super and sub diagonals, and '1/2' in the $(i, n+1-i)$ position for all i = 1 to n except for i = (n/2) and (n/2) +1. The vector b is defined as b= [2.5, 1.5…1.5, 1.0, 1.0, 1.5 …1.5, 2.5]. In this vector, there are n - 4repetitions of 1.5 and 2 repetitions of 1.0.The exact solution of this system is a vector x containing all ones.

All iterative methods are tested by using suggested large scale system. The results are shown below in Fig. 9. This figure shows the execution time for said iterative methods using eight processing elements.

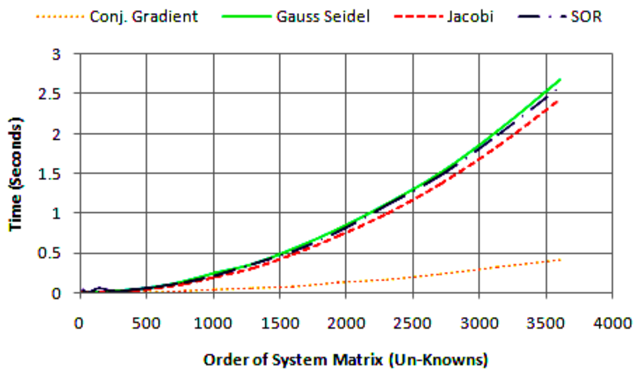

Fig. 9Execution time for iterative methods

Fig. 9 depicts the same trend as for Fig. 1 showing no change in overall trend with increase in problem dimension.

Fig. 10 shows execution time versus no. of processing elements keeping the no. of grid points constant.

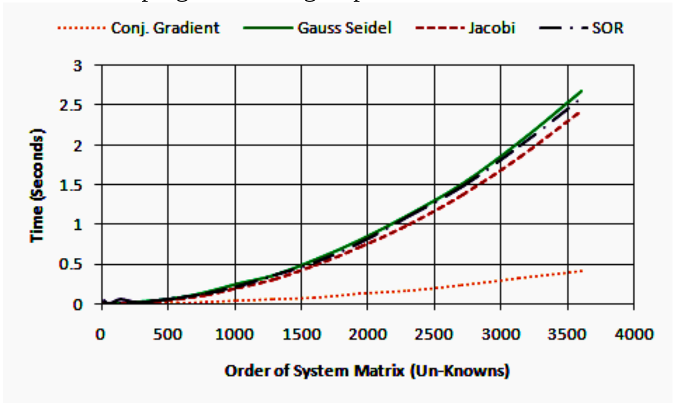

Fig. 10 No. of processing elements versus execution time

It can be seen from Fig. 10, the trend is similar to Fig. 2 though problem dimension has increased

Large Scale System II

This problem has been taken from Matrix Market [15]. This matrix is just used a linear system as given in equation (1): Its solution was also varied verified using MATLAB®. The results are shown as below.

Fig. 11 shows the execution time for said iterative methods using eight processing elements.

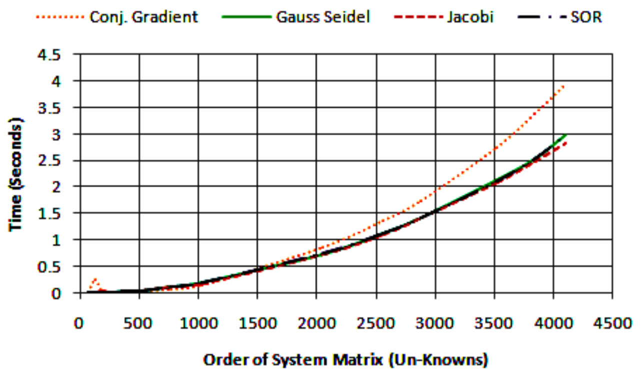

Fig. 11 Execution time for iterative methods

It can be observed Conjugate Gradient performed worse because matrix involved in this system is not positive definite. The matrix *'A'* didn't have symmetry and could not avoid a fill-in of the matrix with non-zero elements [2] , increasing communication overhead. Thus in this case Conjugate Gradient performed worse.

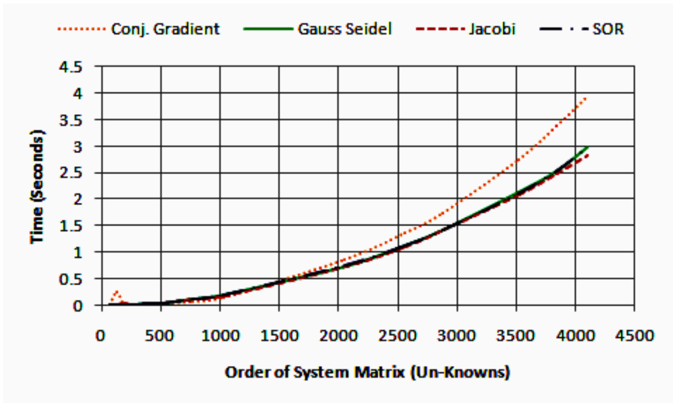

Fig. 12 Execution time for iterative methods

Fig. 12 shows execution time versus no. of processing elements keeping the no. of grid points constant=4096.
The results are similar to Fig. 12 as given below with same reasoning as suggested:

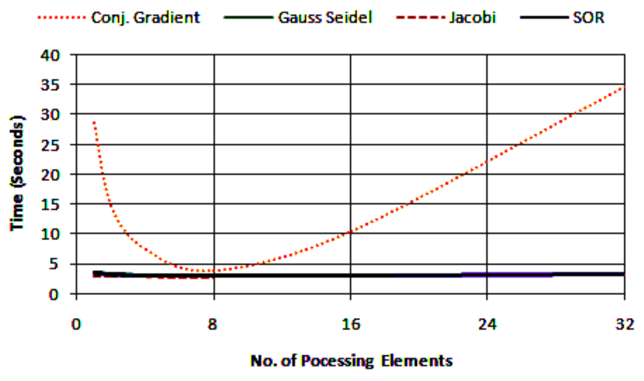

Fig. 13 No. of processing elements versus execution time

SUMMARY AND FUTURE WORK

In this paper we have analyzed most of the parallel iterative algorithms using MPI for two different architectures. It is very difficult to find a single paper implementing all these iterative algorithms methods in altogether. Our findings from computational experiments performed in this regard are listed as:

1. There is a general decrease in execution time with increase in no of processing elements, and a general increase in execution time with increase in data size is observed if there is no network communication involved for all the iterative methods.
2. In most of the cases Conjugate Gradient method could be best choice for parallel PDE solution.
3. Performance of Gauss-Seidel, Jacobi and SOR is significantly affected over the network.
4. Conjugate Gradient could perform worse if the system matrix is not positive definite.

We are planning to extend this work in future by:

1. Implementing Red Black Gauss-Seidel and Multi-grid methods.
2. Implementing the generated matrices in compressed format to reduce communication time.

In short, we consider that this paper has contributed to a few extensions in practical implementation of parallel iterative algorithms.

REFERENCES

[1] Shonkwiler, Ronald W., and Lew Lefton. An Introduction to Parallel and Vector Scientific Computation. Vol. 41. Cambridge University Press, 2006.

[2] Rauber, Thomas, and Gudula Rünger. Parallel programming: For multicore and cluster systems. Springer Science & Business, 2013.

[3] O'Leary, Dianne P. Scientific computing with case studies. SIAM, 2009.

[4] Foster I., Designing and Building Parallel Programs, Addison-Wesley, 1995.

[5] Petersen, Wesley P., and Peter Arbenz. Introduction to parallel computing. No. 9. Oxford University Press, 2004..

[6] Wilkinson B., Michael A., Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers , 2nd ed., Prentice Hall, 2003.

[7] Bisseling, Rob H. Parallel scientific computation. Oxford University Press, 2004.

[8] Quinn, M. J., Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2004.

[9] Ortega, James M., and Robert G. Voigt. Solution of partial differential equations on vector and parallel computers. Siam, 1985.

[10] Mathew, Tarek Poonithara Abraham. Domain decomposition methods for the numerical solution of partial differential equations. Vol. 61. Berlin: Springer, 2008.

[11] Balay, Satish, et al. "Software for the scalable solution of partial differential equations." Sourcebook of parallel computing. Morgan Kaufmann Publishers Inc., 2003.

[12] Korfgen, Bernd, and Inge Gutheil. "Parallel Linear Algebra Methods." Computational Nanoscience: Do it Yourself 31.1: 507, 2006.

[13] Heath, Michael T. Scientific computing. McGraw-Hill, 2001.

[14] Burden, Richard L., and J. Douglas Faires. Numerical analysis. Thomson Brooks/Cole, 2005.

[15] http://math.nist.gov/MatrixMarket/data/NEP/matpde/matpde.html