EFFICIENT SOLVER FOR LINEAR ALGEBRAIC EQUATIONS ON PARALLEL ARCHITECTURE USING MPI

[1]Akshay N. Panajwar, [2]Prof.M.A.Shah
Department of Computer Science and Engineering, Walchand College of Engineering, Sangli, India
Email:[1]akshay.panajwar@gmail.com, [2]medha.shah@walchandsangli.ac.in

Abstract
Most of the problems in engineering science can be modelled using differential equations. However quite often, the differential equations are not amenable to direct analytical solutions because of complex geometries and boundary conditions. Thus recourse is taken to approximate numerical solution techniques. Depending on the discretization of the domain, the numerical model of the problem at hand could become very large – sometimes running into thousands/lakhs of degrees of freedom. Solution of such a large set of equations is a daunting task computationally and hence efforts are directed towards development of efficient strategies and use of High Performance Computing (HPC). The present work aims to implement an efficient solver for system of linear equations with sparse, symmetric and positive-definite matrix of coefficients using the most prominent Conjugate Gradient iterative method and study the performance improvement obtained in parallel computing framework using MPI and GPU enabled CUDA technologies considering a suitable example problem.

Index Terms: **Message Passing Interface (MPI), Compute Unified Device Architecture(CUDA), Finite Element Method (FEM), Conjugate Gradient (CG).**

I. Introduction

In the past few years, The Message Passing Interface(MPI) and GPU(Graphics Processing Unit) enabled CUDA(Compute Unified Device Architecture) is being widely used to develop parallel programs on computing systems such as clusters. Recent advances in high performance computing resources, programming environments for parallel architecture are well suited to perform massively parallel computation on the large set of equations. In this paper, we focus on solving large set of linear algebraic equations of the form

$$Ax=B \qquad (1)$$

Where the known n-by-n matrix 'A' is sparse, symmetric (i.e. A^T=A), positive definite (i.e.x^TAx > 0 for all non-zero vectors $x \in R^n$), and real, and 'B' is known right hand side vector.

The equation arises in various fields of engineering science such as complex structural analysis problems in aerospace, civil and mechanical engineering disciplines; complex fluid flows for example in aerospace, ocean modeling for tsunami applications, blood flow etc. To render them suitable for computer based solution, approximate numerical techniques are often deployed. Efficient solution of such a large set of algebraic equation enables us to build models that are as close to reality as possible, thus enhancing our understanding of scientific problems. Computer based modeling and simulation is recognized as the third pillar/paradigm of scientific development. These linear algebraic equations can be solved by direct method (Gauss Elimination, LU decomposition etc.) or by iterative method (Steepest Descent, Conjugate Gradient)[1].The Non-stationary iterative method (Conjugate Gradient, preconditioned Conjugate Gradient) are more accurate than Stationary methods (Jacobi, Gauss-Seidel)[2]. These methods are widely used to solve many important settings such as finite differences, finite element method [3] for solving partial differential equations and structural and circuit analysis.

The outline of the paper is as follows. Paper includes a summary of prior works which includes iterative methods (CG) to solve system of linear equations, proposed work and finally conclusion.

II. Literature Survey

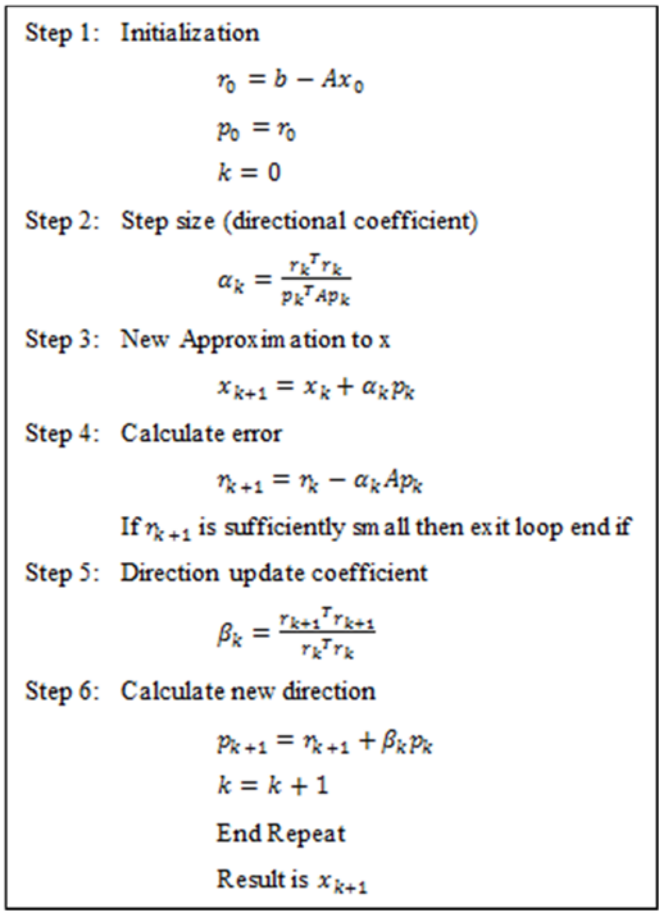
2.1 Conjugate Gradient Method:

The Conjugate Gradient is the most widely used iterative method to solve system of linear equations of the form

$$Ax=B$$

Where, 'A' is symmetric and positive definite matrix, 'x' is unknown vector and 'B' is the known vector. The iterative method like CG are suited for use with sparse matrix and if matrix is dense then direct method using substitution are efficient then iterative methods. The CG will take at most 'n' iterations to reach to the exact solution. The algorithm for CG is shown in fig.1.

let, initially assume unknown vector $x_0 = 0$, where each direction vector 'p_k' gives new approximate 'x_{k+1}' and also error 'r_k'. In each step, using this error, the previous direction and the property that any two directions are conjugate to each other we find a new direction. We repeat this process till the error 'r_k' is sufficiently small.



```
Step 1: Initialization
            $r_0 = b - Ax_0$
            $p_0 = r_0$
            $k = 0$
Step 2: Step size (directional coefficient)
            $\alpha_k = \frac{r_k{}^T r_k}{p_k{}^T A p_k}$
Step 3: New Approximation to x
            $x_{k+1} = x_k + \alpha_k p_k$
Step 4: Calculate error
            $r_{k+1} = r_k - \alpha_k A p_k$
        If $r_{k+1}$ is sufficiently small then exit loop end if
Step 5: Direction update coefficient
            $\beta_k = \frac{r_{k+1}{}^T r_{k+1}}{r_k{}^T r_k}$
Step 6: Calculate new direction
            $p_{k+1} = r_{k+1} + \beta_k p_k$
            $k = k + 1$
            End Repeat
            Result is $x_{k+1}$
```

Fig. 1 Conjugate Gradient Algorithm.

The CG method requires one SpMV and two inner products. The most time-consuming part of this algorithm is multiplication of sparse matrix 'A' and dense vector 'p', hence efforts are directed towards parallel computation and also it is more efficient to store only the non-zero elements of a sparse matrix. There is a number of common storage formats used for sparse matrices discussed in [4].

III. Message Passing Interface

MPI[6] is language independent communication protocol used to program on parallel computers to scale the performance of the application. Message passing is an activity where the processors coordinate their activities by explicitly sending and receiving messages by the functions MPI_Send and MPI_Recv. It is the most common method of programming distributed-memory MIMD system. MPI is considered today's standard in message passing library.

IV. Sparse Matrix Storage Format

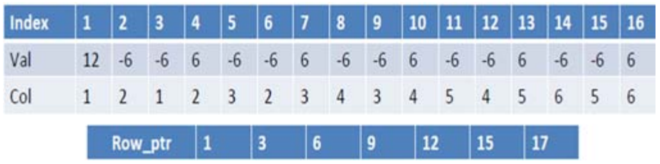
The matrix that arises from the discretization of the differential equation has a large dimension, usually more than thousands by thousands in size and it is also sparse, and so a good sparse matrix format doesn't only reduce the space requirement, but also enables faster sparse matrix operations. Moreover, since most of the simulation time is spent on solving sparse linear system equations, choosing a good sparse matrix format is essential to facilitate faster sparse matrix solvers.

$$\begin{bmatrix} 12 & -6 & 0 & 0 & 0 & 0 \\ -6 & 6 & -6 & 0 & 0 & 0 \\ 0 & -6 & 6 & -6 & 0 & 0 \\ 0 & 0 & -6 & 6 & -6 & 0 \\ 0 & 0 & 0 & -6 & 6 & -6 \\ 0 & 0 & 0 & 0 & -6 & 6 \end{bmatrix}$$

Fig. 2 Sparse Matrix.

Compressed Sparse Row (CSR) is popular and the most general purpose storage format. This can be used for storing matrices with arbitrary sparsity patterns as it makes no assumptions about the structure of the nonzero elements. Like COO, this format also stores only nonzero elements. These elements are stored using three vectors: 'val', 'col' and 'row_ptr'. The 'val' and 'col' vectors are same as for the COO format. For an 'M*N' sparse matrix, the 'row_ptr' vector has length of 'M+1' and stores indexes where each row of the matrix starts in the 'val' vector. The last entry of 'row_ptr' corresponds to the number of nonzero elements in the matrix. There are some advantages of using CSR over COO. The CSR format takes less storage compared to COO due to the compression of the row indices explained in above Figure. Also, with the 'row_ptr' vector we can easily compute the number of nonzero elements in the i^{th} row as 'row_ptr'(i+1) − 'row_ptr'(i).

In parallel algorithms, 'row_ptr' values allow fast row slicing operations and fast access to matrix elements using pointer in direction. This is a most commonly used sparse matrix storage scheme. This is explained below:



Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Val	12	-6	-6	6	-6	-6	6	-6	-6	6	-6	-6	6	-6	-6	6
Col	1	2	1	2	3	2	3	4	3	4	5	4	5	6	5	6

Row_ptr	1	3	6	9	12	15	17

V. Proposed Work

The present work aims to implement an efficient solver for linear algebraic equations and study the performance improvement obtained in a parallel computing framework using a suitable example of 1D Bar problem using FEM (Finite Element Method).

Solving FEM problem divided into following steps
1. Divide problem into number of finite elements.
2. Generate local stiffness matrix.
3. Assemble to form the linear systems of equations of the form Ku=f
4. Solve this linear system of equations by CG method

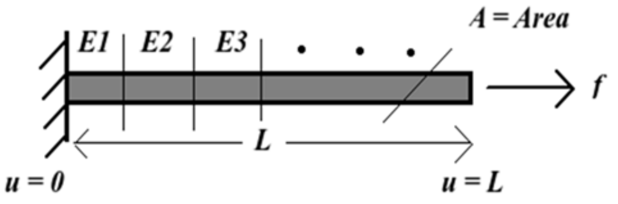


Fig. 3 1D Bar Problem

Solutions to static linear structural mechanics problems reduce to solving a linear system of equations of the type:

$$[K]\{U\} = \{F\}$$

Where 'K' is global stiffness matrix, 'U' is a global displacement vector, and 'F' is the global force vector. One end of bar is fixed and force (f) is applied at other end of bar. Bar is divided into 'n' number of finite elements area of cross sections area (A) at each element. Hence the matrix 'K' which will be generated will be of size n.Equation 'Ku=f' is then solved by CG/PCG to calculate displacement vector 'u'.

The memory requirements and the computational time required to solve such equations increases as the number of equations increases. To deal with such large numerical problems in the Finite Element Analysis, parallel computing on high performance computer is gradually becoming a main stream tool. Many parallel algorithms and programs for finite element computation have been developed on parallel computers, utilizing vast numbers of CPUs or GPU cores to achieve high speed up and scalability.

VI. Parallel Implementation of CG Algorithm

Since CG, is an iterative method, the result of next step is depend upon its previous step, hence it is hard to simultaneously compute intermediate steps, so we focused on distributing the data across processor and ask them to involve in the computation of its local part. As CG algorithm involves two dot products and one matrix vector multiplication per iteration and it can be easily parallelized by using following MPI communication operations:

Step1: Read the Matrix stored in CSR format and Right hand side Vector from the file and divide it across processors using MPI_Bcast and MPI_Scatterv.

Step 2: All processors will compute the dot product locally and do a sum reduction using MPI_Allreduce.

Step 3: In Matrix Vector multiplication all processors requires complete vector and this vector is getting updated after each iterations. So do multiplication in parallel using MPI_Allgatherv, to gather all the local parts of the vector into a single vector and then do the multiplication.

Also, one can used MPI_Send and MPI_Recv to gather vector but MPI_Allgatherv is known to be an efficient way to communicate between processes.[6]

But, as we are dealing with only 1D bar problem which generates tridiagonal matrix, in this case MPI_Allgatherv would not be efficient because every processor requires only one or two elements of vector $P$to carry out matrix vector multiplication, so we used MPI_Isend and MPI_Irecv to avoid communication overhead.

VII. Results

As the most time consuming part of conjugate gradient method is SpMV, so to optimize Sparse matrix vector multiplication we have used CSR sparse matrix storage format which allow us to perform operations only on the nonzero elements present in the matrix. Figure (4) shows the comparison of sequential and parallel implementation using MPI of Conjugate Gradient method for different sizes of tridiagonal matrix.

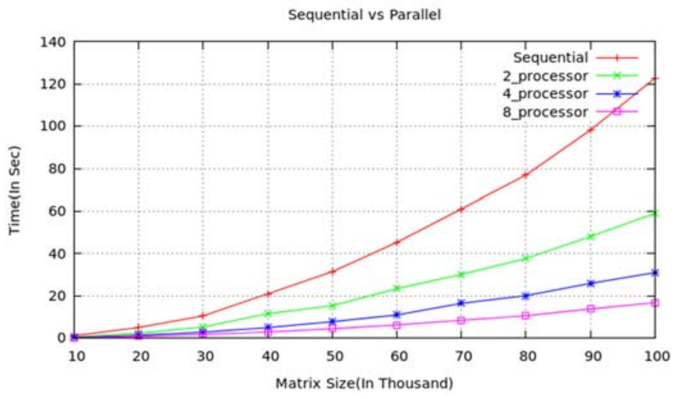


Fig. 4 Sequential vs Parallel

Figure(5) shows the scalability of parallel code executed on 2 node cluster which has total 4 cpu's and 8 cores's/cpu with matrix size N=504000 having non zero elements NNZ = 1511997 and the speedup of parallel conjugate gradient method is shown in Figure(6).

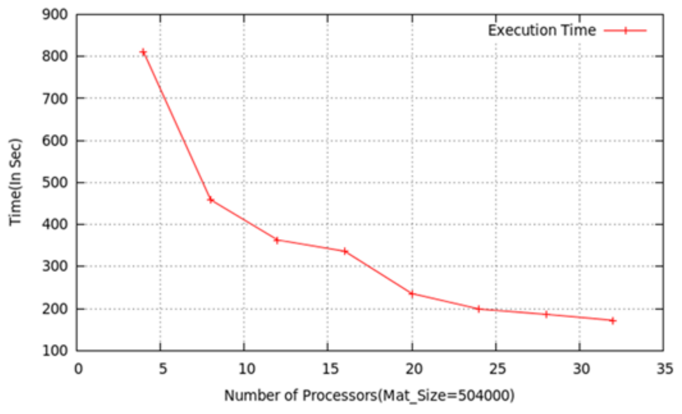


Fig. 5 Scalability of Parallel CG

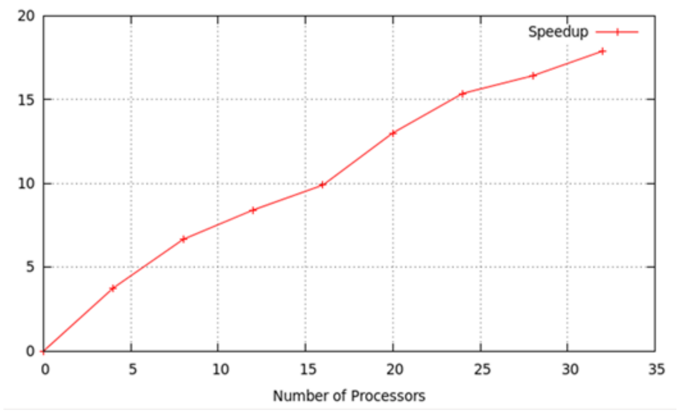


Fig. 6 Speedup

VIII. Conclusions

In this project we have implemented a parallel version of the Conjugate Gradient method using Message Passing Interface and have tested it for various sizes of tridiagonal matrix. Although we could have used various libraries such as SCALAPACK, PETSC for the implementation of CG, but instead we choose to implement it by our self so that we can be familiar with the issues while developing parallel CG. So, according to us there may be some communication overhead either in MPI_Allgatherv or MPI_Isend and MPI_Irecv whatever is used according to input matrix. So efficient communication is must while dealing with such huge sizes of matrices while solving linear algebraic equations.

References

[1] Jonathan Richard Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", August 1994

[2] D. A. Gismalla, "Matlab Software for Iterative Methods and Algorithms to Solve a Linear System”, International Journal of Engineering and Technical Research (IJETR)2014 ISSN: 2321-0869

[3] P. Seshu, Textbook on Finite Element Analysis, Prentice Hall of India, 12th reprint, 2014.

[4] P. Kumbhar, "Performance of PETSc GPU Implementation with Sparse Matrix Storage Schemes", EPCC 2011.

[5] Huaguang Song, "Preconditioning Techniques Analysis for CG Method".

[6] “Message Passing Interface Forum,” http://www.mpi-forum.org,2012.