LITERATURE REVIEW: A Parallel Jacobi-Embedded Gauss-Seidel Method

Nirav C. Pansuriya
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
niravchhaganbhaipan@cmail.carleton.ca

15 October 2021

1 Introduction

In past years, computer architecture has become increasingly parallel, with the use of GPUs and multi-core CPUs. We have massive parallelism available in modern GPU like NVIDIA GForce GTX 280. So, the focus of researchers moved on parallelism rather than continuous improvements of the single unit speed of computation. In physics, mathematics and engineering, we frequently encountered many linear systems. In many scientific simulations, we have to solve large-scale linear equations. But in general, large-scale linear equations take high time and resources like memory. We always want an algorithm that can solve large-scale linear equations in less time and resources. As during the last 2 decades, we have witnessed too many improvements in computer hardware, now we can see the expansion of parallelization in many domains. Many researchers have come up with parallel methods to solve linear systems. Most of these methods only work with sparse matrix linear systems. To overcome this problem, two researchers, Afshin Ahmadi and Amin Khademi - in a recent paper named "A Parallel Jacobi-Embedded Gauss-Seidel Method", came up with a new method, that can work in parallel architecture and also can work with any type of matrix linear system. I aim to implement this method in multicore architecture and compare its performance with some other parallel methods used for solving linear equations systems. My other aim is to extend their work by trying to modify the proposed method to make it faster. I would like to implement the proposed method in distributed memory architecture by making appropriate changes in this method.

2 Literature Review

Large linear equation systems are used in many complex scientific simulators. Solving such large systems can consume a large number of computational resources like memory, CPU etc and it is a very time-consuming process. That is why it is always a research area to develop efficient algorithms to solve such large systems in less amount of time and limited resources.

There are two approaches to solving linear equations. Direct approch and iterative approch.. In a direct approach, we solve equations under finite steps, but its time complexity is $O(n^3)$ [4], which is very high when n is large (here n is number of linear equations). This is because direct approch scales intensively with respect to the size of matrix. For indirect approch, time complexity is $O(n^2)$. In indrect approch, we do initial guess for variables' values, after that with each iteration we find close approximation. Indirect approch also required less memory and resources compared to direct approch. That is why most of the researches are done on iterative approch, as it required less computational resources compared to direct approch.

Jacobian Method and Gauss-Seidel Method are two well-known iterative approaches. [2]. Both methods are used for solving linear equations in matrix form. Let's assume we have N linear equations, so we have N variables. One can define this system in form of Ax = B, where A is the matrix of size N that contains weights of the equations, B is a vector of contestants and x is the vector of the variables. In the Jacobian method, we do initial guess the values of variables (vector x) of the linear system. After that, to solve each diagonal element in matrix A, we plugged in the values of variables (vector x). So we will get the updated values for variables, we do the same thing until it converges. Gauss-Seidel method is the same as the Jacobian method, the only difference is in the Jacobi method, we update variables after each iteration, while in the GS method, we update variables after solving each diagonal element. That is why, GS method has a good convergence rate compared to the Jacobian method, but the GS method is hard to implement in parallel systems compared to the Jacobian method, as the GS method is sequential in nature because, after solving each diagonal element, we update the values of variables.

There are many parallel algorithms of GS. Red-Black GS is a popular algorithm. [3] This is based on multi-colour ordering. This method needs a specific structure of a matrix to implement, so it depends on the sparsity pattern of a matrix. So, we can not use it with every matrix.

In paper [1], an author has come up with a row-based parallel method. This method is based on the GS method. To solve every diagonal element, we do the multiplication of constants and current variables. In this method, we do this multiplication parallelly for every variable, instead of using the loop. Then we wait for the computation of a full row of matrix A and then we update the variables. We continue this method till we get convergence. The author did not give any performance analysis in the paper.

In paper [5], the author has come up with a new parallel method to solve linear equations by combining two other algorithms. The author has combined the parallel nature of the Jacobian method and the Fast convergence of the GS method. In this new approach, one has to divide the matrix into some size of blocks. One will use the Jacobian method to solve the block, as we can easily implement the Jacobian method in a parallel system, so the block will be solved parallelly. After that, variables for that block will be updated as we do in the GS method, and the next block will be solved based on updated variables. This new approach is called the PJG method. The author did not write any code of this method.

My aim is to combine the PJG method and row-based method and come up with a new approach. I will also implement the row-based method and the PJG method. And after

that, I will compare the performance of my approach with the PJG method and row-based method. I will also try to implement my approach in distributed memory environment. My other aim is to do a performance analysis of the PJG method by choosing different block sizes, so I can use that in my approach also.

References

[1] Hadrien Courtecuisse and Jérémie Allard. Parallel dense gauss-seidel algorithm on many-core processors. In *2009 11th IEEE International Conference on High Performance Computing and Communications*, pages 139–147, 2009.

[2] L.A. Hageman. *Applied Iterative Methods.* Elsevier Science, 2016.

[3] Kawai, Masatoshi, Iwashita, Takeshi, Nakashima, Hiroshi, and Osni Marques. Parallel smoother based on block red-black ordering for multigrid poisson solver. *Lecture Notes in Computer Science High Performance Computing for Computational Science - VECPAR 2012*, page 292–299, 2013.

[4] A. Quarteroni, R. Sacco, , and F. Saleri. *Numerical Mathematics*, volume 37. Springer, 2010.