Boston University
College of Engineering
Department of Electrical and Computer Engineering

Solver Performance Multigrid vs Conjugate Gradient

Project Report

Anish Yennapusa, Mark Nudelman, Shen Gao
{anishry,mark,teammate3}@bu.edu

ENG EC 526
Parallel Programming for High Performance Computing

Contents :

1. Abstract

Isolated, the **Multigrid** algorithm is a mathematical method to solve differential equations using a hierarchy of discretizations. In other words, the process of breaking down equations into simpler components and rebuilding the equation to be solved in fewer iterations. The applications of this algorithm can be expanded to linear algebra. For example, in the familiar matrix equation Ax=b. Used as a replacement for more familiar algorithms such as **Conjugate Gradient** or **Red Black** under very specific circumstances. The conjugate gradient was designed to find the solutions of a linear system of equations, specifically, those whose matrix is positive definite. Therefore, both conjugate gradient and red black are unoptimized for when A is a large sparse matrix. Sparse means most of the matrix is filled with 0's and sparsely dotted with non-zero values. Furthermore, the multigrid method naturally gives way to parallelization, allowing for even greater optimization compared to red-black. In this project, we compared the performance of these methods in solving a linear system of equations. We evaluated based on the metrics of computational time taken, iterations to converge as well as the number of matrix-vector products computed. The parallelized multigrid algorithm outperforms the conjugate gradient as well as the red-black method.

2. Methodology

Multigrid

In this project we build a Jacobi Multigrid. The principle of the multigrid (MG) approach is to exploit the fact that high-frequency components of the residual of this system can only be represented on the fine grids, on which they can be readily reduced by well-chosen iterative solvers (called smoothing operators). Moreover, the low-frequency components of the resulting residual (on the fine grid) may also be considered as high-frequency ones on a coarser grid, where the same process may be (recursively) applied in successive ‘cycles’, among which ‘V cycles’ are most common. The process is visualized in figure 1.

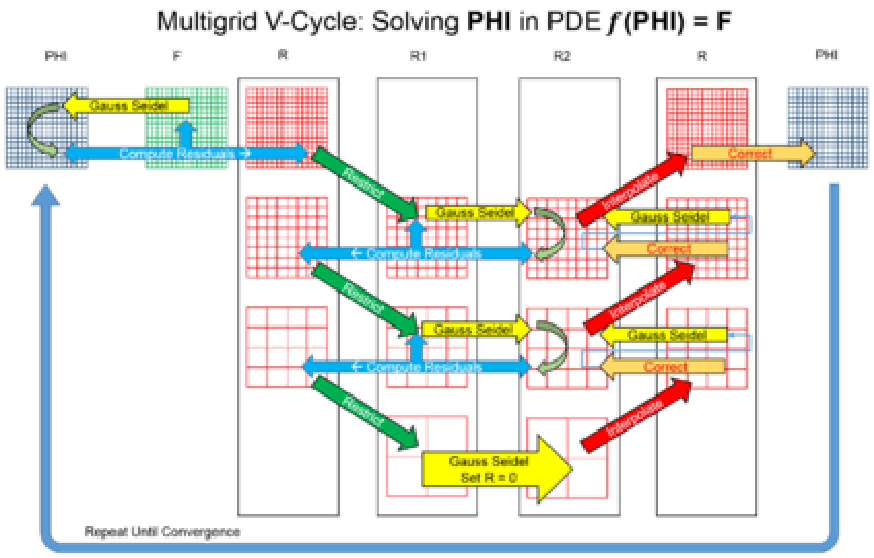


Figure 1

Conjugate Gradient

The conjugate gradient method is an algorithm to find the numerical solution of a system of linear equations. It is applicable to those equations whose matrix is positive semi-definite and is effective on sparse matrices. The algorithm can be implemented as a set of recursive steps. Essentially, the conjugate gradient method finds the optimal solution by making a series of steps through a solution space. It calculates the gradient at a given point and moves in the direction opposite to the gradient by a specific step value. This process is repeated iteratively until the algorithm converges on the solution. In the case of solving linear systems, the conjugate gradient method has the useful property of having its iteration count bounded by the size of the input matrix. I.e The number of iterations in the conjugate gradient method cannot exceed the dimension of the matrix A.

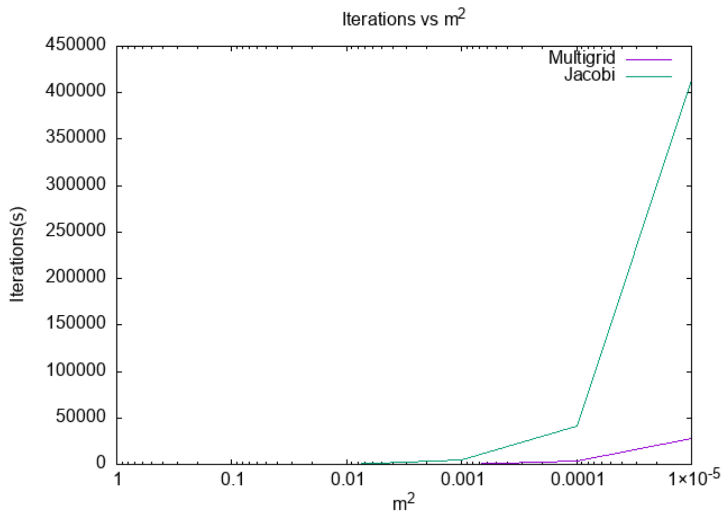
Red Black

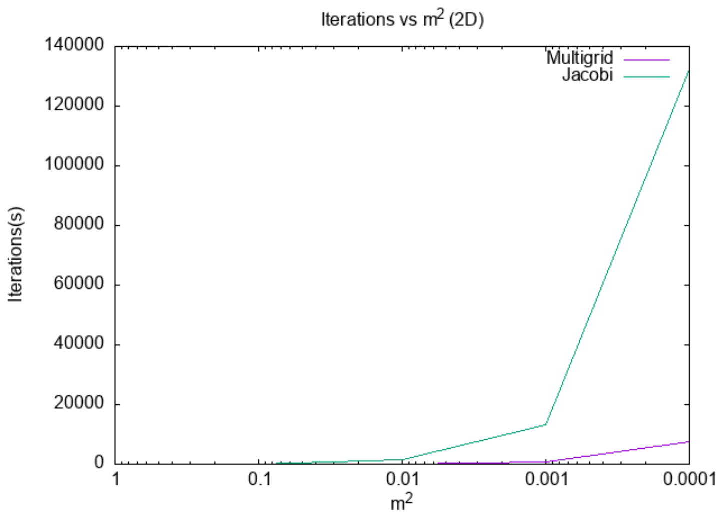
The red black method is a Jacobi solver method that exploits the fact that the order in which the variables are processed can improve efficiency. The algorithm is implemented by improving the Gauss-Seidel method which cannot be parallelized due to data dependency. In Red Black, the nodes are split into odd and even and processed independently such that they avoid dependency issues. In this implementation, the solution achieves a 2 times speed improvement over the direct Jacobi implementation. Furthermore, it can achieve greater speed by exploiting parallelism.

3. Approach

In our approach, we have implemented the three aforementioned algorithms in C++. The problem was formulated as a solution to take in the matrices corresponding to a system of linear equations Ax=b and computed the results. The conjugate gradient method was implemented to count the number of iterations until the residual converged. Similarly, we also count the iterations and residual values through each step of the multigrid and red-black algorithms. Furthermore, the execution time is measured and correlated against the input size of the problem for all three algorithms. The multigrid and red-black algorithms were parallelized using OpenACC. For visualization, we used the GNUPlot software to map and correlate solutions. The algorithms were compared on specific metrics; the number of iterations to convergence, the execution time of the algorithm, and the asymptotic behavior of the residual when plotted against the iteration count. We ran the program over multiple test cases of varying sizes to verify our conclusions.

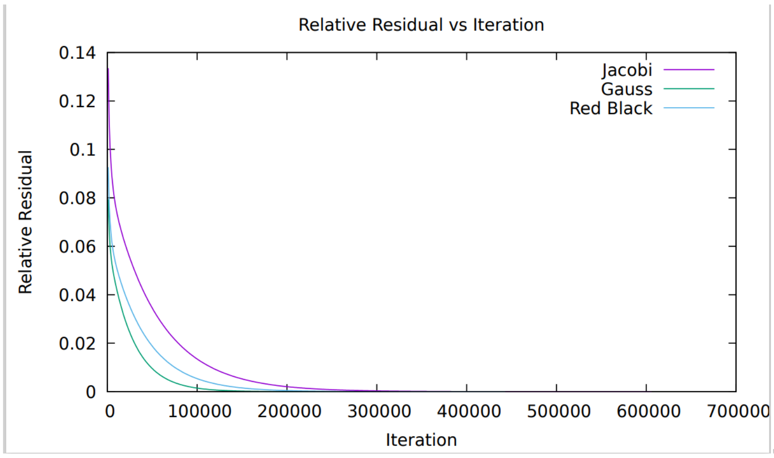
4. Results

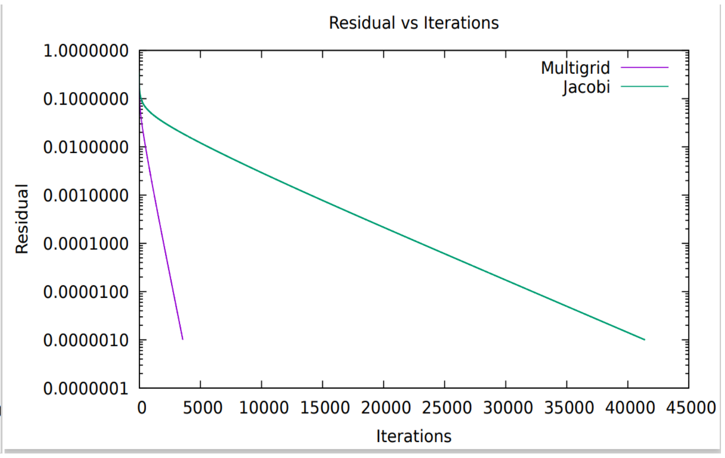




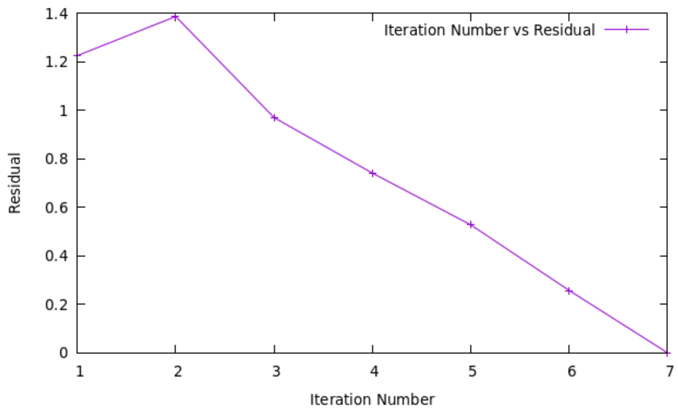
We compared the three algorithms based on the previously discussed metrics. The results are as follows:

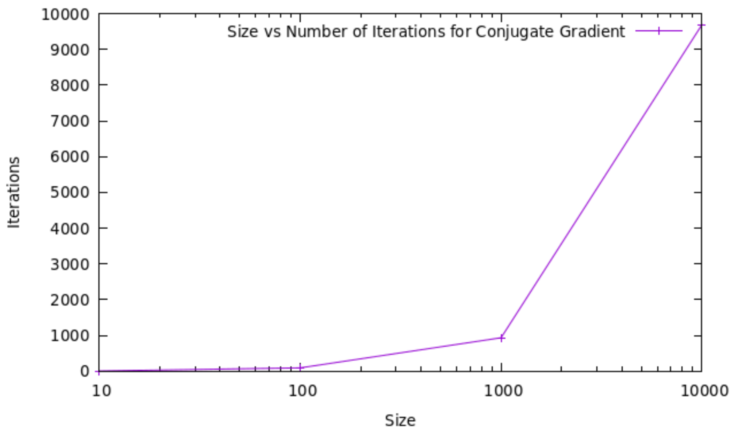
The graphs above compare the one and two-dimensional multigrid to Jacobi, respectively. It is observed that multigrid outperforms Jacobi on every m^2 value. The difference in the number of iterations between the two algorithms increases as the m^2 value decreases, as expected.





As seen in the graphs above, multigrid has the steepest drop for both the relative residual and the residual. In less than 5000 iterations, the multigrid solver reached the residual tolerance of 1*10^-6, compared to the Jacobi algorithm, which took over 40000 iterations to converge. The red-black code was much closer in performance than the Jacobi as is seen in the Relative Residual vs Iteration graph.





When testing a small array of size 7, the multigrid algorithm was able to converge in O(N) complexity. On the other hand, conjugate gradient also performed well on small array sizes. As the number of elements climbed to 10,000, conjugate gradient kept up and converged in the same complexity as multigrid.

5. Conclusion

From the results we observe that the parallelized multigrid solution performs significantly better than the other solutions. Multigrid solver converges in relatively fewer iterations as the size of the problem increases. This asymptotic behavior favors multigrid as the size of the problem increases. Additionally, the Multigrid solver also converges in relatively less time as the size of the problem increases. Compared to the Conjugate gradient, the multigrid solution also performs fewer matrix-vector computations thus saving on computational workload. These results favor multigrid as an optimal solution choice. However, there may be hybrid solvers which may outperform multigrid for specific problem sizes.

6. Team Contributions

Name	**Tasks**
Anish Yennapusa	CG Implementation, Comparison, Report
Shen Gao	Red-black implementation, Graphs, Parallelization
Mark Nudelman	Multigrid Implementation, Presentation, Writeup

7. References

[1] https://github.com/brower/EC526_2022/tree/main/openACC_demo

[2] https://github.com/brower/EC526_2022/tree/main/*Project_Solver_Jacobi_CG_MG

[3] https://github.com/brower/EC526_2022/tree/main/HW6code/MG

[4] Wikipedia contributors. (2022, February 11). Conjugate gradient method. In *Wikipedia, The Free Encyclopedia*. Retrieved 10:58, May 6, 2022, from

[5] Bolz, Jeff, et al. "Sparse matrix solvers on the GPU: conjugate gradients and multigrid." *ACM transactions on graphics (TOG)* 22.3 (2003): 917-924.