High Performance Computing I
Project

Sagar Bhatt
Person Number: 50170651
Department of Mechanical and Aerospace Engineering,
University at Buffalo

Contents

I Introduction **2**

II Method of Solution: **2**
I Compressed Row Storage Dot Product: . . . 3
II Conjugate Gradient Algorithm: . . . 3

III Results **4**
I Analytical Solution . . . 4
II Numerical Solution . . . 5
III Error in Solution . . . 5
IV Strong Scaling . . . 6
V Efficiency . . . 8
V.1 Strong Scaling efficiency . . . 8

List of Figures

1 Analytical solution on a 1000×1000 Grid . . . 4
2 Numerical solution on a 1000×1000 Grid . . . 5
3 Error norm vs grid size . . . 6
4 Log-Log plot of Error norm vs grid size . . . 6
5 Speedup vs number of procs for strong scaling . . . 7
6 Serial fraction vs number of procs for strong scaling . . . 7
7 Scaling efficiency vs number of procs for strong scaling . . . 8

I. Introduction

Laplace's equation is a second-order partial differential equation named after Pierre-Simon Laplace who first studied its properties. This is often written as:

$$\nabla^2 u = 0$$

or

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$
$$u(x,0) = sin(\pi x),$$
$$u(x,1) = sin(\pi x)e^{-\pi}$$
$$u(0,y) = u(1,y) = 0$$

where $\Delta = \nabla^2$ is the Laplace operator and is a scalar function.

Laplace's equation is the simplest example of elliptic partial differential equations. The general theory of solutions to Laplace's equation is known as potential theory. The solutions of Laplace's equation are the harmonic functions, which are important in many fields of science, notably the fields of electromagnetism, astronomy, and fluid dynamics, because they can be used to accurately describe the behavior of electric, gravitational, and fluid potentials. In the study of heat conduction, the Laplace equation is the steady-state heat equation.[4]

The analytical solution to be solved in this problem is given by $u(x,y) = sin(\pi x)e^{\pi y}$, The equation itself needs to be solved by solving the following finite difference equation by conjugate gradient method:

$$\Delta u = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}$$

where, $u_{i,j} = u(x_i, y_i),\ i = 1,2,3...N;\ j = 1,2,3...M$

The conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.[3]

II. Method of Solution:

To solve

$$\Delta u = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}$$

we express the equation in the matrix from and the equation turns into the following expression:

$$\mathbf{AX} = \mathbf{B}$$

where A is a sparse pentadiagonal matrix. One of the most general schemes for storing sparse matrices is the Compressed Sparse Row storage format. It consists of three arrays: a real array A(1 : nnz) to store the nonzero elements of the matrix row-wise, an integer array JA(1 :nnz) to store the column positions of the elements in the real array A, and, finally, a pointer array IA(1 : n + 1), the i-th entry of which points to the beginning of the i-th row in the arrays A and JA. To perform the matrix-by-vector product y = Ax in parallel using this format, note that each component of the resulting vector can be computed independently as the dot product of the i-th row of the matrix with the vector.[1]

I. Compressed Row Storage Dot Product:

```
for 1=1:n
    s ← 0
        for k ← A.row-ptr(i) to A.row-ptr(i+1)-1
            s ← s + A.val(k)• x(A.col-ind(k))
        y(i) ← s
```

II. Conjugate Gradient Algorithm:

Solving $\mathbf{AX} = \mathbf{B}$ can be equivalent to finding the minimum of the quadratic equation $f(x) = \frac{1}{2}x^T Sx - B$. It improves the steepest decent method by avoiding repetition of steps. The steps are taken in 'A-Orthogonal' direction and information from prior steps is utilized to avoid redundancy.

$$d_0 = r_0 = b - Ax_0 \tag{1}$$

Iterate:

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i} \tag{2}$$

$$x_{i+1} = x_i - \alpha_i d_i \tag{3}$$

$$r_{i+1} = r_i - \alpha_i A d_i \tag{4}$$

$$\beta_i = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \tag{5}$$

$$d_{i+1} = r_{i+1} + \beta_i d_i \tag{6}$$

This method is faster than steepest decent and typically converges in n iterations. It can also be used for non linear functions. It does not undo the minimization performed by previous steps. We will also employ jacobi preconditioning in this method. The preconditioner in this case would be *Diag(A)*. [2]

The computed solution is being analyzed on internal as solution is fixed at the boundaries by boundary conditions. This would result in an $\mathbf{A}$ matrix of size $(N-2)^2 \times (N-2)^2$, which means for a 1000×1000 grid would need $998^2 \times 998^2$ $\mathbf{A}$ matrix. This means $\mathbf{A}$ matrix itself would consume $998^2 \times 998^2 \times 8$ bytes = 7391.1 TB. The impracticlity of this size validates the use of CRS format

which stores just the non zero values. That would bring down the size significantly. The value array, row pointer array, column indices would each be of size 3237946 , 647790 and 3237946 consuming 24.7 GB, 2.47 GB and 12.35 GB of memory respectively. Operating on arrays this size and on any higher resolution would require many processors. Hence, 132 processors have been used to do the scaling study of this problem.

Apart from the above techniques, BLAS libraries were used to perform simple vector-vector operations. The BLAS functions used were: `ddot`, `dcopy`, `daxpy` and `dnrm2`.

III. Results

I. Analytical Solution

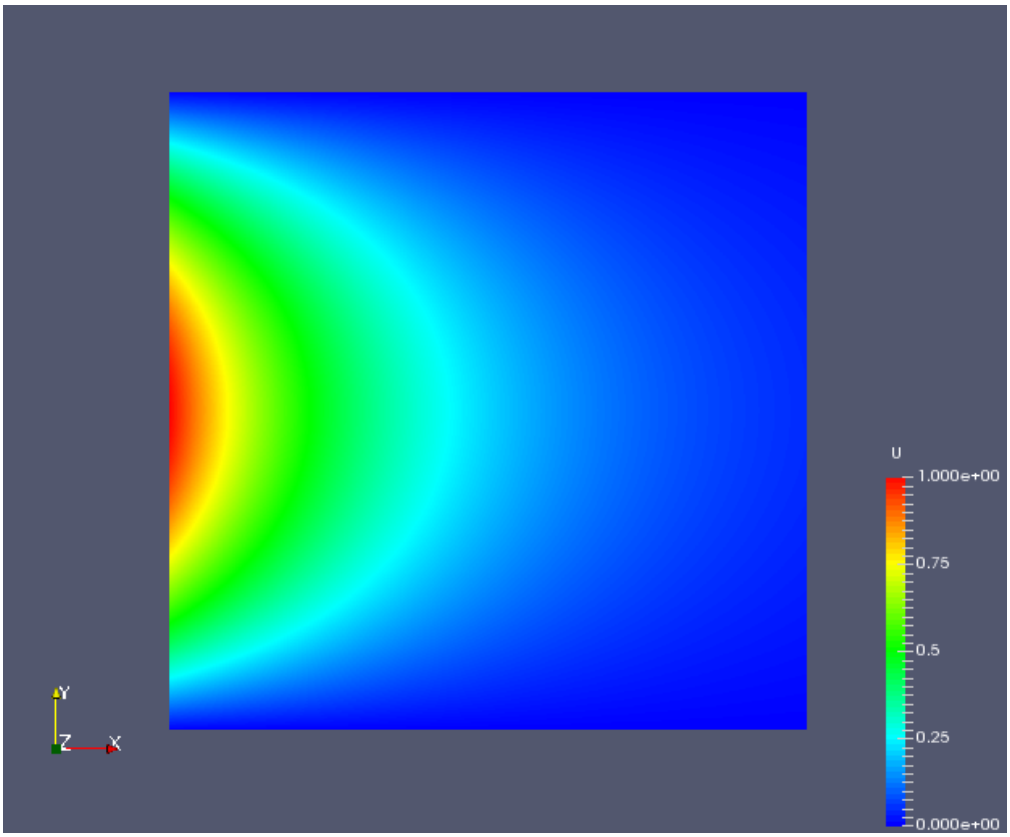

Figure 1: *Analytical solution on a* 1000×1000 *Grid*

II. Numerical Solution

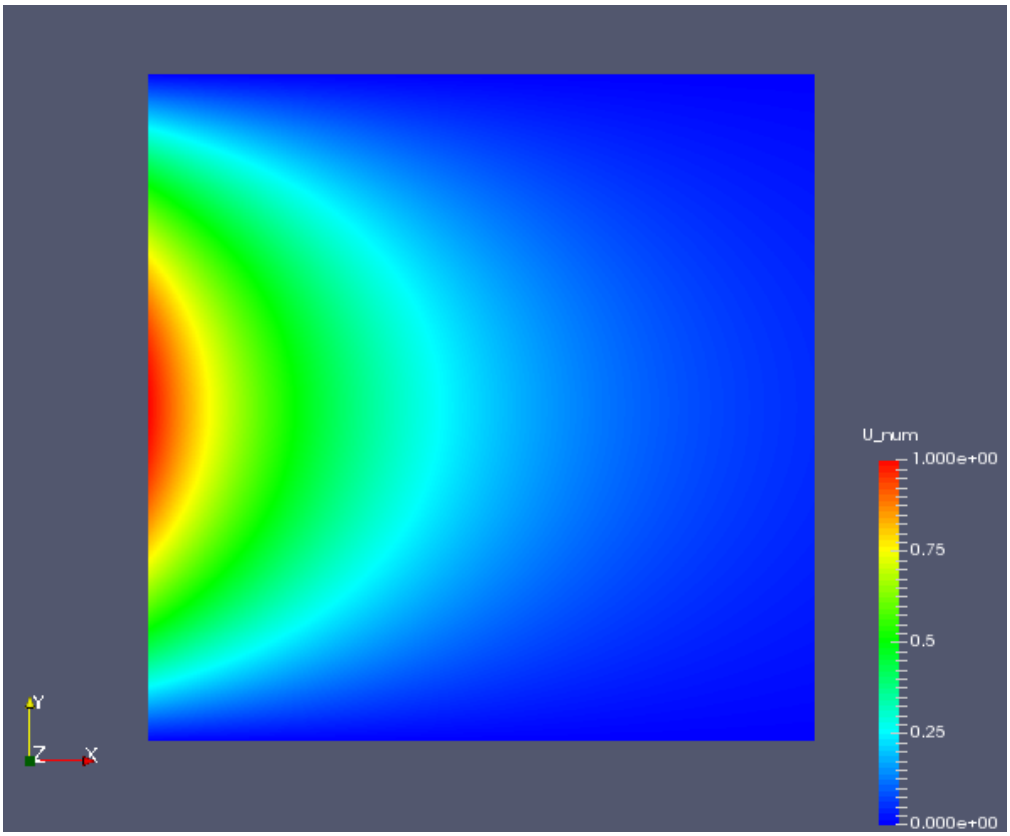

Figure 2: *Numerical solution on a* 1000×1000 *Grid*

III. Error in Solution

The Error was also computed between the analytical solution and the numerical solution. The norm of error is presented here with respect to the grid size, N i.e. for a $N \times N$ grid. A log-log plot is also presented here for Error norm vs the grid size. We can see that the error between the analytical solution and the computed solution decreasing as the grid becomes finer. This result validates the accuracy of the solution presented here.

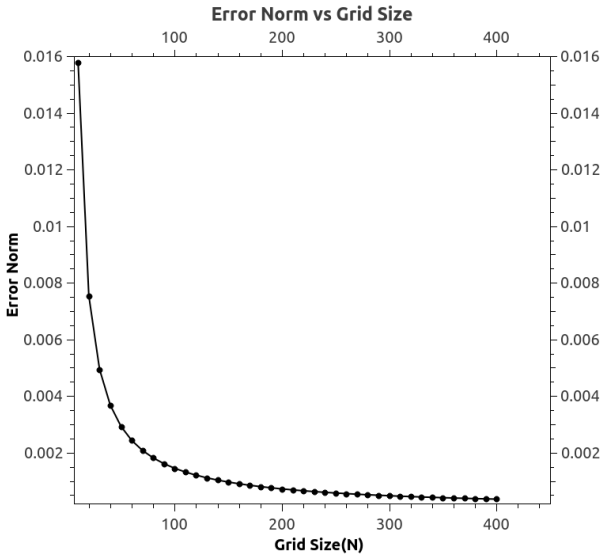

Figure 3: *Error norm vs grid size*

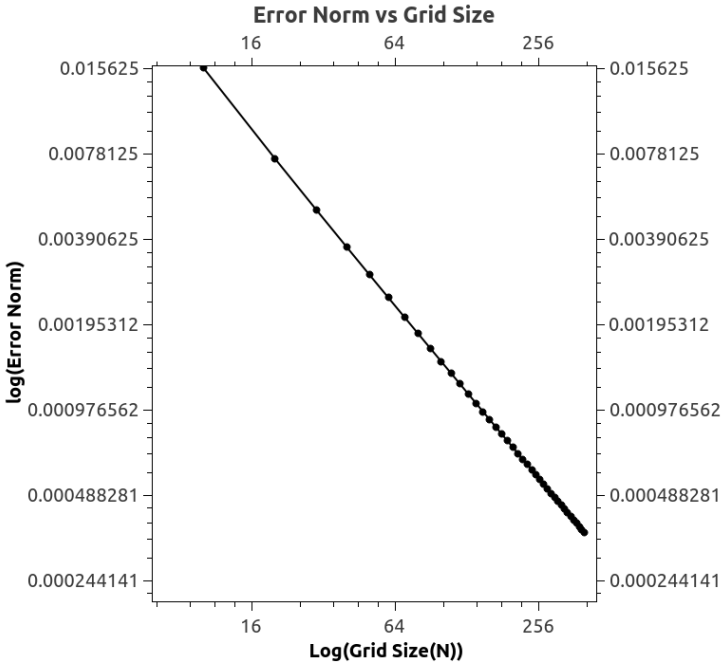

Figure 4: *Log-Log plot of Error norm vs grid size*

IV. Strong Scaling

Speedup was computed using the formula: $s = \dfrac{T(1)}{T(p)}$

The serial fractions were computed using the Karp-Flatt metric: $f = \dfrac{1/s - 1/p}{1 - 1/p}$

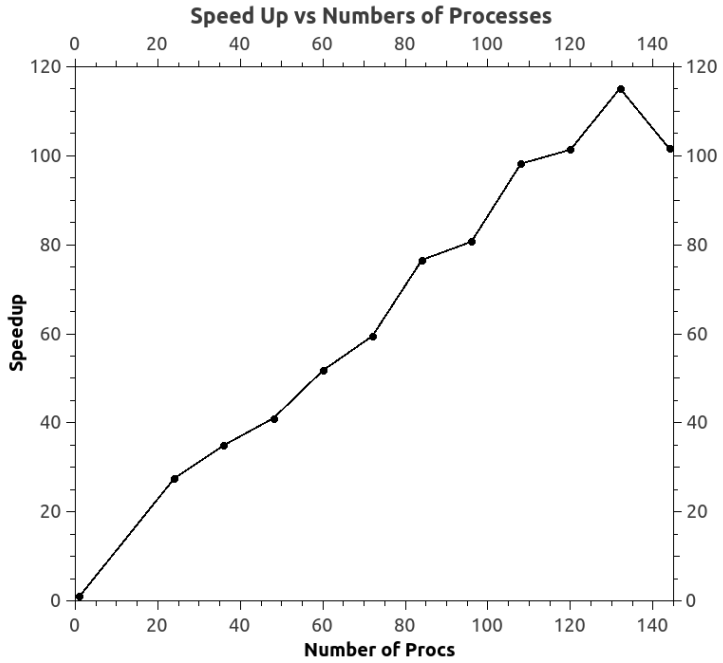

Figure 5: *Speedup vs number of procs for strong scaling*

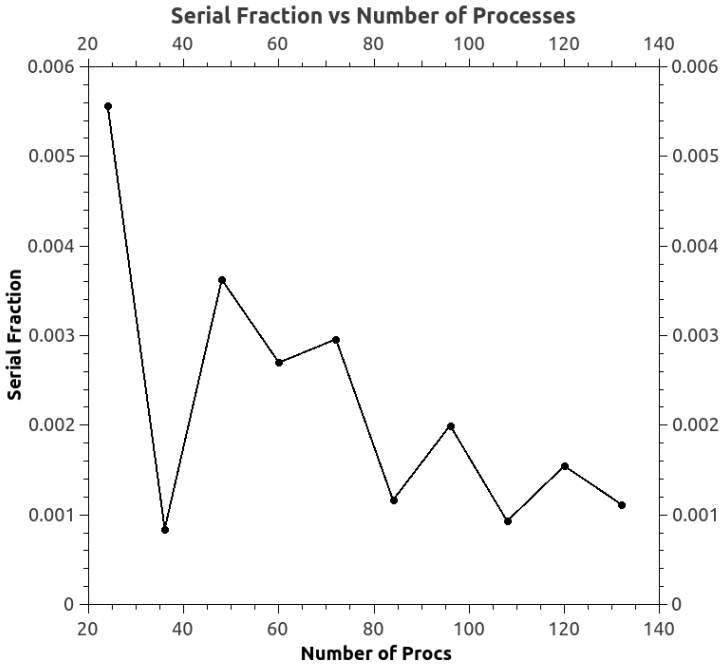

Figure 6: *Serial fraction vs number of procs for strong scaling*

V. Efficiency

Strong Scaling efficiency

Strong Scaling efficiency was computed using:

$$\eta_s = \frac{T(1)}{p \times T(p)} \times 100\%$$

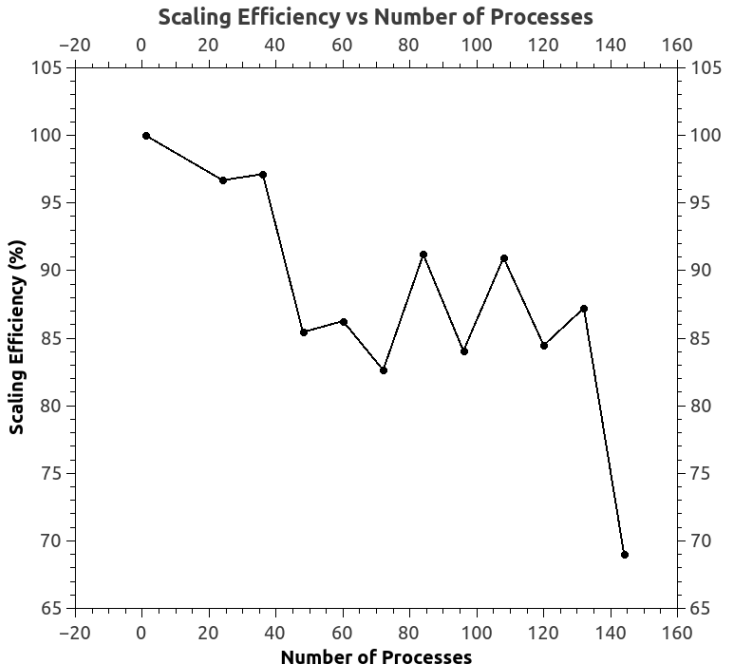

Figure 7: *Scaling efficiency vs number of procs for strong scaling*

References

[1] Rohit Gupta. *Implementation of the Deflated Preconditioned Conjugate Gradient Method for Bubbly Flow on the Graphical Processing Unit (GPU).* PhD thesis, Citeseer, 2010.

[2] Spencer Patty. Math 639, compressed row storage (crs)format for sparse matrices. `http://www.math.tamu.edu/~srobertp/Courses/Math639_2014_Sp/CRSDescription/CRSStuff.pdf`, 2014.

[3] Wikipedia. Conjugate gradient method — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Conjugate_gradient_method&oldid=751363716`, 2016. [Online; accessed 25-November-2016].

[4] Wikipedia. Laplace's equation — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Laplace%27s_equation&oldid=746563954`, 2016. [Online; accessed 28-October-2016].