HW6

Deep Patel
Jacob Hurst

December 13, 2018

1 What

The overall goal of this homework is to run a heat equation experiment, in parallel using MPI, on CARC using backward Euler. This classic model PDE is used throughout computational science, e.g., for particle diffusion, the Schroedinger equation, thermal diffusivity, financial mathematics (Black holes), image analysis, data analysis (graph Laplacians), etc.

2 How

We modeled the heat equation in two-dimensions.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(t, x, y) \tag{1}$$

where (x,y) belong to the domain of unit square and the initial condition, and exact solution are defined by some $u_exact(t, x, y)$ using the method of manufactured solutions, and the Dirichlet boundary conditions are defined by,

$$u(t, x, y) = u_{exact}(t, x, y) \tag{2}$$

Moreover, the time domain is t in [0,0.5]

We conducted a weak and strong scaling study. The scaling studies were carried out on CARC's compute cluster. This is because the computations are demanding and CARC compute nodes have sufficient cores for the scaling studies.

2.1 Serial Computing Tasks

We implemented the method of manufactured solutions to manufacture an initial-boundary value problem for which the true solution is known. We chose a function where the boundary conditions are nonzero. The below equation was used which results in a nonzero boundary condition.

$$u(t, x, y) = sin(\pi * t) * cos(\pi * x) * cos(\pi * y) \tag{3}$$

The forcing term $f(t, x, y)$ is selected as indicated below:

$$f(t, x, y) = u_t - u_{xx} - u_{yy} \tag{4}$$

This results in our case, for $f(t, x, y)$ to be:

$$f(t, x, y) = \pi * cos(\pi * t) * cos(\pi * x) * cos(\pi * y) + 2 * (\pi^2) * sin(\pi * t) * cos(\pi * x) * cos(\pi * y) \tag{5}$$

The Drichlet boundary condition is:

$$g(t, x, y) = u(t, x, y) = sin(pi * t) * cos(pi * x) * cos(pi * y) \tag{6}$$

2.2 Verification of the L2 Norm

The integral of $e = u_{approx} - u_{exact}$ over the entire space-time domain was done using the midpoint rule as in HW4 to do this.

$$\|e_{L_2}\| = \left\{ \int_0^1 e(t, x, y)^2 dx dy dt \right\}^{1/2} \tag{7}$$

Two different spacings were accounted for: the time-step size ht, and the spatial grid spacing h. the number of points in x and y were allowed to be equal, so that there is a uniform grid spacing (h) in space. Error is measured only at the final time-step in order to simplify the parallel case.

2.3 Result of Serial Case: Error of L2 Norm

A log-log plot was generated (shown in Figure 1) of the error versus a decreasing sequence of grid spacings, and shows that the error decreases with the expected rate, which is 2. This is due to the fact that, Eulers method is only $O(h_t)$ for a time-step size ht, but the spatial discretization is $O(h^2)$

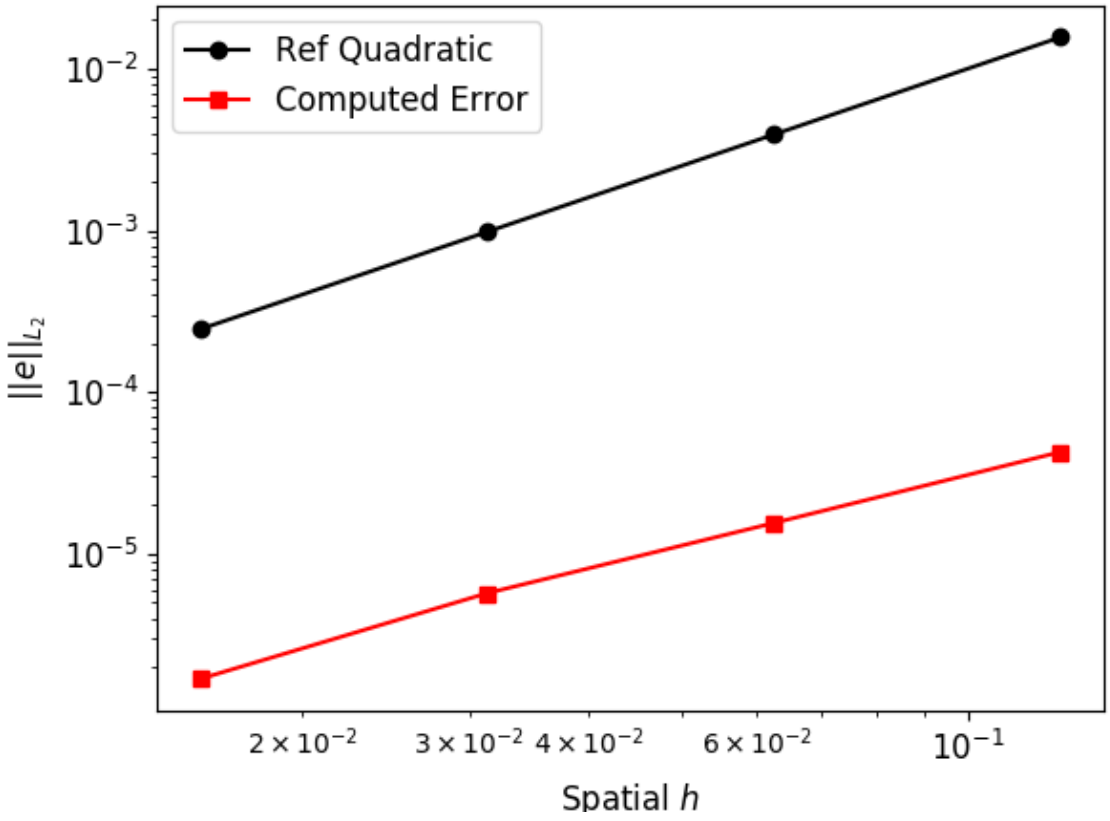

Figure 1: This figure shows the quadratic decrease of error versus a decreasing sequence of grid spacings for serial case.

If the ht/h^2 ratio changes, then the convergence of Jacobi will deteriorate significantly. Therefore, ratio ht/h^2 should be kept at about 3.8.

3 Parallel Case

MPI is utilized to parallelize our serial code. A one-dimensional domain partition strategy, as in HW4, is employed. The threads from HW4 are analogous to the processors used in this assignment.

Parallelizing the code required the implementation of a modified vector norm function and the implementation of a modified matrix vector product. These modified functions were used in our implementation of Jacobi method to invert the system of equations and to calculate the error of our problem solution.

1. A parallel code for matrix-vector multiplication for the Poisson operator is written. The matrix-vector multiply does not account for the forcing function and boundary conditions: which are added separately.

2. Parallel vector norm function, for computing error and residual norms. This uses MPIs Allreduce function to accumulate every processors contribution to a norm.

Additionally, if Jacobi fails to converge, rank 0 notifies the user. The parallel code is verified to be working correctly by doing another convergence study of the error, verifying an error decay rate of 2 in a log-log plot: as shown in Figure 2.

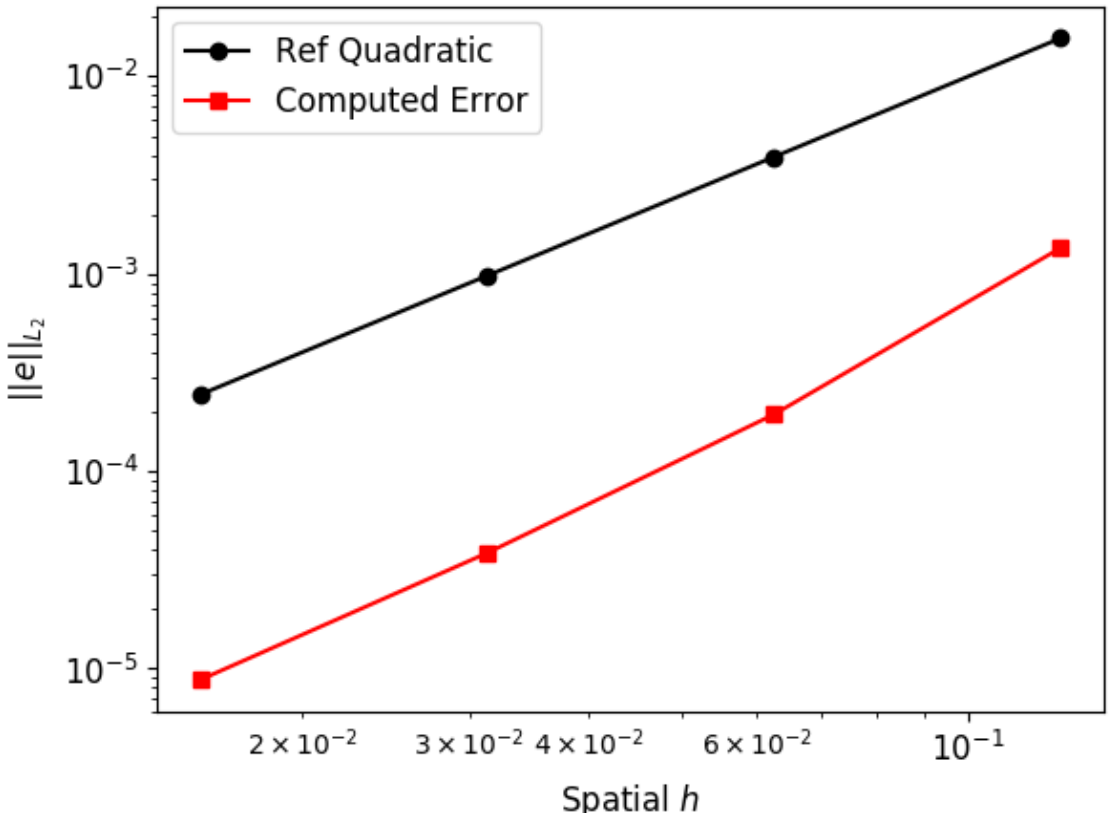

Figure 2: This figure shows the quadratic decrease of error versus a decreasing sequence of grid spacings for parallel case.

3.1 Plots of the solution at the final time

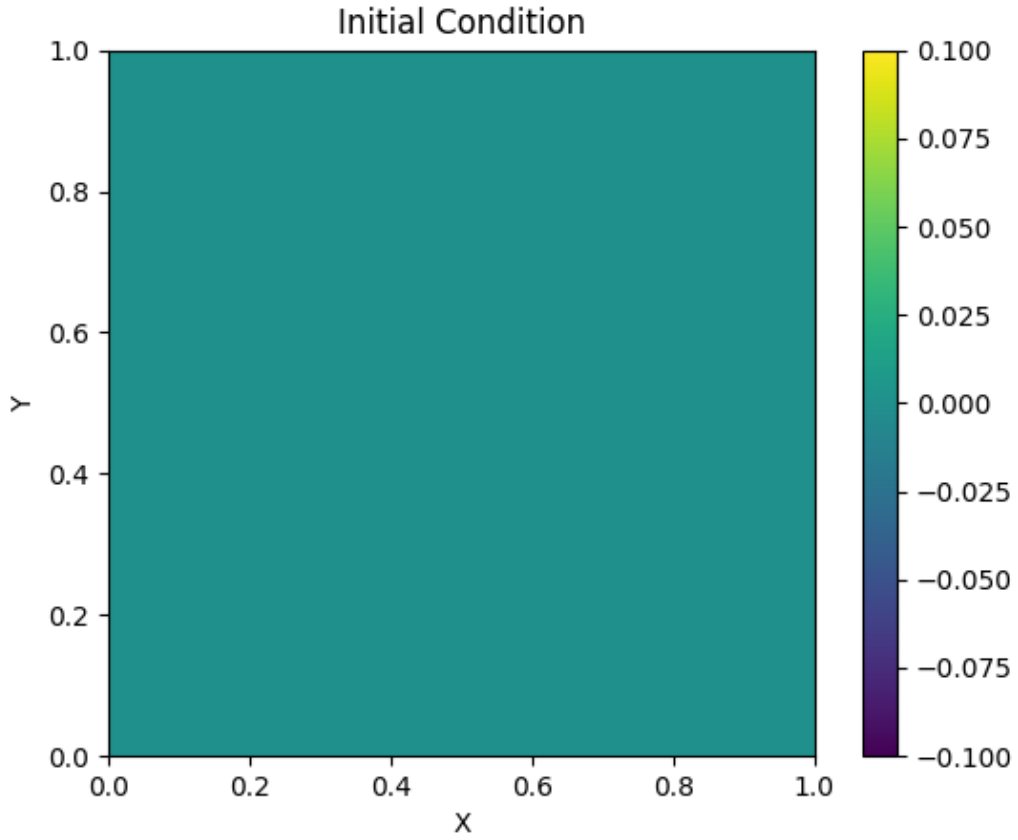

Figure 3: This figure shows the initial condition of the heat-equation solution

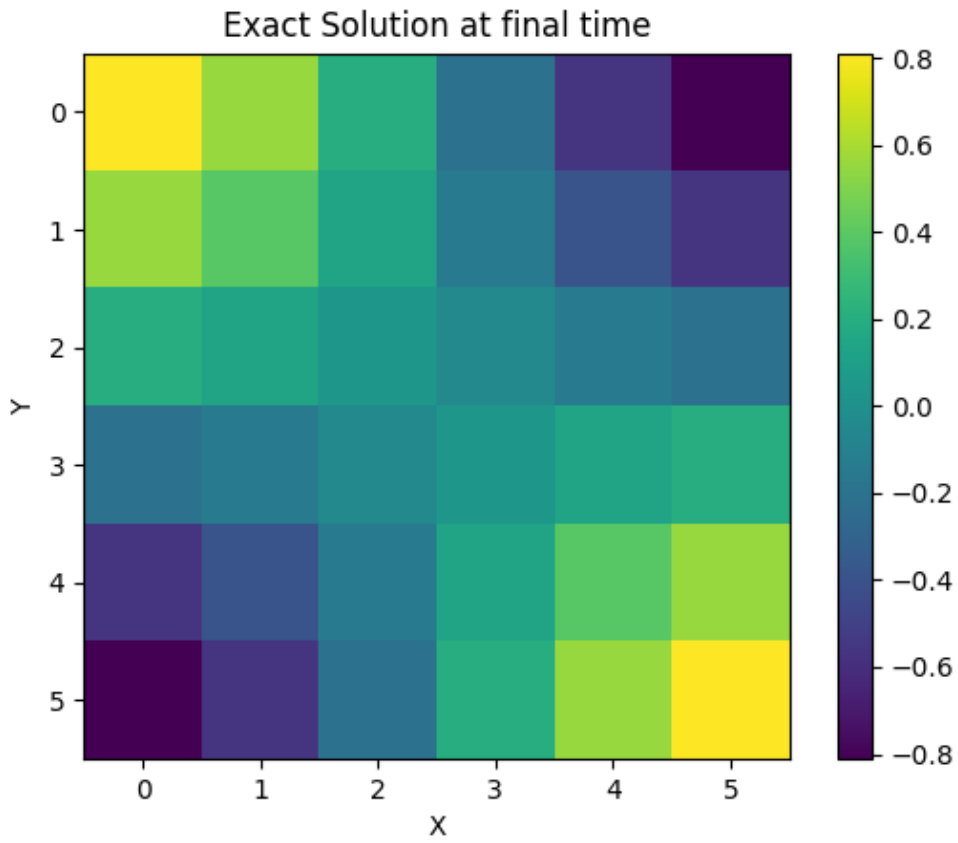

Figure 4: This figure shows the exact theoretical final solution to the heat-equation solution using 8 time spacings 8 spatial grid spacings

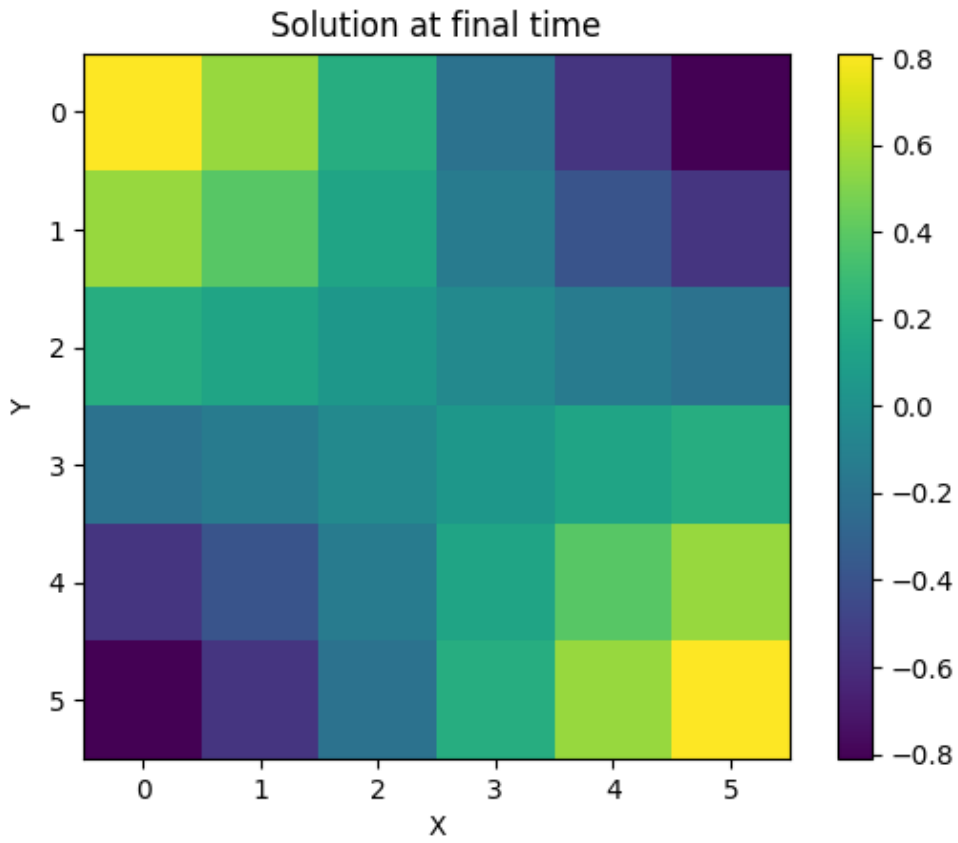

Figure 5: This figure shows the experimental final solution to the heat-equation solution using 8 time spacings 8 spatial grid spacings

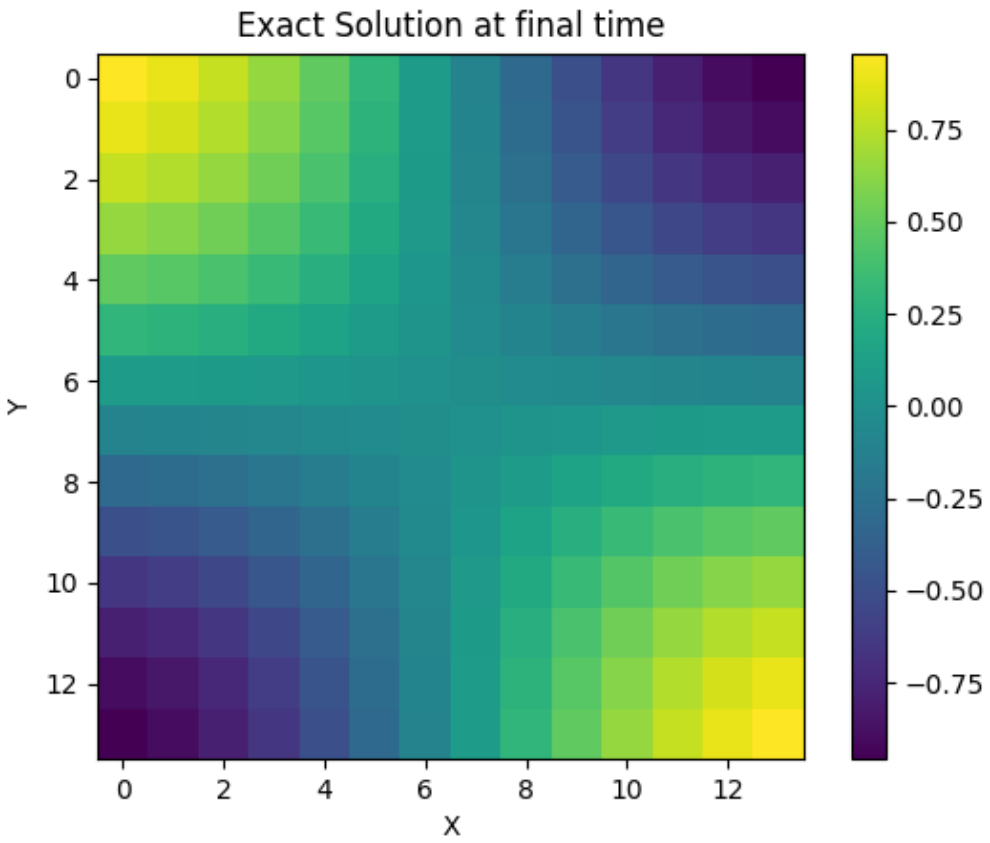

Figure 6: This figure shows the exact theoretical final solution to the heat-equation solution using 32 time spacings 16 spatial grid spacings

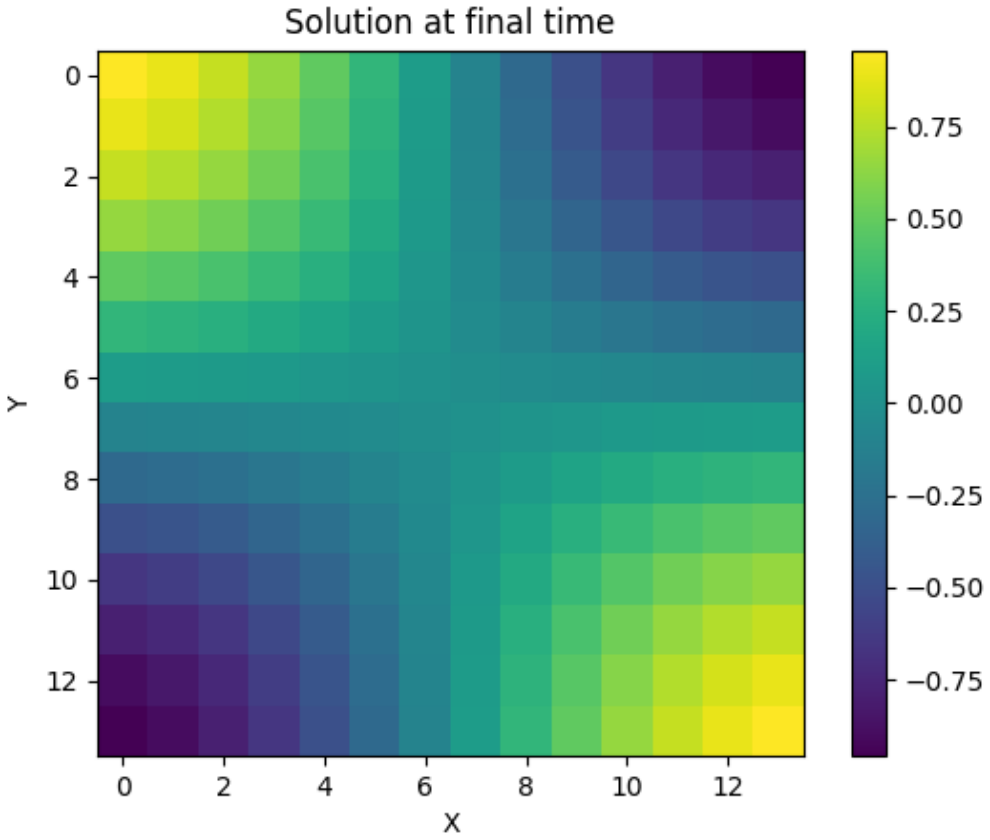

Figure 7: This figure shows the experimental final solution to the heat-equation solution using 32 time spacings 16 spatial grid spacings

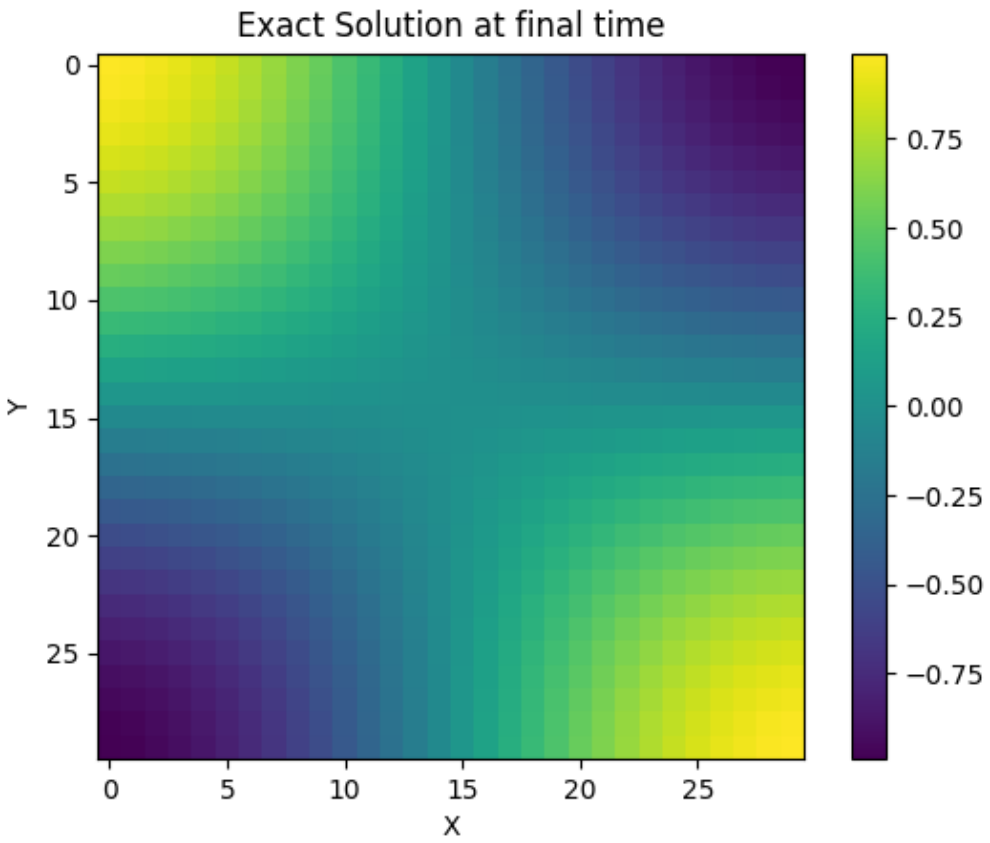

Figure 8: This figure shows the exact theoretical final solution to the heat-equation solution using 128 time spacings 32 spatial grid spacings

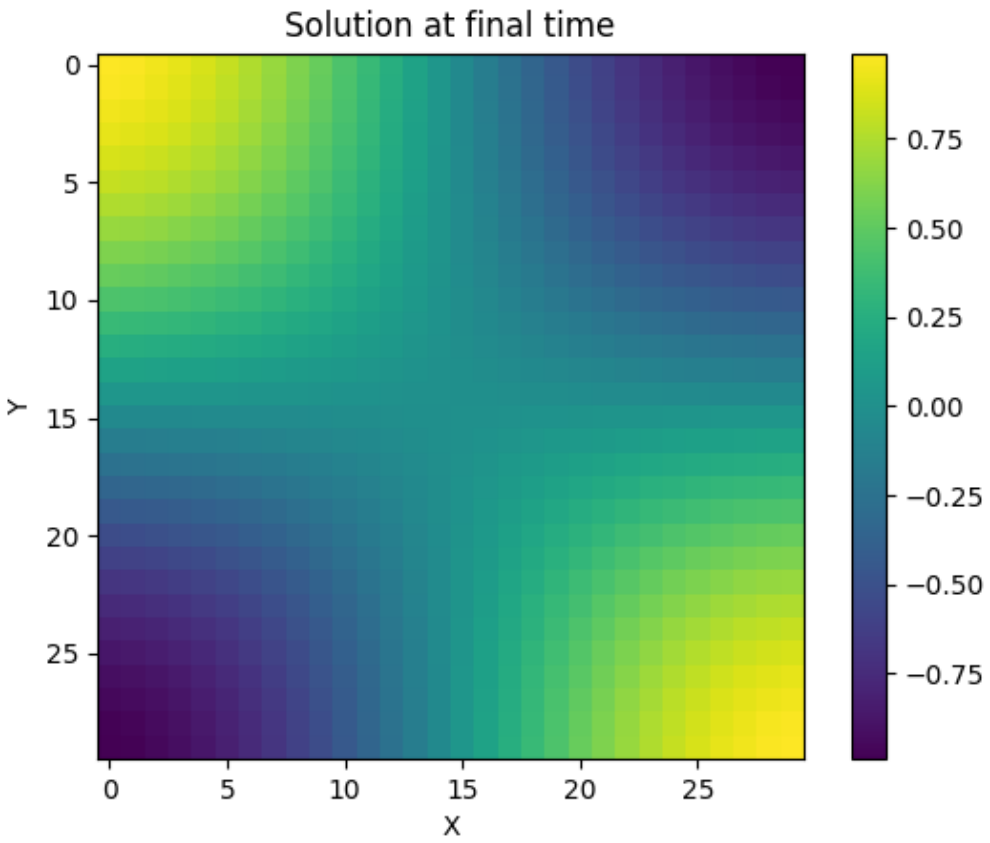

Figure 9: This figure shows the experimental final solution to the heat-equation solution using 128 time spacings 32 spatial grid spacings

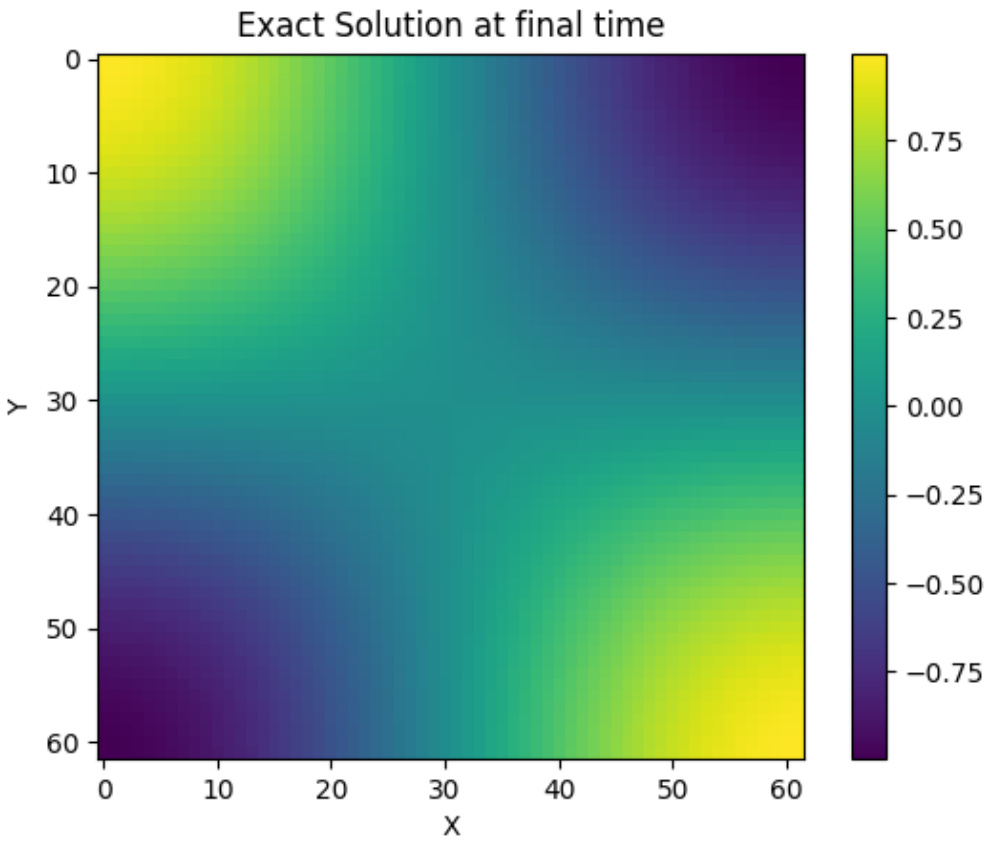

Figure 10: This figure shows the exact theoretical final solution to the heat-equation solution using 512 time spacings 64 spatial grid spacings

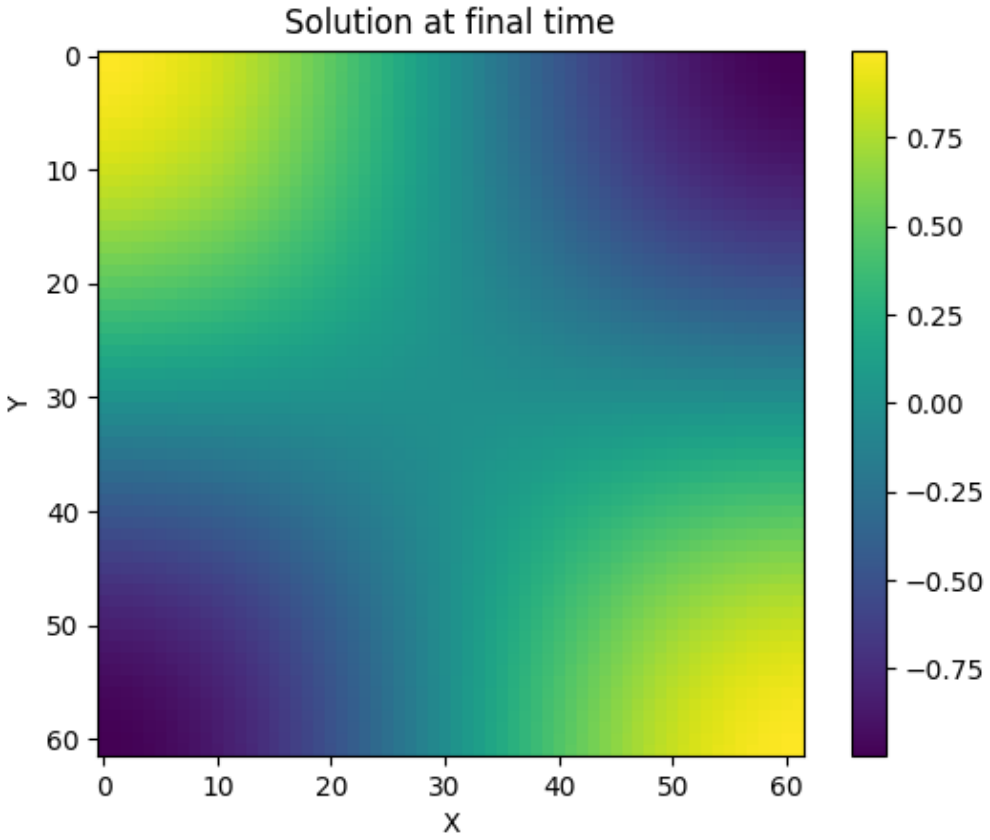

Figure 11: This figure shows the experimental final solution to the heat-equation solution using 512 time spacings 64 spatial grid spacings

3.2 Weak and Strong Scaling study

A weak and strong scaling studies were carried out on Wheeler. The ht/h^2 ratio from the convergence study is maintained at approximately 3.8. This was done by changing the final time T. If the ht/h^2 ratio changes, then the convergence of Jacobi will deteriorate significantly.

In the weak scaling case, a problem size of about 2000 spatial grid points per processor was maintained. As the number of processors increased, the grid size had to be increased to maintain ht/h^2 ratio. Four test cases were tried; 1 node with 1 core on 48x48 spatial grid points, 1 node with 4 cores on 96x96 spatial grid points, 2 nodes with 8 cores on 192x192 spatial grid points, and 8 nodes with 8 cores on 384x384 spatial grid points. A minimum of 5 timings was taken for each data point. In the figure below, the resulting timings are shown alongside 4^x. It can be observed that the timings behave similarly as we scale on processors/problem sizes. Given that the problem size is approximately the same per processor as the number of processors increases, the increasing trend we observe may be attributed to the general overhead in communications between processes as well as the increase in the number of time-steps.

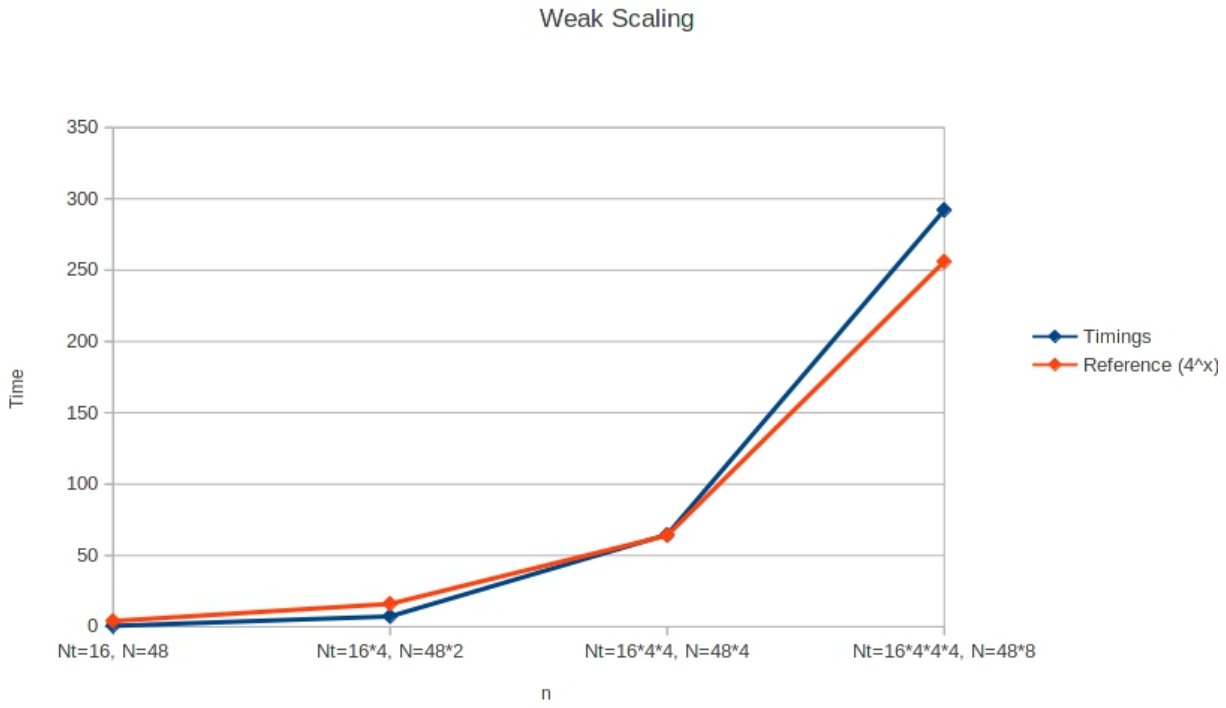

Figure 12: This figure shows the experimental timing data as the problem size increases.

In the strong scaling case, the problem size of 256x256 spatial grid points with 256 time-steps was maintained for 1 node with 2 cores, 1 node with 4 cores, 1 node with 8 cores, 2 nodes with 8 cores, 4 nodes with 8 cores, 8 nodes with 8 cores, and 16 nodes with 8 cores. As above, a minimum of 5 timings was taken for each data point. In the figure below, the resulting timings are shown. It can be observed for the first 3 data points (single node with 2, 4, and 8 cores), timings that are significantly higher than the other timings. Once we increase to 4 nodes with 8 cores we see significant improvements in the timings but as our nodes increase further to 8 and 16, the improvements diminish and even worsen. This is likely due to the scalability of the parallel solution as it relates to increasing nodes. As our number of nodes involved in calculating the solution grows, so does our overhead in communication between these nodes. The increasing communication requirements constrain the improvements of parallelizing the code.

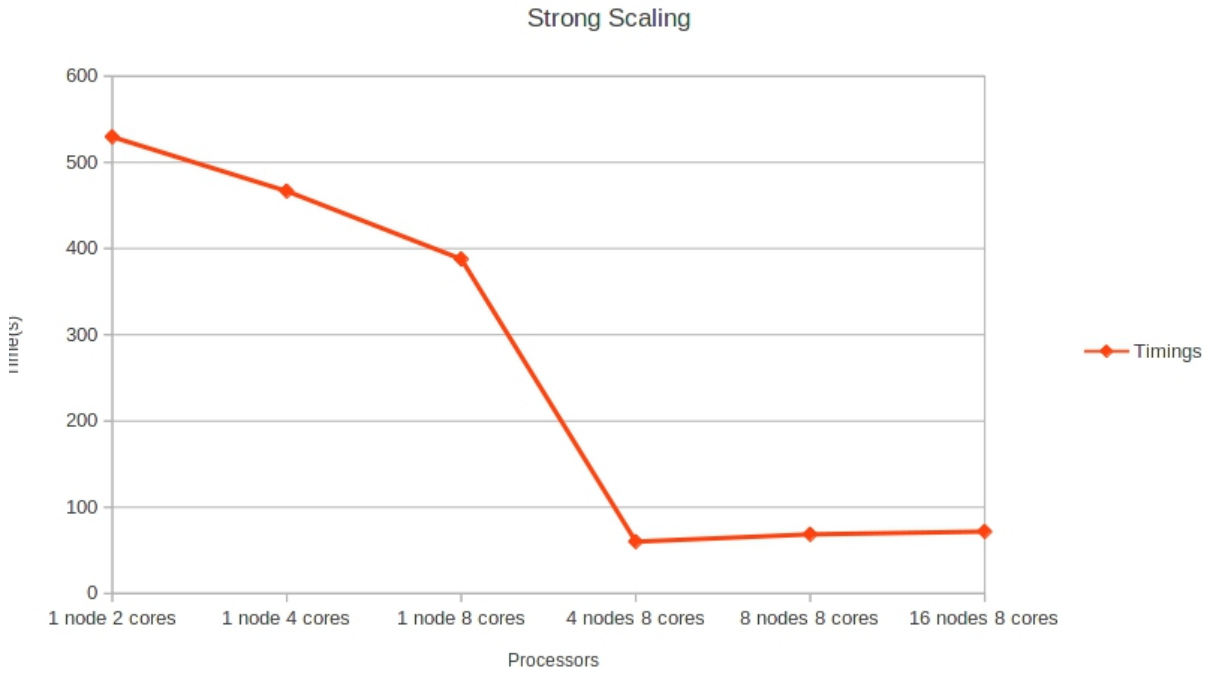

Figure 13: This figure shows the swamping of nodes above 4 nodes 8 cores