James Hahn
CS1645
High Performance Computing

Verification

Below are the results of the verification test with source term $S(x, y) = xe^y$ and boundary conditions $T(0, y) = 0$, $T(2, y) = 2e^y$, $T(x, 0) = x$, and $T(x, 1) = xe$. The exact solution is $T = xe^y$. The contour map (Figure 1) can be found below plotted on the domain $0 \leq x \leq 2$ and $0 \leq y \leq 1$ with one thread. Additionally, Figure 2 displays the contour map with four threads on a grid size of 400x200. A convergence rate of 10^{-12} was used on an iteration-by-iteration basis. All results in this report are from Jacobi algorithm requested in the project description.

Grid Size	**L-∞ Error**
5x5	1.0929×10^{-3}
11x11	1.8886×10^{-4}
21x21	4.8004×10^{-5}
41x41	1.2026×10^{-5}
51x51	7.6982×10^{-6}
101x101	1.9238×10^{-6}
201x201	4.7511×10^{-7}

The discretization is verified since the above tests were ran on successively finer grids. The algorithm was also tested on odd-dimension grids, such as 11x5, 21x11, and 41x21. The algorithm still worked and converged. It is important to note these lopsided dimensions will not necessarily work with more than 1 thread because the project specifications clarify each dimension is evenly divisible by the number of threads in that dimension. For example, if there are 3 threads in the y-dimension and 2 threads in the x-dimension, the grid dimensions must be divisible by 3 in the y-dimension and divisible by 2 in the x-dimension or the program will exit with an error.

Figure 1
Contour Map of verification test with 1 thread

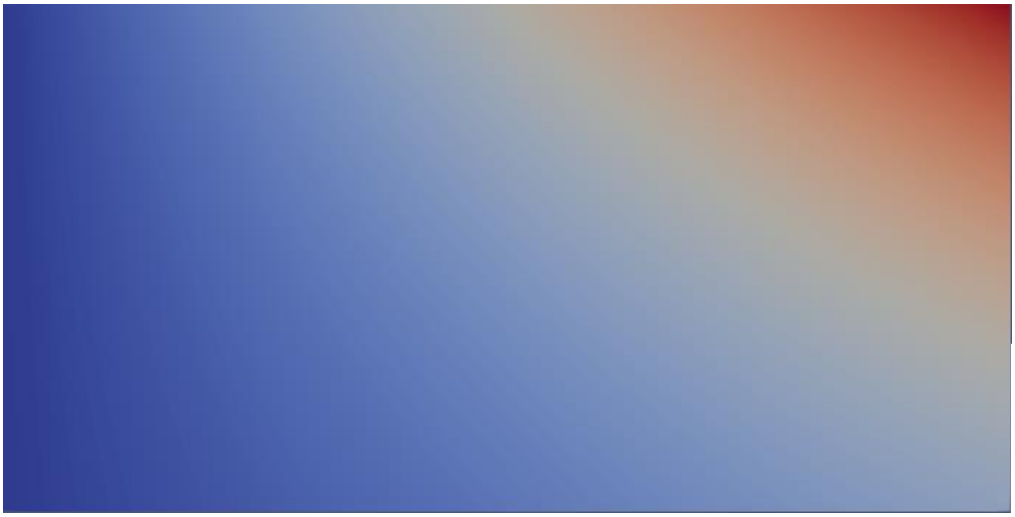
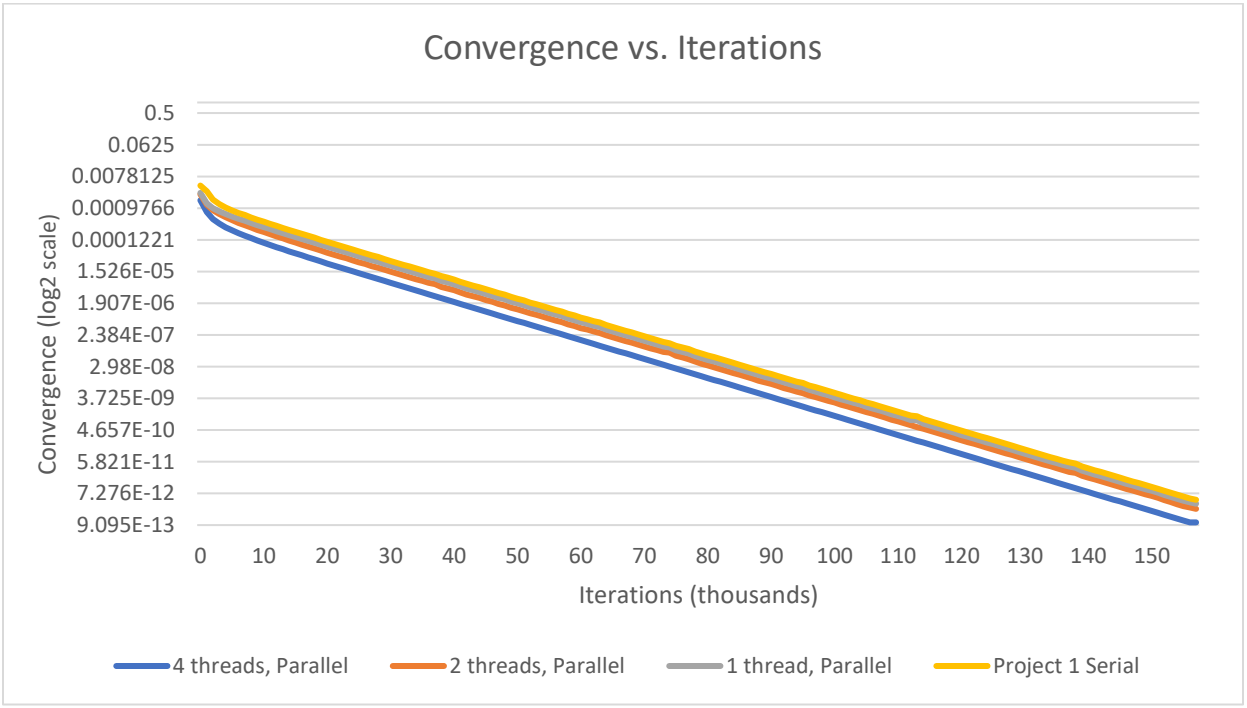


Figure 2
Contour Map of verification test with 4 threads

Convergence Analysis

The convergence of the algorithm was analyzed. On a grid size of 200x200, the simulation is ran in a serial manner, parallelized with one thread, parallelized with two threads, and parallelized with four threads. A log scale was used for the convergence.



As one can see, the program's convergence does not depend on the number of threads used for the program. This is because only the Jacobi algorithm is implemented, so every iteration T_k depends solely on the previous iteration's T_{k-1} values. Additionally, one will notice the convergence values were converted to a $\log_2$ scale, so the matrix values do converge quicker at the beginning of the simulation and much slower toward the end, as indicated by the linear line discovered by the transformation. To further explain this phenomenon, doubling the number of

threads theoretically halves the speed of convergence, as well as halving the importance of convergence because the difference of convergence from 10 to 20 iterations and convergence values of 2.0×10^{-3} and 1.0×10^{-3} is much more significant than shifting from 50k to 100k iterations and convergence values of 2.0×10^{-9} and 1.0×10^{-9}.

Performance

As grid size increases, so did the time taken to solve Poisson's equation. The grid sizes were pushed to significant and experimentally viable sizes.

When the simulation is ran with 2 threads, the partitioning scheme is 1 thread in the x-dimension and 2 processes in the y-dimension. When 4 threads are used, 2 threads are placed in each direction.

All simulations were ran on the University of Pittsburgh's Center for Research Computing (CRC). All times were recorded using C++'s ctime.h library with the clock_t structures. Only the time spent on solving the equation is monitored; setup and error calculation are not considered in the measurements. All times are averaged across five simulations to prevent outliers from affecting measurements.

Grid Size	Time (seconds)	Threads
50x50	1.4900	1
50x50	0.8266	2
50x50	0.5816	4
100x100	23.7700	1
100x100	12.7966	2
100x100	7.8100	4
200x200	362.7900	1
200x200	187.8150	2
200x200	112.8350	4

Clearly, converting this problem into a multi-threading problem drastically reduces computational time. As the number of threads increases, the average runtime across the processes decreases. Even more precisely, when the number of threads doubles, the runtime almost halves. As such, the runtime decreases almost proportionally to the increase in threads.

Algorithm Presentation

Clearly, the project handout contains a brief, general, vague description of the algorithm and Poisson equation. After further investigation, the Jacobi algorithm was implemented for this project to solve any solution given the source term, domain constraints, and boundary conditions.

As noted in the handout, the second derivative can be used to solve the true values of the solution. It turns out, the following equation, after solving for $T_{i,j}$, can be used to approximate the desired value:

$$(T_{i-1,j} - 2T_{i,j} + T_{i+1,j}) / (\Delta x)^2 + (T_{i,j-1} - 2T_{i,j} + T_{i,j+1}) / (\Delta y)^2 = S_{i,j}$$

Which ends up being:

$$[\ (\Delta y)^2 T_{i-1,\,j} + (\Delta y)^2 T_{i+1,\,j} + (\Delta x)^2 T_{i,\,j-1} + (\Delta x)^2 T_{i,\,j+1} - (\Delta y)^2\,(\Delta x)^2 S_{i,\,j}\ ] \,/\, (2(\Delta y)^2 + 2(\Delta x)^2) = T_{i,\,j}$$

So, this above equation was used to approximate the value of $T_{i,\,j}$ on each iteration, based on its neighbors and underlying source term, which was always provided for all coordinates.

An important note is this algorithm does not iterate over the boundaries because they are important boundary conditions to keep the algorithm stable. As such, $T_{i,\,j}$ is only a concern for all interior points of the grid.

One important challenge in the implementation of this algorithm, since it is utilizes threads, is to ensure each thread maintains an MxN matrix containing only that thread's data and ensuring some threads iterate over all M values and others iterate over M-1 values. For example, in a four thread scheme, thread 0 possesses the overall top and left boundary values of the matrix. Thus, when updating values for the Jacobi algorithm, for loops must iterate from 1…M and 1…N, but thread 1, which possesses the top and right boundary values, iterates from 1…M and 0…N-1. Finally, thread 2, with the left and bottom boundary values, iterates from 0…M-1 and 1…N. Maintaining the integrity of these boundary conditions across any scheme and any number of threads is imperative or the algorithm is at risk of never converging with poor values, thus producing high error.

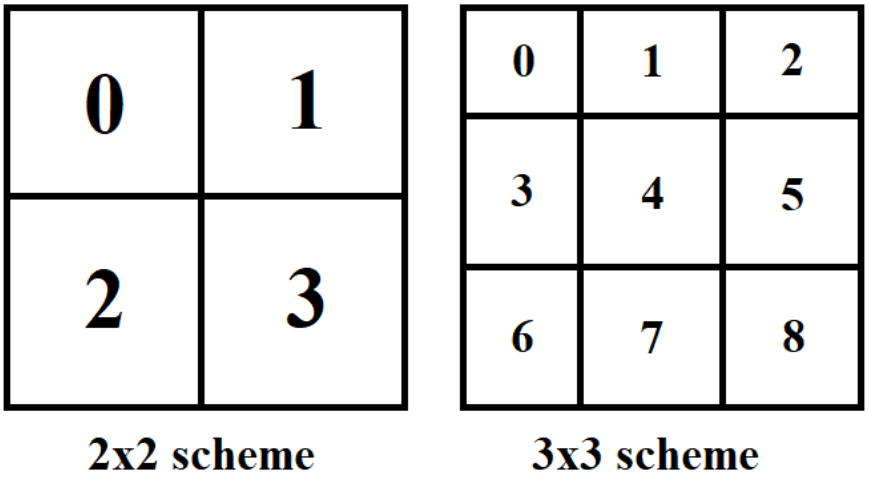


Figure 4
Contour Map of verification test with 1 thread

In the end, to find the error between the 'solved' grid and the 'true' grid, given in the verification test, we calculated the L-∞ norm (maximum difference between 'solved' value and 'true' value in the grid).

Satisfactory results were generated, as the grid both converged and produced useful contour maps related to the underlying solution values.