CUDA vs. Poisson

Carlos Vazquez Gomez

April 24, 2019

1 Algorithm

I use a Jacobi update loop using $\lfloor P/1000 \rfloor$ blocks with 1000 threads each, where P is the number of points on the grid. Originally I wanted to use whole-grid synchronization with Cooperative Kernels, but this turned out to be the wrong direction, as they only allow as many threads as can be concurrently resident on the grid to be launched (which would have been a 200x280 grid maximum, or 56,000 threads on the GTX 1080).

The algorithm is quite simple, consisting of only four kernels. The first one, `prepare_grids`, is launched once at the beginning with the purpose of preparing the grids with boundary conditions and source terms. Then, `update_temporary` and `update_real` stage and update the T temperature array with a call to `cudaDeviceSynchronize` in between to prevent race conditions. Finally, every 20,000 iterations, `get_error` uses atomic adds to calculate the L2-normed error.

2 Verification

Now we verify that our code performs accurately by testing its second order accuracy (the absolute error squared should be proportional to the step size squared). We also test whether or not we can solve on heterogeneous grids, for example 200x300 or 50x500. [h]

2.1 Second Order Accuracy

Our algorithm is shown to be approximately second order accurate by plotting both the square of $1/G_x$, where $G)x$ is the grid points per dimension, and the square of the absolute infinity normed final error. The results are shown in Figure 1. Both curves follow each other approximately (off by a constant multiple), leading us to believe that they are linearly proportional.

2.2 Heterogeneous Divisions

Figure 2 shows a Paraview rendering of the solution to a 200x300 grid, showing the capability of solving heterogeneous grids. Additional proof is evident from

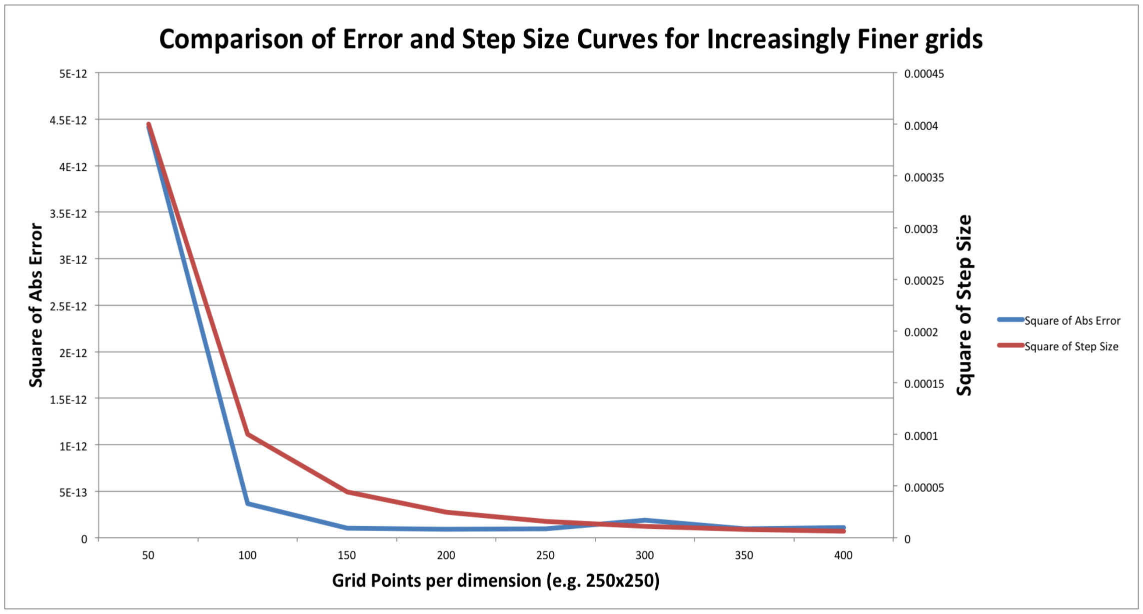

Figure 1: Ours is a second order accurate algorithm

Table 1, where the absolute errors of multiple similarly sized grids are compared.

Grid	Absolute Error
20x100	1.79e-04
40x80	3.82e-05
60x60	1.52e-06
80x40	4.10e-05
100x20	2.29e-05

Table 1: Solving on heterogeneous grids yields similar errors to solving on a homogeneous grid of 60x60

3 Performance

Finally, we compare the speeds of all of our implementations of the Poisson 2D solvers. We summarize the results in Table 2. I estimated the times for the serial and MPI implementations by calculating portions of the problem, realizing how many cycles it would take to converge (gathered from the GPU and implementation openMP runs), and extending the time. We conclude that CUDA or OpenMP are really the only practical options if faced with solving

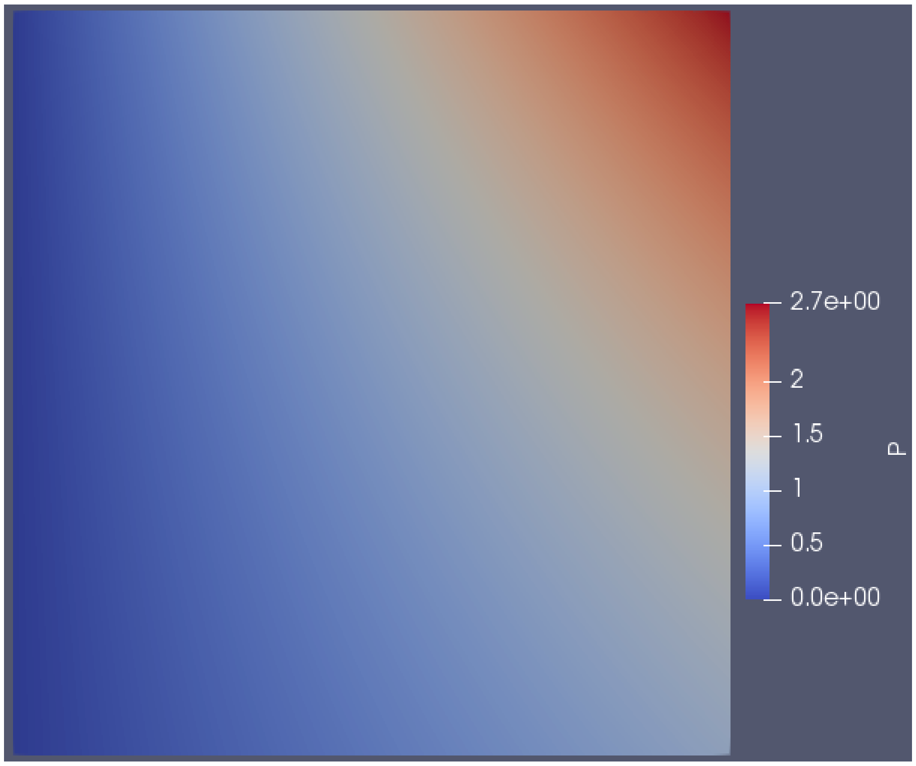

Figure 2: 200x300 grid result in ParaView evaulated using CUDA, showing correct evaluation of heterogeneous grid

Implementation	Total Time
Serial (P1)	14.4 hours (estimated)
MPI (P2)	1.2 hours (estimated)
OpenMP (P3)	34.5 minutes
CUDA (P4)	72 seconds

Table 2: Performance of various levels parallelism for solving a 500x500 grid

such a large physics problem as a 500x500 grid.