# Homework 4
# High Performance Computing

Evan Toler

April 20, 2020

Repository: `https://github.com/evantoler/HPC-S2020.git`

1. **Matrix-vector operations on a GPU.** Write CUDA code for an inner product between two given (long) vectors on a GPU. Then, generalize this code to implement a matrix-vector multiplication (no blocking needed here) on the GPU. Check the correctness of your implementation by performing the same computation on the CPU and compare them. Report the memory band your code obtains on different GPUs.

**Solution:** Our CUDA dot product and mat-vec implementation is the file `MMult-gpu.cu`. We model our approach on the reduction principles from Lecture 10, and in particular the example `gpu16.cu`. We check accuracy using our parallel matrix multiplication code from Homework 2.

To test our bandwidth, we generate a random matrix $A \in \mathbb{R}^{N \times N}$, with $N = 2^{10}$. We also generate a random vector $x \in \mathbb{R}^N$ and compute the product $Ax$ using our CPU and GPU codes. Table 1 below summarizes the bandwidth on three of Courant's compute servers.

| Server | GPU | GB/s (CPU) | GB/s (GPU) |
|---|---|---|---|
| cuda1.cims.nyu.edu | Two GeForce GTX TITAN Black (6 GB memory each) | 1.35 | 0.1287 |
| cuda3.cims.nyu.edu | Two TITAN V (12 GB memory each) | 0.48 | 0.23 |
| cuda5.cims.nyu.edu | Two GeForce GTX TITAN Z (12 GB memory each) | 10.76 | 0.1473 |

Table 1: Bandwidth comparisons for matrix-vector multiplication.

Our GPU method has bandwidths orders of magnitude less than the CPU method. Perhaps there are memory inefficiencies in our code that slow down the GPU. On the other hand, matrix multiplication does not seem inherently promising to perform on a GPU. There are many loops and moving parts to keep track of in order to conform algorithms to the CUDA paradigm. When matrices are stored in the conventional column-major indexing, computing inner products with rows of $A$ involves non-sequential memory access that slows down the bandwidth.

2. **2D Jacobi method on a GPU.** Implement the 2D Jacobi method as discussed in the 2nd homework assignment using CUDA. Check the correctness of your implementation by performing the same computation on the CPU and compare them.

**Solution:** We solve the system $Au = f$ associated with the Laplace equation, as before. We set the dimension of the problem to be $N = 2^{10}$ and run the Jacobi method for 100 iterations on both the

CPU and the GPU. We repeat this computation 30 times, and then compute the average bandwidth of each method over those 30 trials. Table 2 summarizes our code performance on the compute servers.

| Server | GPU | GB/s (CPU) | GB/s (GPU) |
|---|---|---|---|
| cuda1.cims.nyu.edu | Two GeForce GTX TITAN Black (6 GB memory each) | 1.52 | 105.01 |
| cuda3.cims.nyu.edu | Two TITAN V (12 GB memory each) | 0.92 | 35.36 |
| cuda5.cims.nyu.edu | Two GeForce GTX TITAN Z (12 GB memory each) | 6.70 | 114.43 |

Table 2: Bandwidth comparisons for Jacobi iterations.

The Jacobi method aligns very naturally with GPU computing. Each thread on the GPU updates one solution element $u_j$, and the kernel to implement this is straightforward and concise. Correspondingly, our GPU code achieves bandwidths at orders of magnitude greater than with our CPU code.

Finally, for both the Mat-vec and Jacobi code, the cuda3 server performed slower than the other servers. This could have just been because the serer was very busy whenever I ran, or it could indicate that the TITAN V GPU architecture is not as well suited to implementing these algorithms.

3. **Update on final project.** Describe with a few sentences the status of your final project: What tasks that you formulated in the proposal have you worked on? Did you run into unforeseen problems?

**Solution:** We have implemented the Parareal algorithm and a parallel FFT code in one dimension. We are currently working on a PDE solver that incorporates these codes for the problems we mentioned in our proposal. Piecing these codes together has been tricky, since we need to be put together the format, assumptions, and paradigms of each component. We talked as a team about these difficulties and now have a clearer idea of how to address them moving forward.