CPEG655 Final Project

Joel Farquhar

**Jacobi Method**

**Introduction**

For this project, I decided to create a GPU implementation of the Jacobi Method in cuda. The Jacobi Method is an approximation algorithm which solves for x given the equation Ax = b. It begins with an initial guess for x and it iteratively improves the approximation until an arbitrary threshold is met (for the purpose of this project 25 iterations were performed). This algorithm is useful in cases where it is difficult to invert A or where speed is more important than accuracy (although the accuracy is still fairly good). This algorithm will converge in most cases, but in order to ensure it is guaranteed it is necessary that the diagonal values are greater than the sum of the other elements in the row.

**Code Structure**

In order to compare the running time of the CPU and GPU implementations, the code for these two approaches is contained in totally separate files. The src directory contains all of the source code for both methodologies, with the Jacobi.c, Jacobi.h, and main.c files in the src directory forming the CPU code, and JacobiRWD.cu, JacobiRWD.cuh, and mainRWD.cu representing the GPU code. This code also contains test benches for both implementations, with JacobiTests.c forming the test bench for the CPU code and JacobiRWDTests.cu forming the GPU test bench. The obj directory contains the object files corresponding to this source code, and the Makefile controls the compilation of the project, including the assigning of global variables like N. The compareScript.sh is a script which allows for the comparison of the Jacobi Method on both the CPU and GPU time.

**CPU vs GPU Approaches**

The main difference between the CPU and GPU approaches is that the CPU implementation is entirely sequential, while the GPU code is parallelized using cuda. As such, the GPU code uses multiple thread blocks with the number of blocks, the number of threads per block, and the number of elements a given thread works on determined by global parameters NB, NT, and NK. The setup of the functions also varies slightly, as the GPU code needed to be broken into host, device, and global functions. Since the cuda functions divide the work amongst the threads, their formulation is also different so that each thread does just its work and does not interfere with the computation of other threads. In the end the result is the same between the two approaches, they just come up with the answer differently. Although this does not affect the functionality, the current implementations both do not free memory and as such have several memory leaks. In the future, I would like to fix this and use a tool such as valgrind to ensure that no memory leaks exist.

**Test Benches**

In order to ensure correctness and allow for fine grained comparison between the two approaches, test benches were made which check the running time and correctness of method calls. Although there is room for expansion and not every method is checked, this test bench made debugging the overall process easier, as it was often difficult to figure out which part of the Jacobi Method process was breaking. It also helped ensure that the methods functioned correctly under varying input parameters for N, NB, NT, and NK. In the future, I would like to expand these test benches to check more functions, cover more edge cases, and to improve individual components of the Jacobi Method process such as matrix addition or subtraction.

**Results**

The exact speedup of the GPU approach as compared to the CPU approach varies as random numbers are used, but on average the GPU approach was about 1.8x faster than the CPU approach. This is what parameter values selected based off the findalOptimalParamsScript, although it was only run once. Given extra time, I would rerun this script multiple times and average the running times for each in order to see which worked the best on average, rather than from a single test. This result could also be improved significantly with extra tinkering in each of the sub methods in order to ensure that they each work as quickly as possible.