Casey Adams, Kevin Connell
ECEC 622
Project 8 - CUDA Gaussian Elimination
3/17/2021

CUDA: Gaussian Elimination

Implementation of CUDA

The gaussian elimination approach used here works in two parts. The first part is division and the second part is elimination. This was written as two separate kernels in cuda but was later combined for the sake of simplicity. However, inside the kernel the two sets of code are clearly divided. The first part iterates through the active row and divides all elements by the value of the pivot element which is at position (k,k). This results in the pivot being a value of 1. This process can be parallelized across the row to reduce computation time. Once completed, the threads synchronize and the elimination step starts. In this step, each thread gets a separate row and the value of the pivot column of that row is divided into the value of the pivot at (k,k). This is used as a factor which is multiplied by the value of the pivot row and the result is subtracted from each of the remaining rows in the matrix. This ensures all elements below the pivot are 0. This process is repeated for each row/col in the matrix.

Performance

Table 1 represents the difference in Matrix size speeds across both the GPU and CPU.

Table 1

	512x512	1024x1024	2048x2048	4096x4096
CPU	0.0578s	0.464s	3.887s	32.531s
GPU	0.0996s	0.788s	4.317s	18.212s

As can be seen, the CPU outperforms the GPU in most of the smaller matrix sizes. This is in part because the task is not as parallelizable as some others. There are situations where some rows are longer than others and so some threads have significantly more work to do than others. This means a lot of threads are left idle. Additionally, other groups were using the xunil server cluster at the same time which could have significantly reduced the GPU performance. Despite this, the GPU did manage to perform faster than the CPU once a matrix size of 4096 was reached.

Conclusion

Parallelization using CUDA is capable of speeding up the processing time for gaussian elimination given a large enough matrix size. At smaller matrix sizes, the CPU still outperforms the GPU. This is in part because some threads of the GPU are left idle as they have a significantly shorter row to operate on. In theory this could be sped up by having threads run independent of a given row however this would significantly increase code complexity.