Casey Brooks

CSCE 435

HW 6

**Explanation of Changes**

There were a couple changes I made to the code to enable much faster execution in parallel on the GPU. There were two strategies I employed, one of which turned out to be far faster than the other.

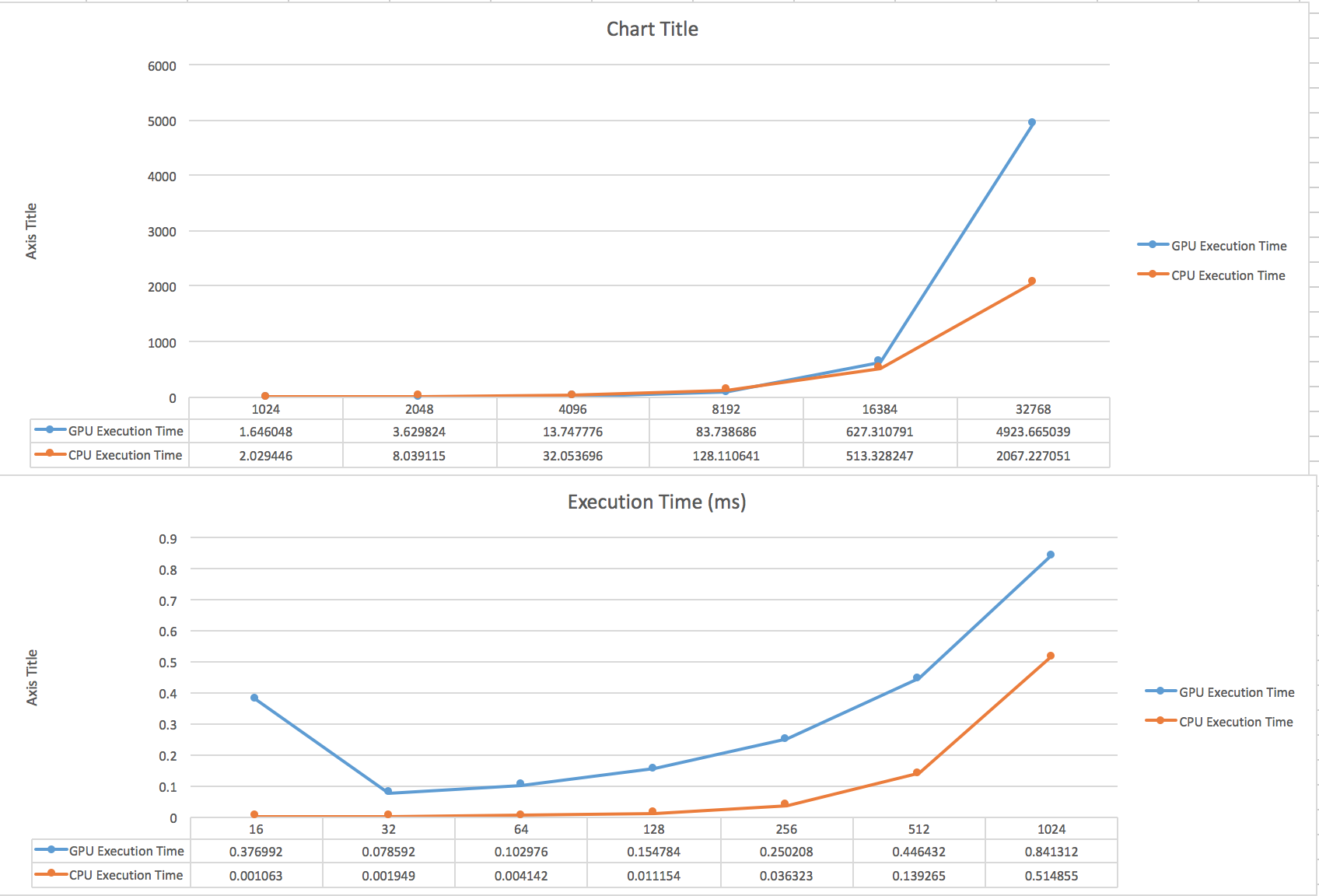
The first strategy was to create a grid of blocks, where each thread defined exactly one pair of points. Each thread would calculate the distance between two points, then atomically update the value D with the min of its calculated distance and the value of D. While this way is massively parallel, the need to synchronize access to D for every single pair of points made it very slow.

The second strategy was to remove just the outer loop of the brute-force algorithm in minimum\_distance\_host, and replace it with parallelism employed through the GPU. That is, every value of i will be taken by one thread on the GPU, which will calculate the minimum distance from the point at i to all other points j, and will only update the value D after it has completely finished.

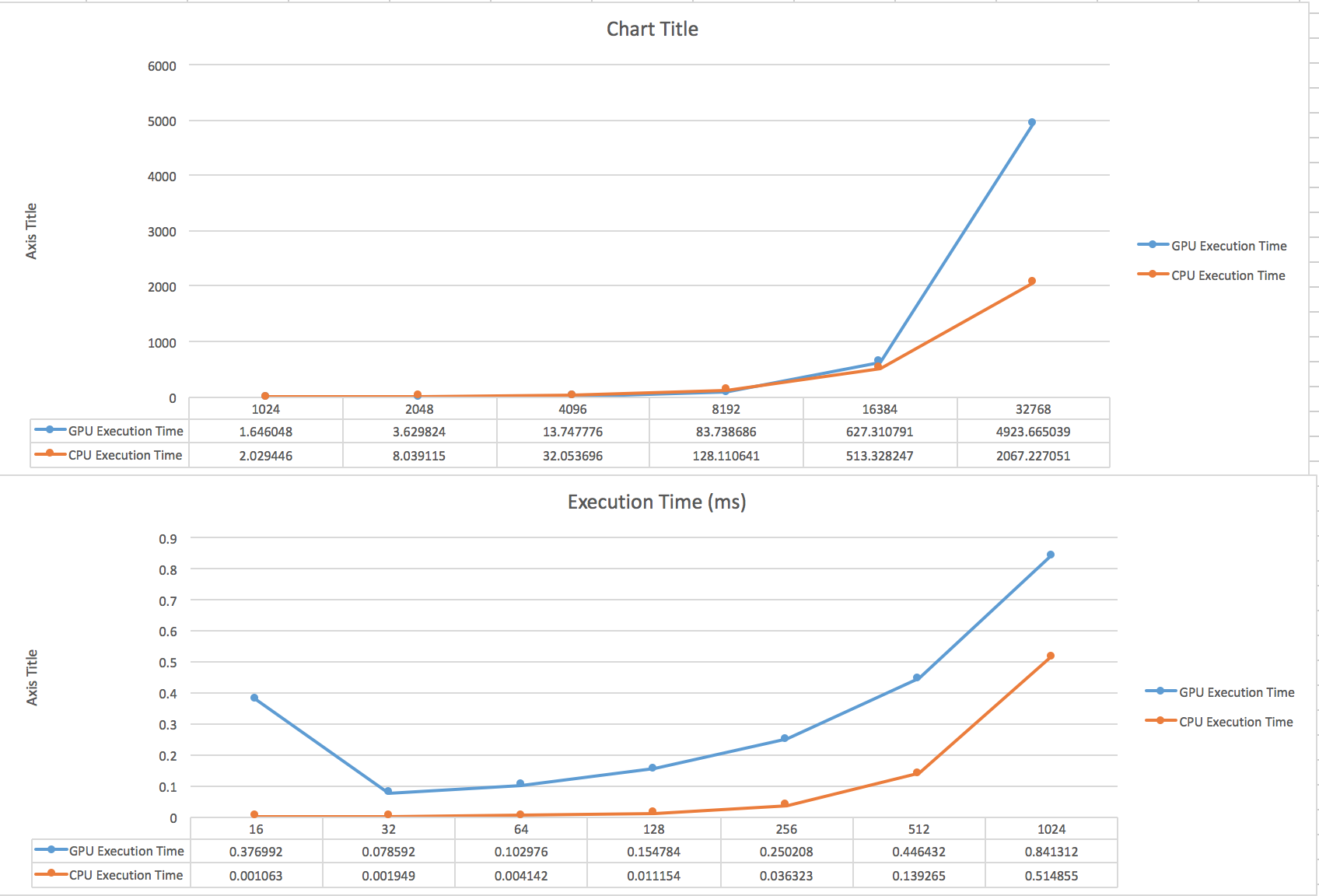
The i value of each thread is calculated by threadIdx + (blockIdx\*blockDim). I used a function for a atomic min that works with float values from <http://stackoverflow.com/questions/17399119/cant-we-use-atomic-operations-for-floating-point-variables-in-cuda> to synchronize the reading and writing of D to ensure that it maintains the correct values across all threads.

**Graphs**

*Execution Time (ms) n=4-10*



*Execution Time (ms) n=11-16*



*Data transfer time (ms)*

