HW3 - Intro GPU Programs

Aidan Brookes

Part 1 - Counting

An efficient algorithm for counting the number of matching values in a matrix is not trivial. While simply comparing all values of the matrix is straightforward and may be completed with the same or better speed compared to to a matrix copy, the step of summing these values requires requires a reduction which may be optimized significantly. Using the simplest approach (an atomic operation) takes about 60ms. Later implementing a similar shared memory strategy to the transpose did reduce the time to about 57 ms, but it is clear that the atomic counter is the most significant bottleneck. In order to improve the speed more I would try summing the results with an interleaved reduction strategy, but this is complicated by the huge variation in array sizes being tested.

Part 2 - Transpose

The decomposition strategy I used for transposing a matrix was to divide it into chunks of 32 by 32. When mapped to thread blocks, however, the blocks are given dimensions 32 x 8. This means that each thread will compute the transpose for 4 elements. This parameter may be tuned, and I found that 32x8 performed better than 16x8 and 64x16. This is likely due to the ratio between thread overhead and time computing indexes for each element. In order to reduce the number of accesses to global memory, I stored each chunk to a shared array whose value is contributed to by all threads in the block. This has the side effect of simplifying the index calculations. I found that this improvement results in an improvement from about 60ms to about 50ms.