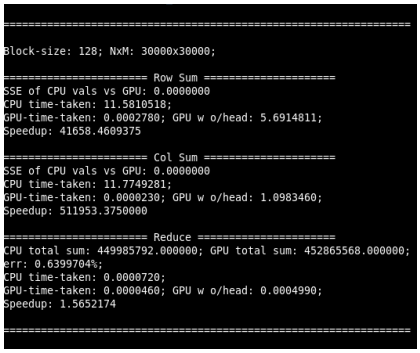
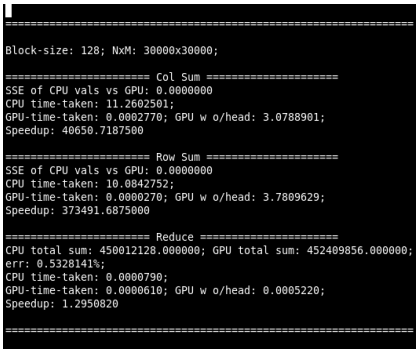


(a) Scrnshot of example output with row sum first.



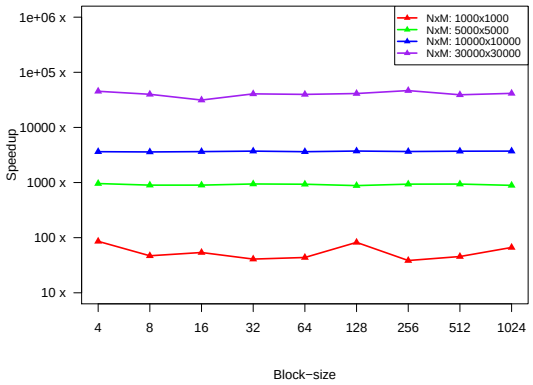
(b) Screenshot of example output with column sum first.

Figure 1: Example output screenshots.

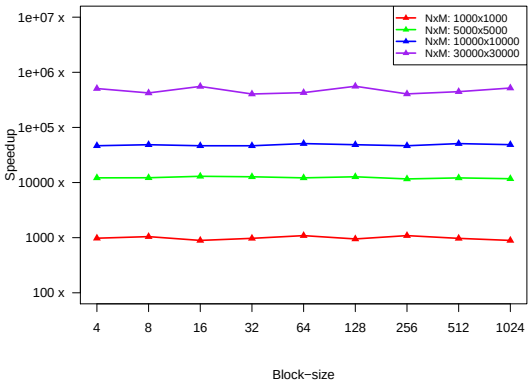
CPU vs. GPU times

Between the three functions written, as was expected, in the serial version of the code the row sum function runs faster than the column sum function as the matrices are written in row major format and so the data is contiguous for the row sum function, this more efficient. Naturally, the reduce function runs much faster than both of these as it has only to do $\mathcal{O}(N)$ operations as opposed to $\mathcal{O}(N \times M)$ seen in the earlier two functions. These times can be seen in an example output screenshot seen in Figure 1a.

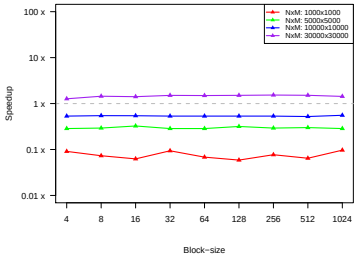
The GPU times, on the other hand, did not perform quite exactly as expected. One would expect the times for the row-sum to be faster than the column sum in the GPU's case also, however, they actually performed slower. This was tested and did not appear to be related to the algorithm but actually moreso related to whichever function was run first. Figure 1b shows anoterh output screenshot where the column sum function was ran first. It can be seen to run slower this time, implying that this is possibly due to some hardware problem of the GPU needing to "warm-up" for lack of a better term. Thankfully, the speedups seen did behave as one would expect. However, the difference in speedup between the row sum and column sum functions can be taken with a grain of salt as by interchanging the order within which they are executed will effectively swap their speedup times. Figure 2 shows the substantial speedups seen in the first two functions getting up to orders of $10^6\times$. As the size of the problem increases the speedup also as expected increases. The reduce function only actually appears to see speedups at $N \times M$ reaching $30,000 \times 30,000$, presumably due to the need for multiple thread synchronisations compared to the one sync in the othe



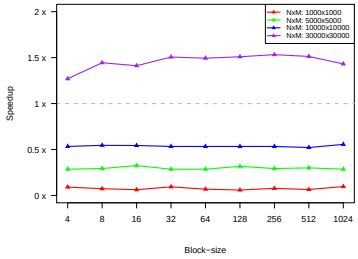
(a) Log scale of speedupds for row sum.



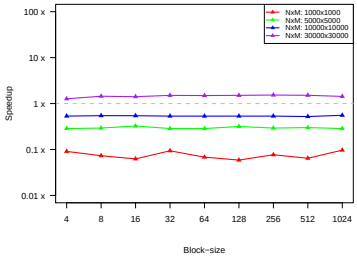
(b) Log scale of speedupds for column sum.



(c) Log scale of speedupds for reduce.

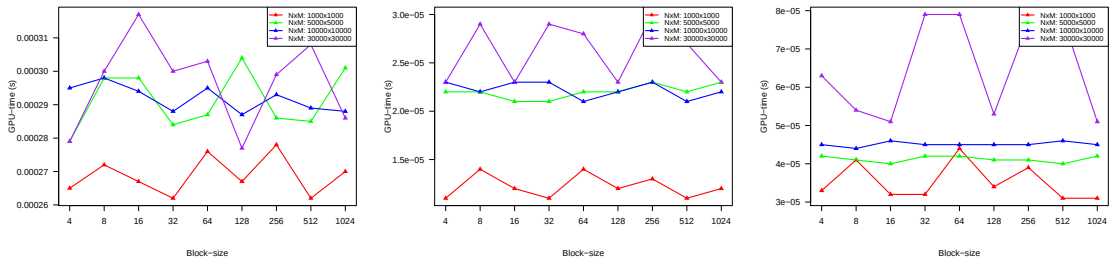


(d) Absolute value of speedups for reduce.



(e) Absolute times of reduce for CPU & GPU.

Figure 2: Function speedups of GPU vs. CPU.



(a) Absolute times for row sum. (b) Absolute times for column sum. (c) Absolute times for reduce.

Figure 3: Absolute times take on GPU.

other two functions. Figure 3 shows the absolute times of the GPU execution of each function for each of the matrix sizes, versus the block size in number-of-threads. It is not clear from any of those results nor from the speedups that there is any evident optimal block size for this problem, and so this ended up being inconclusive. The absolute times taken for the GPU are seen to be quite erratic and no real pattern can be seen in them. Rather strangely, in some of the case that can be seen, the larger problem size takes less time than the smaller.

Errors

In order to evaluate the accuracy of the software, an SSE calculation was performed on the resulting vectors from the row sum and column sum functions between the CPU returned values and GPU returned values. In all cases, the resulting SSE was 0.0 to 7 decimal places. The reduce function, a percentage difference was found between the CPU and GPU returned values. In most cases, barring in Figure 4 showing 3 outliers, all errors were of the order of $\sim 10^{-5}$ percent. However, again in the accuracy tests, it does not appear clear that there is any evident optimal choice of block size.

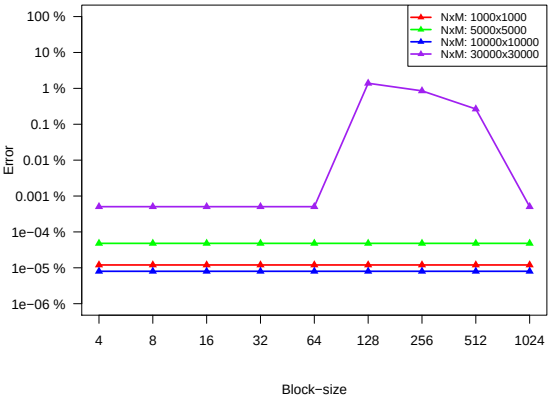
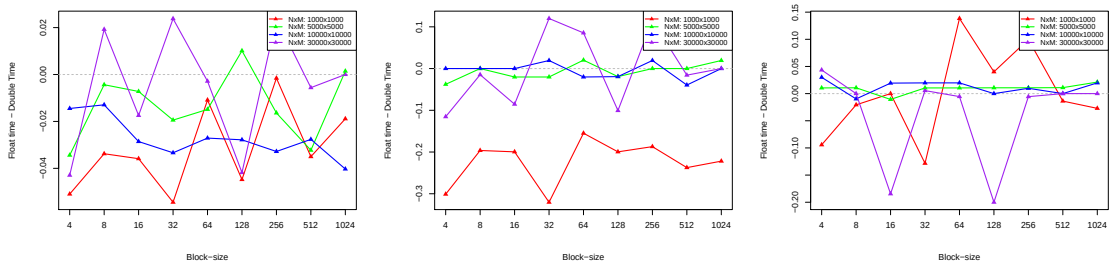


Figure 4: Percentage error between CPU and GPU final values for reduce.



(a) Row sum float time - double time.

(b) Column sum float time - double time.

(c) Row sum float time - double time.

Figure 5: Difference in time taken using floats or doubles.

Doubles vs. Floats

Again. some erratic results were received for some of the tests in this program. One would expect that the time taken to process floats on the GPU would run universally slower than processing float. However, as can be seen in Figure 5, there is no consistent result to say that doubles run slower than floats for any block size of matrix size.