Execution time results

Even Flørenæs

TDT 4200

Parallel Computing

Fall 2016

Execution results using cudaEvent to time kernel:

Block sizes	Global memory	Shared memory	Texture memory
X: 2 , Y: 2	min,max,avg: 6.05 ms,9.36 ms ,6.10 ms	min,max,avg: 14.79 ms,18.36 ms,14.87 ms	min,max,avg: 22.08 ms, 34.34 ms, 22.83 ms
X: 4, Y: 4	min,max,avg: 1.83 ms, 3.13 ms,1.86 ms	min,max,avg: 4.02 ms, 6,04 ms, 4.06 ms	min,max,avg: 5.54 ms, 9.57 ms, 5.81 ms
X: 8 , Y: 8	min,max,avg: 1.18 ms, 2.61 ms, 1.20 ms	min,max,avg: 1.22 ms, 2.58 ms,1.25 ms	min,max,avg: 2.77 ms, 4.77 ms, 2.94 ms
X: 16, Y: 16	min,max,avg: 0.80 ms, 0.88 ms, 0.82 ms	min,max,avg: 1.17 ms, 2.49 ms, 1.20 ms	min,max,avg: 2.79 ms, 4.75 ms, 2.86 ms
X: 32, Y: 32	min,max,avg: 0.78 ms, 0.93 ms, 0.79 ms	min,max,avg: 1.32 ms, 2.69 ms, 1.35 ms	min,max,avg: 2.96 ms, 4.94 ms, 3.12 ms

As can be seen in the table above, the implementation using only global memory is faster than both the shared memory and texture memory implementation. The shared memory implementation is faster than the texture memory, leaving the texture memory as the slowest implementation. Initially I would had thought the results would be different. Shared memory is an on-chip memory with higher bandwidth and lower latency compared to the global memory. But there are several consideration which can limit this effect.

The benefit of shared memory compared to global will not be present if there are shared memory bank conflicts in this implementation. Also the implementation will have to synchronise all threads inside the kernel to be able to read values of the shared memory array which will possibly lead to latency issues. The large benefit of shared memory is the low cost of read if we read the same values multiple times. If we read many of the same elements for multiple threads reading from the shared memory compared to the global, it will be faster. But in this implementation we place at least one element of the global memory in the shared memory (more than one for border elements), and read 5 times from the shared memory. The ratio between adding values to shared memory and number of times reading from it, leaves this implementation not very beneficial of using shared memory.

The texture memory implementation is the slowest of them all. Texture memory can help exploit spatial locality compared to global memory, suggesting that we would benefit from using the texture implementation, but the timing results shows this is not true. The architecture used to time the implementation architecture of the GTX 750 is (according to the white paper) build optimise the L1 and L2 cache, which could suggest that the benefit of using texture cache would be reduced. Also my implementation never reuses any values in the texture memory, which leads to not exploiting the benefit of the texture cache. Also the texture cache can experience latency overhead unlike L1/L2 cache which could possibly explain some of the slowdown.