Universität Heidelberg
Institute for Computer Engineering (ZITI)

Master of Science Computer Engineering

GPU Computing

Exercise 5

Group gpu04
Pingitzer, Danny
Altuntop, Ekrem
Junge, Andreas G.

Due date November 27, 2019

5 Exercise

5.1 Reading

Read the following paper and provide review as explained in the first lecture (see slides):

John Nickolls and William J. Dally. 2010. The GPU Computing Era. IEEE Micro 30, 2 (March 2010), 56-69.:

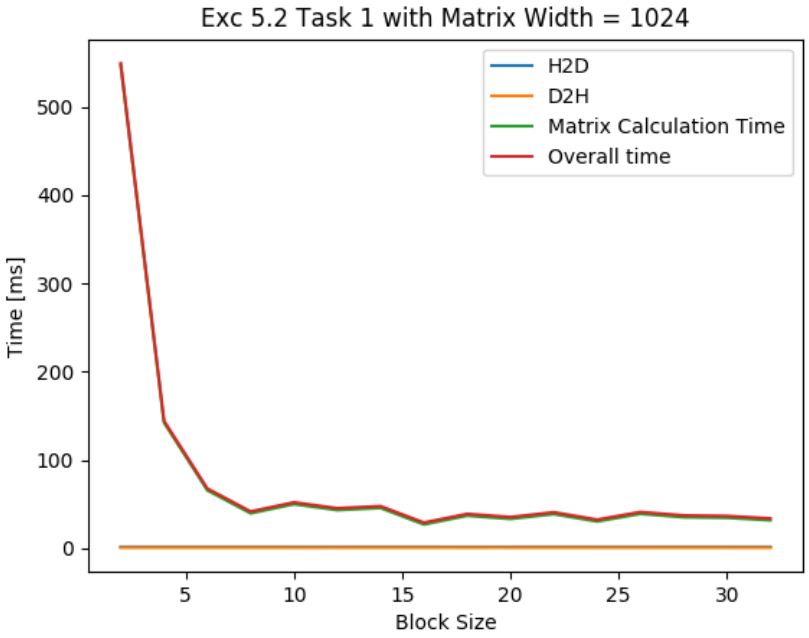
@todo: Text by Ekrem.

5.2 Matrix Multiply – GPU naïve version

Vary the threads-per-block (blocksize) parameter to determine an optimal value. Choose this parameter and report the overall runtime (sum of a,b,c) together with the different components (a,b,c) by varying the problem size.

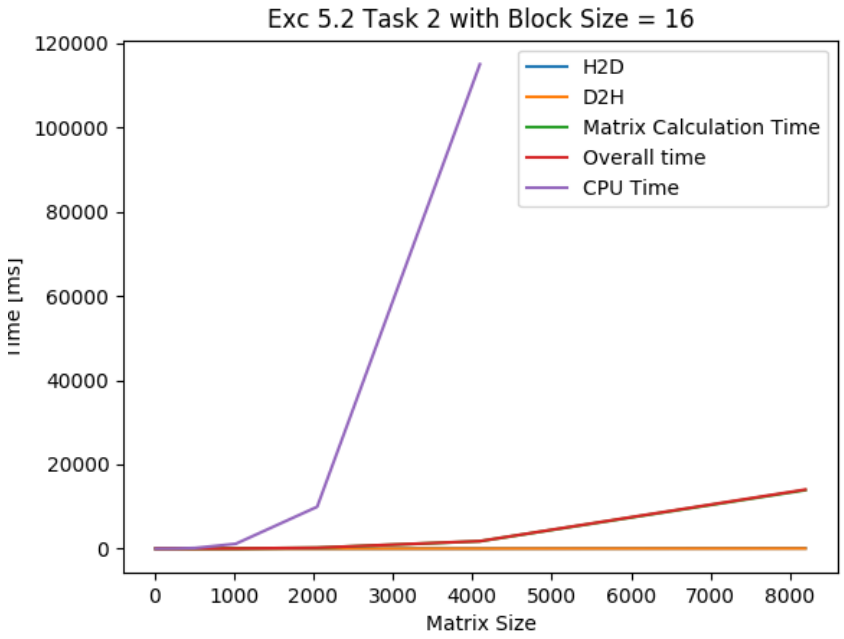
We varied the threads-per-block parameter at the constant problem size of 1024. We found, that the optimal thread-per-block count was: 16, with an average of 26.9205 ms per Matrix Multiplication.
We did 10 iterations per measurement point and divided the total runtime by this, in order to have accurate results (more iterations would have been better, but would have a unacceptable runtime).

Figure 1: Block size from 2...32 in stepsize 2



With this threads-per-block number of 16, we varied the problem size (matrix size) and also compared it to sequential CPU execution time:

Figure 2: Matrix sizes: 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192



Report the highest speed-up you achieved compared to the CPU version from 5.2, both with and without data movements.
On the CPU we could only measure up to a matrix size of 4096, everything above took too much runtime.
For matrix size 4096 we have:

- CPU Runtime: 115026 ms
- GPU H2D + D2H movement time: 32.6053 ms
- GPU Runtime: 1755.21 ms

This is equivalent to a speedup of factor ca. 65,5 without data movements and a speedup of factor ca. 64,3 with consideration of data movement time.
This ratio will increase with larger matrix size.

5.3 Matrix Multiply – GPU version using shared memory

Choose a suitable problem size (rather large) and vary the threads-per-block parameter to determine an optimal value. Report run time graphically.
We chose a problem size of 1024 and 2048 to compare and check.
We found that 32 threads-per-block are optimal, with an average execution time of 19.2625 ms (for Problem Size 1024) and 151.929 ms respectively (for Problem Size 2048).

Figure 3: Block sizes from 2...32 in stepsize 2

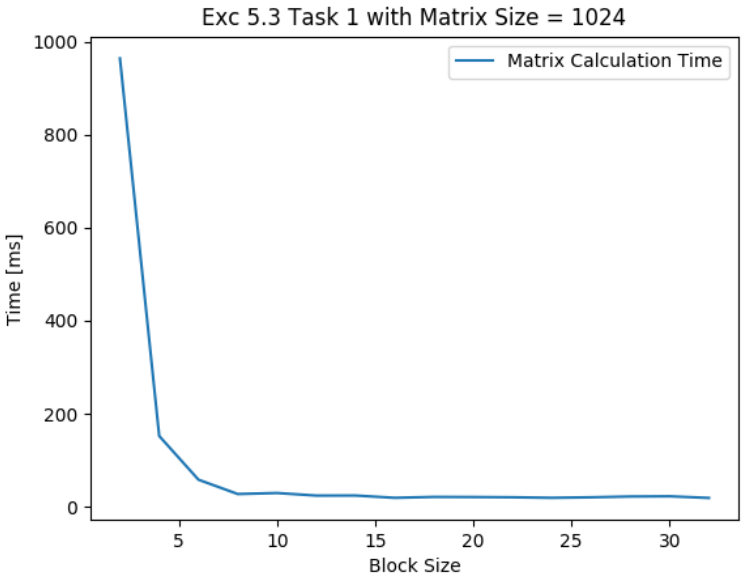
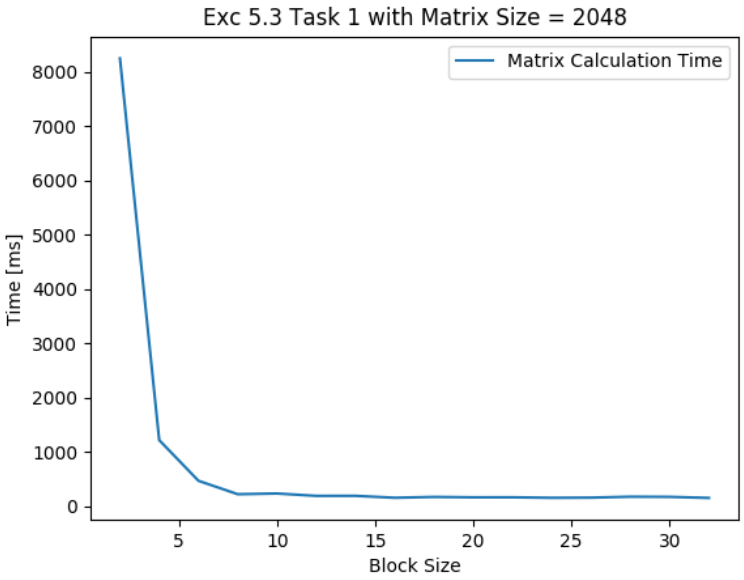
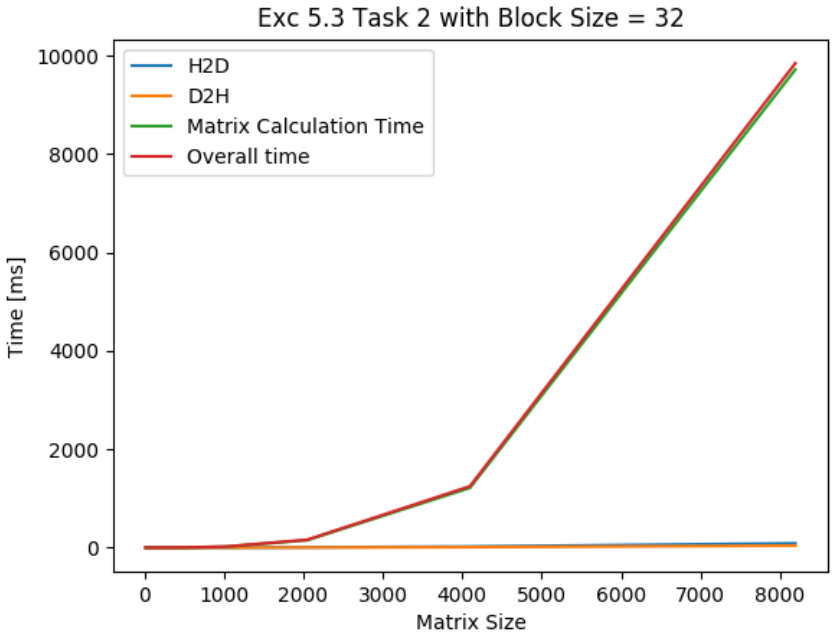


Figure 4: Block sizes from 2...32 in stepsize 2



Report the overall runtime (sum of a-c) together with the different components (a-c) by varying the problem size.



Report the highest speed-up you achieve compared to the two previous versions, both with and without data movements.
On the CPU we could only measure up to a matrix size of 4096, everything above took too much runtime.
For matrix size 4096 we have:

- CPU Runtime: 115026 ms
- GPU H2D + D2H movement time: ca. 32 ms
- GPU Runtime (no shared memory): 1755.21 ms
- GPU Runtime (shared memory): 1214.37 ms

Speedup of shared memory version over the CPU version without data movements: 94,7 and with consideration of data movements: 92,2. Speedup of shared memory version over the GPU non shared memory version: 1,4