Heidelberg University
Institute of Computer Engineering
Computing Systems Group

Holger Fröning
Hendrik Borras

GPU Computing
Winter Term 2020/2021

Exercise 5

- **Return electronically until Tuesday, Dec 15, 2020, 09:00**
- **Include your names on the top sheet. Hand in only one PDF.**
- **A maximum of four students are allowed to work jointly on the exercises.**
- **Hand in source code if the exercise required programming. You can bundle the source code along with the PDF in a .zip file.**
- **Programming exercises can only be graded if they compile on the cluster head node. Make sure to document the commands which you used for the compilation.**

Perform tests with at least 10 measurement points and report graphically. Report the run time (usec, sec, ...) on the y-axis and the size on the x-axis. If another parameter is required (threads per block, etc), use different lines and a legend.

5.1 Reading

Read the following two papers and provide reviews 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.

(5 points)

5.2 Matrix Multiply – GPU naïve version

Implement a CUDA version of the matrix multiply operation. Make use of thread blocks to support arbitrary problem sizes (as long as they fit into device memory). The program should accept the matrix size (element per a dimension) and the number of thread per block as parameters. It should report the run times for (a) host-to-device data transfer, (b) kernel execution, and (c) device-to-host data transfer. Don't include initialization time in the measurements. As you can assume that the matrices are normally initialized by some previous calculation.

- Ensure correctness by comparing your results to the results of the CPU version. Note: it is helpful to do this automatically for each run; however long run times for the CPU version might be prohibitive.
- Vary the threads-per-block 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.
- Report the highest speed-up you achieve compared to the CPU version from 4.4, both with and without data movements.

(12 points)

Holger Fröning
Hendrik Borras

5.3 Matrix Multiply – GPU version using shared memory

Extend the previous code to use the shared memory of each SM to leverage data re-use as explained in the lecture. Allocate shared memory only dynamically, static allocations are not allowed.

- Ensure correctness by comparing your results to the results of the CPU version.
- Choose a suitable problem size (rather large) and vary the threads-per-block parameter to determine an optimal value. Report run time graphically.
- 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.

(18 points)

5.4 Willingness to present

Please declare whether you are willing to present any of the previous exercises.
The declaration can be made on a per-exercise basis. Each declaration corresponds to 50% of the exercise points. You can only declare your willingness to present exercises for which you also hand in a solution. If no willingness to present is declared you may still be required to present an exercise for which your group has handed in a solution. This may happen if as example nobody else has declared their willingness to present.

- Reading (Task: 5.1)
- Matrix Multiply – GPU naïve version (Task: 5.2)
- Matrix Multiply – GPU version using shared memory (Task: 5.3)

(18 points)

Total: 53 points