Heidelberg University
Institute of Computer Engineering
Computer Engineering Group

Holger Fröning
Bálint Soproni

GPU Computing - Architecture and Programming
Winter Term 2023/2024

Exercise 6

- **Hand in via Moodle until Monday, Dec 11, 2023, 09:00**
- **Include all names on the top sheet. Hand in a single PDF.**
- **Please compress your results into a single archive (`.zip` or `.tar.gz`).**
- **Please employ the following naming convention `hpc<XX>_ex<N>.{zip,tar.gz}`, where `XX` is your group number, and `N` is the number of the current exercise.**
- **A maximum of four students is allowed to collaborate on the exercises.**
- **In case an exercise requires programming:**
  - **include clean and documented code**
  - **include a Makefile for compiling**

General notes

Perform tests with at least 10 measurement points and report graphically. Report the sustained bandwidth (GB/s, ...) on the y-axis and the array size on the x-axis. If another parameter is required (threads per block, etc), use different lines and a legend.

Please use Pascal Thread Scheduling for exercise **??** and **??** to avoid unwanted behavior by setting -arch=compute_60 -code=sm_70 compiler flags [1].

6.1 Reading

- Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 4 (April 2009), 65-76.

(5 points)

6.2 Reduction – CPU sequential version

Implement an unoptimized, sequential CPU version of a global sum reduction. Your program should accept the array size as a parameter and should report run time and the bandwidth for this operation (no allocations or initializations).

- Report the bandwidth graphically by varying the problem size. Interpret your results.

(5 points)

6.3 Reduction – GPU parallel initial version

Implement a CUDA version of the global sum reduction. Make use of thread blocks to support arbitrary problem sizes (as long as they fit into device memory). The program should accept the array size and the block size as parameters. It should report the run time and bandwidth for the kernel execution. Don't include initialization or data movements in these measurements. Ensure that this code uses only two iterations (i.e. the second iteration starts one thread block with as many threads as the first iteration had blocks). Do not perform additional optimizations here, the main goal is to verify correctness by comparing to the CPU code and obtain initial data for later performance comparisons.

- Ensure correctness by comparing your results to the results of the CPU version.
- Report sustained bandwidth over different array sizes (x-axis) and vary the block size (different lines).

(8 points)

[1] `https://developer.nvidia.com/blog/using-cuda-warp-level-primitives/`

Holger Fröning
Bálint Soproni

6.4 Reduction – GPU parallel optimized version

Start with the code developed in **??**. The goal here is to apply one or multiple possible optimization, i.e. the only objective is to improve the performance of the reduction operation. The lecture provides useful hints and insights about possible optimizations.

- Ensure correctness by comparing your results to the results of the CPU version or the previous GPU version.
- Report sustained bandwidth over different array sizes (x-axis) and vary the block size (different lines).
- Compare sustained bandwidth over the CPU and initial GPU version.

(20 points)

6.5 Bonus: Reduction – GPU parallel optimized version with Volta Features

Start with the code developed in **??**. The goal here is to apply one or multiple possible optimization by using Volta features. Do not use the compiler directives -arch=compute_60 -code=sm_70. The lecture provides useful hints and insights about possible optimizations.
Alternatively, you can start with the code from **??**. In this case the maximum bonus points are 15.

- Ensure correctness by comparing your results to the results of the CPU version or the previous GPU version.
- Report sustained bandwidth over different array sizes (x-axis) and vary the block size (different lines).
- Compare sustained bandwidth over the CPU and GPU version.

(20 bonus)

6.6 Reduction – Profiling

In this task we are going to take a short look at the different ways in which one can profile CUDA applications using NVidia tools. In particular we are going to compare one of your reduction implementations against the CUDA Thrust implementation. This guarantees significant differences in your measurements. For all of the following measurements please choose a relatively large memory size, i.e. more than 100 MB. Then choose one of your reduction implementations and threads per block settings as your reference implementation. Apply the same settings to the Thrust implementation. Note that Thurst chooses the block and grid size automatically.
Where useful, you can include screenshots with your submission.
We will start with looking at the profiling results for both implementations with Nsight Systems. We start here, because Nsight Systems gives a high level overview of the whole application. You can download Nsight Systems here[2] and find documentation about the CLI here[3]. In order to download the application you will need to register an NVidia account, otherwise everything should be free.
To create the profiling reports on the cluster you can use the following command:

```
nsys profile -o <report_file> <path to application> <application parameters>
```

You can then transfer the generated reports from the cluster to your personal computer and open them with Nsight Systems.
Please answer the following questions:

- How large is the performance impact on the reduction implementations compared to not using the nsys profiler?
- From the general timings of the kernels and their launches, is there something that you can conclude for potential further optimizations of your code? And if so what?

[2] `https://developer.nvidia.com/nsight-systems`
[3] `https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profiling`

- You may notice something strange about the memory transfers when the Thrust implementation completes. What might be happening here? (Hint: Look at the implementation in the template and the Thurst documentation for the reduction[4].)

Now we will look at profiling both reduction implementations using Nsight Compute. Previously with Nsight Systems the kernels were shown simply as monolithic blocks without much further information of what happened on the GPU itself. In contrast to this Nsight Compute provides deeper insights into the performance statistics of a given kernel while running on the GPU. You can download Nsight Compute here[5] and find documentation about the CLI here[6].
Run your applications once with only the default profiling sections enabled:

```
ncu -o <report_file> <path to application> <application parameters>
```

And another time with all sections enabled:

```
ncu --section ".*" -o <report_file> <path to application> <application parameters>
```

You can then transfer the generated reports from the cluster to your personal computer and open them with Nsight Compute.
Please answer the following questions:

- How large is the performance impact on the programs in each profiling case (default and all sections)?
- What for differences can you observe by comparing the information in the "GPU Speed of Light", "Launch Statistics" and the "Memory Workload Analysis" sections? And what could be the reason for these differences? Note that the "Memory Workload Analysis" section is only contained in the profiling report, which recorded all sections.

Note: In order to make NSight Systems work on our cluster, you need to change the tmp directory:

```
export TMPDIR=$(mktemp -d /tmp/$USER-XXXX)
```

(20 points)

- Reading (Task: **??**)
- Reduction – CPU sequential version (Task: **??**)
- Reduction – GPU parallel initial version (Task: **??**)
- Reduction – GPU parallel optimized version (Task: **??**)
- Reduction – GPU parallel optimized version with Volta features (Task: **??**)
- Reduction – Profiling (Task: **??**)

(29 points + 10 bonus)

Total: 87 points + 30 bonus

[4] https://thrust.github.io/doc/group__reductions.html
[5] https://developer.nvidia.com/nsight-compute
[6] https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html