Universität Heidelberg
Institut für Technische Informatik

Robert Strzodka
Application Specific Computing

Advanced Parallel Algorithms

Exercise 3

- **Submit electronically (MOODLE) as one archive named after you until Thursday, 01.12.2022 9:00**
- **Archive should encompass a PDF and your code**
- **Code should be compilable (include Makefile or similar)**
- **Include names on the top of the sheets.**
- **A maximum of two students is allowed to work jointly on the exercises.**

Reading assignment

CUDA toolkit examples

- `2_Concepts_and_Techniques/histogram` with PDF in `doc`
- `6_Performance/transpose` with PDF in `doc`
- `2_Concepts_and_Techniques/sortingNetworks`

CUDA C Programming Guide

- C.7. Grid Synchronization

3.1 Warp functions (5+10 points)

In the `CUDA_C_Programming_Guide.pdf` read about B.19. Warp Vote Functions, B.20. Warp Match Functions, B.21. Warp Reduce Functions, and B.22. Warp Shuffle Functions. These allow efficient communication between the lanes in a warp and enable many fast implementations:

- `__shfl_up_sync(mask, value, delta)` moves data up the lanes by delta, however, the lower delta lines do not get anything. Which instruction gives us up-rotation, such that the lower delta lines would get the values from the highest delta lanes?
- Suppose each lane of a warp holds an int. We would want that each lane of the warp knows the lane index of the lane which holds the minimal int value. Which series of warp instructions will do that quickly on a compute capability 7.x GPU?

3.2 Reduction performance (15 points)

The example `2_Concepts_and_Techniques/reduction` demonstrates efficient reduction algorithms on the GPU utilizing multiple kernel calls. In `2_Concepts_and_Techniques/threadFenceReduction` a different technique is used at the end of the first kernel, namely, the last thread block completes the summation of the partial sums from the previous thread blocks, that otherwise would be processed in the next kernel.

With the option `-multipass` you can switch whether the version with multiple passes/kernels should be executed or the version with a single pass using the thread fence. By testing both examples for different number of thread blocks and data sizes (both can be set with command line options) determine which runs faster. Make a comparison plot of multipass vs. single-pass showing the best thread block size for each data size.

3.3 Reduction in one kernel (40 points)

Instead of the technique in `2_Concepts_and_Techniques/threadFenceReduction` we could perform the one kernel reduction in a different fashion using the new cooperative launch feature. Start as many blocks as can run simultaneously and submit the kernel launch with `cudaLaunchCooperativeKernel`. Each block processes now multiple tiles of the reduction and can now easily accumulate the sum for all of its tiles in a single register, which is a partial sum of the entire array. In the end these partial sums of the blocks must be reduced, which can be done in two different ways:

1. One thread of each block increases the value of a global accumulator atomically with the block accumulator value.
2. An array with one field for each block is used to store to the partial sums. Subsequently a `grid.sync()` is called, one block reads the values of that array of partial sums, reduces them, and stores the result in global memory.

Use the default kernel of `2_Concepts_and_Techniques/reduction` to calculate the tile reductions. Provide a plot with the total amount of Bytes reduced on the x-axis and the bandwidth in GB/s on the y-axis. Start with 100 MB and measure until the device memory is exceeded. Compare the performance of both above mentioned versions.