Universität Heidelberg
Institut für Technische Informatik

Holger Fröning
Lorenz Braun

GPU Computing
Term 2015/2016 (Winter)

Exercise 10

- **Return electronically until Thursday, 26.01.2016 14:00**
- **Include name on the top sheet. Hand in only one PDF.**
- **A maximum of three students is allowed to work jointly on the exercises.**
- **Hand in source code if the exercise required programming.**

10.1 Reading

- Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. 2008. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (SC '08).
- Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 2010. 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10).

(20 points)

Heat Relaxation

This is a continuation of the last exercise in which we developed a stencil code for GPUs. In this exercise, we will further optimize it by using texture memory. Texture memory provides readonly access that is optimized for spatial locality and cacheable.
Using texture memory in this case can prevent redundant loads to global memory. The texture memory as a cache will serve loads from several blocks that request the same region. However, texture memory is not designed to reduce latency, so texture access has similar costs compared to global memory access, independent of a cache hit or miss.
The main benefits of texture memory are to consolidate redundant global memory accesses (exercise 10.2) and to allow for more flexible data access patterns (exercise 10.3). We will explore both benefits using the stencil experiment.

10.2 Stencil Code – Using Texture Memory

- Modify your code to use texture memory loads instead of global memory loads for the heat array.
- For this purpose, the load from global memory to shared memory can be replaced by using the function tex1Dfetch to access global memory:
  Before: `Smem[tx][ty] = gmem[index];`
  Now: `Smem[tx][ty]= tex1Dfetch(tex, index);`
- Test your program extensively and verify correctness by comparing the result to a previous version.

(35 points)

10.3 Stencil Code – Optimizing Instruction Overhead

- Continue with the stencil code using texture memory. The flexible data access pattern of texture memory allows simplifying logic expressions in the code, thereby reducing instruction overhead. For more information, see the Nvidia CUDA Programming Guide:
  `http://docs.nvidia.com/cuda/cuda-c-programming-guide`
- For this workload, this in particular the if-clauses required for the halo regions are now obsolete (coordinates that are out of range). Simplify your code by leveraging the more flexible access pattern of texture memory.
- Test your program extensively and verify correctness by comparing the result to a previous version.
- Compare the performance results with the previous versions. Report performance numbers in terms of the achieved memory bandwidth [GB/s]. Interpret your results!

(20 points)

Total: 75 points