Universität Heidelberg
Institut für Technische Informatik

Holger Fröning
Benjamin Baumann

Introduction to High Performance Computing
Term 2013/2014 (Winter)

Exercise 9

- **Return electronically (MOODLE) until Monday, 12.01.2014 23:55.**
- **Include name and your account (introhpc[xx]) on the top sheet.**
- **A maximum of two students is allowed to work jointly on the exercises.**

Notes

- For CUDA developments, use the same nodes as for the previous exercises. Each node is equipped with two GTX480 GPUs. In this exercise, make only use of one GPU.
- Ensure that the computing nodes are quiescent when performing experiments. In particular, look for other participants which could be performing tests. Try not to disturb them; you also might not want to be disturbed.

9.1 Reading

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

- Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. and Phillips, J. C. 2008. GPU Computing. In Proceedings of the IEEE, 96, 5 (May 2008), 879–899.

(10 points)

9.2 SAXPY –CUDA implementation

- Goal is to develop a CUDA program that performs a saxpy operation of the form y[] = a*x[]+y[].
- Start with implementing a sequential version for the CPU. Use this version to ensure correct results when running on the GPU.
- Implement a CUDA-based parallel version of this program. Ensure the following:
  - Optimize thread per block count and blocks count total to maximize performance
  - Dynamic allocation of all structures
  - Verify results of parallel execution with sequential execution
- Implement two versions, one using pageable host memory and the other using pinned host memory.
- Include instrumentation for time measurement (e.g., gettimeofday()), but do not include initialization in this measurement.
- Report your results graphically for both implementations. Vary the problem size on the x-axis. Include graphs for CPU time, GPU time, and data movements. Interpret your results.

(20 points)

9.3 SAXPY – MPI+CUDA implementation

- Implement a CUDA version that is based on MPI for distribution and control purposes. You can constrain the computation to GPUs only, i.e. do not use CPU to help processing the workload.
- Choose a problem size in a way that all data still fits into the global memory of a GPU, i.e. beside the initial data distribution and result gathering there is no significant communication among the nodes. Choose the problem size in a way that the execution yields stable results.
- Verify correctness using one of the previous versions.
- Include the time for initial data distribution and result gathering in your time measurements.
- Choose a suitable problem size to yield stable results. Execute on up to 6 nodes and report execution time, speed-up and efficiency in the following table. Interpret your results!

Nodes	**Time [s]**	**Speed-up**	**Efficiency**
1 (no mpi)		1.0	100%
2			
3			
4			
5			
6			
7			
8			

(30 points)

Total: 60 points