Universität Heidelberg
Institut für Technische Informatik

Holger Fröning
Benjamin Baumann

Introduction to High Performance Computing
Term 2014/2015 (Winter)

Exercise 10

- **Return electronically until Monday, 19.01.2014 23:55 Uhr**
- **Include name on the top sheet. Stich several sheets together.**
- **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.

10.1 Matrix multiply – parallel version using CUDA

- Goal is to develop a CUDA program performing a matrix multiply operation of the form C = A * B. Matrices can be assumed to be square.
- Start with the sequential version implemented in one of the previous exercises. It is up to you to use the non-optimized or optimized version. 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 matrices
  - Verify results of parallel execution with sequential execution
  - Maximize performance, for instance using shared memory and memory access coalescing.
  - Decouple the tile size (used for shared memory) and block size to optimize the use of shared memory. You can use some CUDA calls to determine specific properties of the GPU, or use the deviceQuery command.
- Include instrumentation for time measurement (e.g., gettimeofday()), but do not include initialization in this measurement.
- Initialize according to: A[i,j] = i+j, B[i,j] = i*j
- Fill out the following table.
- Report your results graphically for your best implementation. Explain which optimizations you have made to reach this performance. Put performance numbers into context, i.e. explain if the problem is memory- or compute-bound for a GPU and how much out of the theoretical peak performance you have reached.

Holger Fröning
Benjamin Baumann

Problem size	Time [s]	Speedup (over seq. version)	FLOPs [M/G]	GFLOP/s
128				
256				
512				
1024				
2048				
4096				
8192				

(40 points)

10.2 Matrix multiply – parallel version using MPI+CUDA

- 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.
- Execute on up to 4 nodes and report execution times and speed-up in the following table. Include data distribution and result gathering in the measurement. Report a graphical representation of the results. Choose a suitable matrix size for stable performance results.
- Report speed-up and efficiency and interpret the results.

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

(30 points)

Total: 70 points