Topic 1: Matrix Multiplication on CUDA

For problems working on matrix multiplication, an implementation template (mm.cu) is provided. The template includes a basic implementation of matrix multiplication. You should replace it with your own solutions.

When you measure the performance of the programs you write for this lab, make sure the measurement is stable and consistent. A simple way to do that is to repeat the execution multiple times and use the average.

Problem 1: NVIDIA CUDA

You can use any computers equipped with CUDA GPU. The department machine that can be used for this lab is "cpeg655.ece.udel.edu". The machine is equipped with four NVidia graphic cards. You should try logging on to cpeg655.ece.udel.edu and inform the instructor as soon as possible if you have any problem accessing the machine. Log in with your ECE acad account.

SLURM (Simple Linux Utility for Resource Management) is used to queue and run jobs on the CUDA hardware. Jobs can be queued to run using the srun command:

- srun -N1 --gres=gpu:G <command>
  - -N1 indicates the number of nodes (computers). Currently, only one CUDA machine exists, so don't change this option.
  - G is the number of GPU cards the job requires. Currently 4 cards are available for jobs on cuda.acad.

Additional SLURM commands are:

- squeue - Shows the current state of the SLURM queue for cuda.acad.
- sinfo - Shows info about the current SLURM configuration.
- scancel - Used to cancel a currently running job.

More information and documentation about SLURM is available at https://computing.llnl.gov/linux/slurm/documentation.html

CUDA is installed to /usr/local/cuda. You may need to add /usr/local/cuda/bin to your path, and /usr/local/cuda/lib64 to your LD_LIBRARY_PATH. Please avoid setting your LD_LIBRARY_PATH if things are working without setting it. Also you need to set LIBRARY_PATH viable using the command “export LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:$LIBRARY_PATH”.

SLURM is installed in /usr/bin. You usually don’t need to add /usr/bin to your path.

a. Write a multithreaded program using the CUDA programming model to compute matrix multiplication, that is, *C=A*B*, where *A, B, C* are three matrices with size *N-by-N*. One thread computes one element of the product matrix *C*. Use only one thread block to compute the whole matrix. Measure and report the performance of two sizes, *N=16* and *N=32*.

b. Write a multithreaded program using the CUDA programming model to compute matrix multiplication using the tiling algorithm. One thread still computes only one element of the product matrix, but a thread block computes only a tile of the matrix *C*, and uses multiple thread blocks to compute the whole matrix. Measure and report the performance of two matrix sizes, *N=512* and *N=1024*, and for each size, program your code with two tile sizes, *8* and *16*.

Problem 2: Optimizations on CUDA

This problem asks you to improve the performance of the four versions of code that are developed for the problem 2.b. For each version:

a. Extend the version of 1.b with N=1024 by making one thread compute a NB*NB tile of matrix C. Also try to change the number of threads NT*NT in a thread block. You can use multiple thread blocks. If the number of thread blocks is NK, the relationship between NB, NT, NK and the matrix size N is $N^2 = NB^2 * NT^2 * NK$. Find the best NB and NT for N=1024.

b. Unroll the innermost loop of the version **a** with factors 4 and 8. Measure the performance, and explain the performance difference.

c. For the 8-unrolled versions from the problem 2.b, instead of directly loading elements of matrix A from the global memory, loading all elements of A used in the tiled loops into shared memory first, and replace all references to the original A with the references to the shared memory copy of A. Create two versions of shared memory usage, one with bank conflict, e.g., a half-warp write/read 4 banks, and the other without any bank-conflict. Explain why one has conflicts and the other doesn't. Measure the performance.

What to submit:

The package you submit for this lab should include your source code, performance results and analysis, and brief description that you think might help the instructor understand your code, e.g., about any design choices you make in your program. The instructor will compile, run and measure the performance of your code. Therefore, you should also describe how to compile and run your code in your submitted documentation.