CS 6023 GPU Programming
Assignment 1

Handout date: 13/08/2018
Due date: 22/08/2018 (end of day)

General instructions:

1. The goal of this assignment is to get started with programming in CUDA C and to understand performance as a function of thread and block configurations.
2. This is an **individual assignment**. Collaborations and discussions with others are not encouraged. However, if you do collaborate with anyone, you are requested to explicitly state the same in your submitted report.
3. You have to use C with CUDA for your implementation.
4. You have to turn in three components: (a) well documented code (.cu files for each programming question), (b) output files (.txt for each relevant programming question), and (c) a detailed report (single .pdf file) of the results of the experiment. All these are to be submitted on Moodle. No hard copy is required.
5. Code will be checked for functional correctness, quality of code, and performance. Report will be verified for completeness and to-the-point answers to the subjective questions posed (eg. see question 5 below).
6. If you have doubts please raise them in the Moodle Q&A forum, and follow the same regularly. Also, to receive programming and other support, attend the TA session on Thursday 10am at the usual classroom CS 36.

Submission instructions:

1. Make a report PDF as *<username>_report.pdf*, where <username> is the username used by you to login to the GPU cluster. For many it may be your roll number.
2. Put all files inside a folder named your username and compress the folder into a tarball in the following format : *username_A01.tar.gz*
   [example directory structure here]
3. Upload the tarball in Moodle. For instructions on how to do this contact the TAs.

Q1. **(1 point)** Every GPU has a variety of technology dependent parameters specific to that generation of hardware. These parameters provide vital clues on optimizing the performance of parallel programs. We will now see how to identify these parameters for a given GPU.

CUDA Runtime supports querying of several device properties by the user (see cudaDeviceProp in the CUDA API). Write a program in CUDA C to query the following parameters:

- Is L1 cache supported locally?
- Is L1 cache supported globally?
- What is the L2 cache size (in bytes)?
- What is the maximum allowed number of threads per block?
- How many registers are allocated per block?

- How many registers are available in a Streaming Multiprocessor?
- What is the warp size?
- What is the total amount of memory available in the GPU (in bytes)?

Report these results for Tesla K40c GPU present in the cluster. The output file should contain these values in the given order. Only one value per line should be reported, nothing else.
Submit the source code as <username>_1.cu
Submit the output as <username>_1_out.txt

Q2. (1 point) The performance of a parallel application is bounded by the roof-line model. We will now plot this for the GPU used in all assignments.

Use the parameters from Q1, to plot the roofline model for the K40c GPU.
Include the plot of the roof-line model in the report titled <username>_report.pdf

Q3. (2 points) As discussed in the lectures, GPUs are efficient in performing the same operation in parallel on very large data sets; this is referred to as SIMD operation. We will now apply SIMD to simple vector arithmetic.

In this question, you need to perform addition of two one-dimensional vectors of size 32,768 = 2^15. Generate two random vectors of this size on the CPU and add them on the GPU. The kernel is to be such that each thread operates on a single data item. For the invocation of the kernel, use 128 blocks of 256 threads each. Report the results in a text file containing one number each on three separate lines: first the two addendums and then the sum.
Submit the source code as <username>_3.cu
Submit the output as <username>_3_out.txt

Q4. (2 points) As discussed in the lectures, the number of blocks and threads used in a kernel invocation can be changed. The choice trade-offs the variety of resources available to threads and blocks on hardware. We will study more about this in the lectures. Now, we will empirically evaluate this trade-off.

Modify the program from Q3 to explore various choices of number of block and threads. Note that it is advisable to constrain the number of threads and blocks to powers of two. For the various configurations, identify the execution time for the addition on the GPU. Plot the runtime as a function of the threads and blocks. Also identify the optimal configuration.
Submit the source code as <username>_4.cu
Include the plot and the optimal configuration in the report titled <username>_report.pdf

Q5. (2 point) We have so far only considered a single arithmetic operation per kernel invocation. We will now explore how the throughput changes as we increase the number of operations per thread.

Modify the program of Q3 (with the optimal thread and kernel parameters of Q4) to support multiple add operations in each thread. Vary the number of operations in powers of two from 1 to 32. Accordingly adjust the number of threads. Plot the runtime as a function of the number of operations. Also identify the optimal configuration.
Submit the source code as <username>_5.cu
Include the plot and the optimal configuration in the report titled <username>.pdf

Q6. (2 points) How does the runtime of an application increase with the amount of work to be done? Given that vector addition is an O(n) operation on a sequential machine, we expect a linear increase in time as a function of input size. We will now empirically evaluate this on a GPU.

Modify the program of Q3 (with the optimal thread and kernel parameters of Q4 and optimal thread workload from Q5) to add vectors of sizes varying from 2^15 to 2^20, taken in powers of 2. For each case generate random numbers for the vectors in the CPU and perform the vector addition on the GPU. Plot the runtime as a function of the input size.
Submit the source code as <username>_6.cu
Include the plot in the report titled <username>_report.pdf

Q7. (Bonus!) Given all the empirical data you have collected on the GPUs, it is time for some reflection. Reason about the observed runtimes in light of the device properties you identified in Q1.
Include your comments in the report titled <username>_report.pdf