Homework 4

CAAM 520, Spring 2019

Posted March 20, 2019. Due April 5, 2019 by 5pm.

1. Your solutions to the homework must be committed to your Github repository in a sub-directory HW04.
2. You are required to include a Makefile that creates an excecutable called "hw04' in that directory. All source code, header files, LaTeXfiles, Makefiles, etc. should be committed to your repository in the same sub-directory.
3. Use LaTeX to write and typeset your report (saved as "report.pdf" in your repository sub-directory). Document all your steps, including code snippets within your report.
4. You may only consult the instructor for assistance, but are encouraged to use textbooks and internet resources. Cite all external resources used via footnotes or a bibliography.

Assignment: Your task is to create a CUDA code which computes the solution $\boldsymbol{u}$ to the system $\boldsymbol{Au} = \boldsymbol{b}$ arising from the finite difference discretization of Poisson's equation using a pure Jacobi iteration (weight equal to 1). You may use the change in the solution update as a stopping criterion. Use single precision for all computations (`float` instead of `double`).

Your code should include the following components:

1. host code to allocate arrays and move data onto and off the GPU.
2. two CUDA kernels: one to perform one Jacobi iteration, and another to compute the stopping criteria using a reduction.

Your code should follow guidelines for efficient GPU programming (e.g. avoid repeated data movement between host and device, coalesced global memory accesses where possible, efficient reduction code, etc), and work for a variable number of threads per block (grid size).

Documentation: Your report should include the following:

1. descriptions of your partition of parallel work and algorithm (e.g. threading structure, i.e. describe a map from the thread array indices to the work done by each indexed thread).
2. documented verification of your code's correctness through reported errors and comparison with serial code.

3. computed computational throughput (GFLOPS/sec) and bandwidths (GB/s) of your kernel (using `nvprof --metrics dram_write_throughput, dram_read_throughput, flop_count_sp`). Use several different values for N and thread-block size.

4. plots of computational throughput against arithmetic intensity for several different values of N and choices of thread-block size on a roofline plot, and a discussion of the computational performance of your code.

Extra credit (25 points):

1. Improve the performance of your code by using a shared memory implementation. You may wish to use a different parallelization pattern for the shared memory case. Cite any references you use.

2. Compute computational throughput and bandwidth for several values of N and any computational parameters (e.g. thread-block size), and plot the resulting data points on a roofline plot. Compare the results to the case without shared memory.