CAAM 520 Spring 2015 Homework 04

T. Warburton

March 19, 2015

You must complete the following task by 5pm on Friday 03/27/15.

Your solutions to the homework must be committed to your Subversion repository in a sub-directory HW04.

You must work on your own for this homework. You may use the reference solution code provided for HW01 posted here or your own codes from HW01-HW03.

All source code, header files, \LaTeX files, Makefiles etc should be committed to your own repository in directories named HW04/Q1 and HW04/Q2. You are required to include a Makefile in each directory that makes an executable called main in each directory (that is we will assume your executables are HW04/Q1/main and HW04/Q2/main).

Use \LaTeX to write and typeset your report (saved as HW04/report.pdf in your Subversion repository).

You may only consult the instructor or graders for verbal assistance. You are encouraged to use textbooks and internet resources. You must cite all resources used via footnotes or a bibliography.

100 points will be awarded for the successful completion of each question.
Extra credit will be awarded as appropriate.
Q1 Build an OpenMP based sparse matrix solver for a structured resistor network.

Problem description: Build a parallel loop current solver using OpenMP #pragma preprocessor directives.

Tasks: Your task is to create an OpenMP parallel C++ code that iteratively solves for the currents in each indivisible loop of a structured resistor network. You should write a flexible code that can handle an $N \times N$ grid circuit for arbitrary, user supplied, integer $N$. Choose one of the iterative linear system solvers from HW01.

Loop directives: Design a work partitioning scheme for the resistor network using features from OpenMP.

Parallel algorithm: Describe in pseudocode the implementation of your chosen distributed iterative method including parallel communications.

OpenMP implementation: Implement your algorithm using OpenMP compiler directives. You can choose which subset of OpenMP functions you use.

Test case: Compare the output of your starting serial code with the output of your OpenMP based for 1,2,3,...,8 OpenMP threads. Explain any differences.

Performance: Perform a strong scaling study, i.e. for a fixed size problem vary the number of OpenMP threads used and estimate the time the iterative solver takes to complete a calculation. Use say $N = 500$ with 1,2,3,4,5,6,7, and finally 8 OpenMP threads on ketchup [note that the Intel i7-5960X CPU in ketchup has eight cores with two way hardware threading per core]. Present your results in a graph plotting speed-up versus number of OpenMP threads.
Q2 Build a CUDA accelerated sparse matrix solver for a structured resistor network.

Problem description: Build a parallel loop current solver using the CUDA API from NVIDIA.

Tasks: Your task is to create an CUDA parallel C++ code that iteratively solves for the currents in each indivisible loop of a structured resistor network. You should write a flexible code that can handle an $N \times N$ grid circuit for arbitrary, user supplied, integer $N$. Choose one of the iterative linear system solvers from HW01.

Kernel design: Design a work partitioning scheme for the resistor network using CUDA features that we have covered so far like shared memory and maximizing memory efficiency.

Parallel algorithm: Describe in pseudocode the implementation of your chosen distributed iterative method including threading structure, i.e. describe a map from the thread array indices to the work done by each indexed thread.

CUDA implementation: Implement your algorithm using features from CUDA that we have discussed so far including:

- Host code to allocate arrays on host and device.
- Host code to explicitly move data onto and off the accelerator.
- Design thread grid (thread-block size, and grid size).
- Kernel device code to perform the current updates in your iterative method.

Test case: Compare the output of your starting serial code with the output of your CUDA code. Explain any differences.

Performance:

- Compare the time taken by your OpenMP on the Intel i7-5960X CPU and the CUDA solver on one of the three NVIDIA GTX 980 graphics cards hosted on the ketchup machine.
- Estimate the memory bandwidth attained by each solver (bandwidth is typically estimated as gigabytes of data loads and stores per unit time).
- Perform a weak scaling study for your CUDA code: for a range of $N$ plot the time taken per iteration against $N$.
- Use the internet to find the specifications for the CPU and GPU in ketchup and comment on the efficiency of your implementations.