

Parallel Computer Architecture

CUDA Programming Assignment

Equation Solver

Instructor: Prof. Naga Kandasamy
ECE Department
Drexel University

March 4, 2018

The assignment is due March 14, 2018 by 11:59 pm via BBLearn. You may work on the problem in a team of up to two people. The submitted code must be your own. Please do not copy code from other students/teams or from online sources. Violation of this policy will result in a score of zero for the entire assignment.

Consider the Gauss-Seidel equation solver discussed within the lecture notes on how to write parallel programs—see the file called `parallelization.process.pdf` on BBLearn. You will find the source code for the reference implementation in the zip file on BBLearn.

Recall that the order in which the grid points are updated in the sequential algorithm is not fundamental to the Gauss-Seidel solution method; it is simply one possible ordering that is convenient to program sequentially. Since the Gauss-Seidel method is not an exact solution method but rather iterates until convergence, we can update the grid points in a different order as long as we use updated values for grid points frequently enough, a technique called the *Jacobi method* where we don't use updated values from the current iteration for any grid points but always use the values as they were at the end of the previous iteration. Using the sequential program as a starting point, develop a parallel version of the Jacobi method using an element-based decomposition strategy where each GPU thread is responsible for processing a single grid element.

The program provided to you accepts no arguments. It creates a randomly initialized grid of $N \times N$ elements and applies the update rule to each element within the grid until the specified convergence criteria is satisfied. The solution provided by the GPU is compared to that generated by the CPU by printing out the relevant statistics.

Answer the following questions.

- **(10 points)** Edit the `compute_on_device()` function in the file `solver.cu` and the `solver_kernel_naive()` function in `solver_kernel.cu` file to complete the functionality of the equation solver on the GPU using only global memory.

- **(15 points)** Improve the performance of the kernel developed in the previous step by using shared memory on the GPU. Edit the kernel function `solver_kernel_optimized()` in the `solver_kernel.cu` file to complete the functionality.
- Upload all of the files needed to run your code on BBLearn as a single zip file. Submit a short report describing: (1) the design of your kernels using code or pseudocode to clarify the discussion; (2) the speedup obtained over the serial version for both the naive and optimized kernels, for grid sizes of 2048×2048 , 4096×4096 , and 8192×8192 ; and (3) sensitivity of your kernels to thread-block size in terms of the execution time.