GPU Programming for Scientific Computation Syllabus

Steven Reeves

1 Course Description

Graphics Processing Units (GPUs) originally made their debut as specialized processors designed to rapidly manipulate and alter memory to hasten the creation of images in a frame buffer intended for display devices. However, this efficacy for fast operations was noticed in more general applications. Now modern GPUs are used for efficient computation in computer graphics, image processing, machine learning, and in general high performance computing applications. The goal of this course will be to introduce the concept of offloading work onto GPUs as accelerators for various applications. Students will be expected to write their own CUDA C code for homeworks and an end-of-term project. Prerequisites should be a familiarity with the C programming language, SSH and other remote desktop protocols, and some numerical analysis.

2 Schedule

The weekly schedule and topics are as follows.

Weeks 1 - 2

Introduction to the ideas of parallelism and the GPU programming model

- CPU vs GPU
- Parallelizing algorithms on paper
- First CUDA program

Weeks 3 - 4

Hardware of Graphics Processing Units and parallel communication patterns

- Brief on GPU architecture
- Basics of CUDA C
- Floating point precision and support on GPUs

Weeks 5 - 9

Key parallel primitives and algorithms on GPU. The CUDA programming language will be mastered while learning how to implement these algorithms.

- Matrix Operations
- Stencil
  - Image Blurring, Filters, Gauss-Jacobi
  - Finite difference updates for PDEs
- Histogram, binning

- Reduce
  - Maximum and Minimum
  - Summation
- Prefix-sum (Scan) Algorithm
  - Radix Sort
  - Generating Cummulative Distributions
- Complex algorithms
  - N-body solutions

Week 10

Optimizing GPU Applications

- Coalesced Memory Transactions
- Grid Blocks, Thread Blocks, domain decomposition
- Asynchronous Kernels and Multistreaming

Possible Items:

Libraries on GPU

- cuBLAS
- Thrust
- cuFFT
- cuRAND
- more

Advanced Concepts

- Multi-node GPU processing
- Multi-GPU per node processing
- CUDA in other languages (Python/Fortran)
- Scaling Studies

3 Course Reference Material, Course Work, and Grading Scheme

3.1 Planned Learning Outcomes

Upon completion of this course, passing students will have a working proficiency with CUDA, algorithmic GPU programming and parallel computing. They will have a familiarity with classic scientific computing algorithms and problems, as perscribed within weeks 5-8 of the schedule. Finally, students will be able to optimize and debug GPU code.

3.2 Required Texts

The reference materials for this course will be GPU Gems 3, NVIDIA Corperation: link1 and CUDA Programming Guide, NVIDIA Corperation: link2

3.3 Course Work For Students

The course work will be organised as follows: a weekly homework assignment, either theoretical, code based, or both. Further, there will be an end of the term project for students to complete. There will be two options for the term project. The first option will be a standard CUDA GPU programming project with a report discussing results. The second option will be for the students to apply GPU programming to a problem of their choice, with instructor approval, and can give an optional presentation for extra credit.

3.3.1 Hourly Breakdown for Student Participation

Lecture will take approximately 3 hours per week. Homework, studying, and debugging will take students, on average, 9 hours per week to complete. The final project should take the students no more than 30 hours. This includes time for programming, debugging and writing the report.

3.4 Grading Schemes

The grading scheme for the course is as follows:

A	$100\% \geq x \geq 90\%$
B	$90\% > x \geq 80\%$
C	$80\% > x \geq 70\%$
D	$70\% > x \geq 60\%$
F	$60\% > x$

Final grades will have the distribution in which 60% of the total grade will be comprised of homework scores and 40% is reserved for the final project.