EECS 368/468 Programming Massively Parallel Processors with CUDA

Course Description

This course focuses on developing and optimizing applications software on massively parallel graphics processing units (GPUs). Such processing units routinely come with hundreds to thousands of cores per chip and sell for a few hundred to a few thousand dollars. The massive parallelism they offer allows applications to run 2x-450x faster than on conventional multicores. However, to reach this performance improvement, the application must fully utilize the computational, storage and communication resources provided by the device. This course discusses state-of-the-art parallel programming optimization methods to achieve this goal.

Ideally this course will bring together people with strong programming skills, with people with a strong need for solving compute-intensive problems that can benefit from programming graphics processors. The initial part of the course will discuss a popular programming interface for graphics processors, the CUDA programming tools for NVIDIA processors. The course will continue with a closer view of the internal architecture of graphics processors and how it impacts performance. Finally, implementations of applications and algorithms on graphics processors will be discussed.

The course is lab intensive, and it will utilize the machines at the **Wilkinson lab**. Students taking the course for EECS-368 credit will work on labs that utilize advanced parallel programming, data layout, and algorithm decomposition concepts. Students taking the course for EECS-468 credit will work on the same labs and also a quarter-long open-ended final project that draws upon their own interests and line of research. Ideally, in their final project these students will form interdisciplinary teams and complete the first steps of optimizing a real-world compute-intensive problem in science or engineering (e.g., materials science, astrophysics, civil engineering, chemical engineering, biomedical engineering, etc.).

Course Coordinator

Prof. Nikos Hardavellas

Course Staff

Instructor: Prof. Nikos Hardavellas
Email: nikos@northwestern.edu
Office Hours: Tuesdays, 3:30-4:30pm, Mudd 3517

TA: Emirhan Poyraz
Email: IbrahimPoyraz2014@u.northwestern.edu
Office Hours: Wednesdays, 2-3pm, Wilkinson Lab, Tech M338

Exam Day and Time

Thursday, 03/19/2020: 12-2pm. We do not have exams, but this will be our overflow day. Plan to be there as needed (i.e., do not catch an early flight back home).

In compliance with Section 504 of the 1973 Rehabilitation Act and the Americans with Disabilities Act, Northwestern University is committed to providing equal access to all programming. Students with disabilities seeking accommodations are encouraged to contact AccessibleNU (ANU) at +1-847-467-5530 or accessiblenu@northwestern.edu. AccessibleNU is located at 2122 Sheridan Road, Room 130. You can find directions and more information on ANU at their website at https://www.northwestern.edu/accessiblenu/. The instructor is also available to discuss disability-related needs during office hours or by appointment.

Prerequisites and Recommendations

Students should have the following prerequisites, **or obtain permission from the instructor**:

- EECS 213 or equivalent understanding of computer systems and architecture
- EECS 211 or EECS 230 or equivalent intermediate C programming experience

Useful but not required: EECS 311, EECS 361, EECS 358.

Prerequisites By Topic

- Basic concepts of computer architecture: processors, ALUs, memories, caches, input-output
- Basic concepts of computer systems: memory accessing and layout, padding, alignment
- Intermediate concepts on programming using C: control flow, scoping, type casting, pointers
- Simple concepts of data structures like arrays and linked lists in programs
- Some knowledge of scientific and engineering applications

Textbooks

Required:

- *Programming Massively Parallel Processors: A Hands-on Approach*. D. Kirk and W.-M. Hwu. Morgan-Kaufman

Reference textbooks (not required):

- *CUDA by Example*. J. Sanders and E. Kandrot. Pearson/Addison-Wesley
- *Patterns for Parallel Programming*. T.G. Mattson, B.A. Sanders, and B.L. Massingill. Pearson/Addison-Wesley
- *Introduction to Parallel Computing*. A. Grama, A. Gupta, G. Karypis, and V. Kumar. Pearson/Addison-Wesley
- *Heterogeneous Computing with OpenCL*. B.R. Gaster, L. Howes, D.R. Kaeli, P. Mistry, and D. Schaa. Morgan Kaufmann

Communication Channels

- We will use Canvas to post homework assignments, slides, reference documents, etc. **It is your responsibility to check Canvas regularly**.
- We will use Piazza for class discussion and to post class announcements. This particular channel is intended to foster communication among you, the students, the instructor, and the TA. You'll find that someone else in the class will have thought of the same problem that you have and will perhaps have some valuable insight that will prove helpful. The teaching staff will be monitoring the discussion threads and will step in with guidance soon after questions are posted. Students are also encouraged to answer questions. Before asking a question, please check the previous piazza postings; often you will find that the answer to your question has already been provided. The link to piazza is http://piazza.com/northwestern/winter2020/eecs368468
- Finally, there is always email. We encourage you to post all class-related questions to Piazza, and use email only to discuss a sensitive issue with the instructor/TA or to request an appointment.
- **It is your responsibility to check the Canvas web pages and Piazza postings regularly.**

Lab Assignments (tentative, subject to change)

There will be several programming assignments. Each programming assignment will involve successively more sophisticated programming skills. The labs will be done in **groups of 2-3 students**. All assignments will be submitted electronically through Canvas. Signup into one of the available Lab Groups on Canvas and submit as a group. Multiple submissions are OK, only the last one will be graded.

Tentative lab schedule:

1. Matrix multiplication. The lab's focus is on producing correct code. This project reinforces the acquisition of basic GPU/CUDA programming skills, the software interface, and the basic architecture of the device.
2. Tiled matrix multiplication. This lab focuses on data layout and decomposition, and full utilization of shared memory resources and global bandwidth through bank conflict avoidance and memory coalescing.
3. Histograms. In this lab you are called to define optimization goals and strategy, implement them, and keep a research lab journal on which you report statistics and analyze every optimization you tried, even ones that did not work or degraded performance. For this assignment you will need to read recent research papers that outline some of the best-known ways to solve this problem.
4. Parallel prefix sum / vector reduction. This lab focuses on the application of efficient parallel algorithms that utilize shared memory and synchronization and minimize path divergence.
5. 2D convolution (tentative – typically we do not have time for it, but I always try). While the previous labs involved detailed instructions and scaffolding code, this lab provides students only with a problem statement, along with optimization goals and hints. This lab's focus is on independent thinking and reinforces concepts learned in the previous labs.

You will do the lab assignments on the Linux machines at **Wilkinson Lab**. You should have physical access to the lab. If you prefer to work remotely at a Wilkinson Lab machine, the hostnames are at http://www.mccormick.northwestern.edu/eecs/documents/current-students/student-lab-hostnames.pdf. If you cannot access these machines, contact root@eecs.northwestern.edu. Similarly, contact root if you do not have physical access to the labs. **All labs will be graded on the Wilkinson Lab machines; it is your responsibility to make sure your code works there**.

Final Project (EECS-468 only)

The project focuses on open-ended research. The work will be a quarter-long open-ended final project that draws upon the students' own interests and line of research. Ideally, in their final project the students will form interdisciplinary teams and complete the first steps of optimizing a real-world contemporary compute-intense problem in science or engineering (e.g., materials science, astrophysics, civil engineering, etc.). The students are expected to base their ideas on recent literature and their own line of research and needs. It is the students' responsibility to propose a research project suitable for this class. The project selection requires the permission of the instructor. For this, there will be a project proposal phase. The final project will culminate to a paper report, and a slide or poster presentation (TBD) and demo held at the final few class meetings. Submit project materials by signing up into one of the available 468 Project Groups on Canvas and submit as a group.

Late Assignments Policy

Late assignments: 10% penalty for the first 24 hours after a homework is due. Zero credit after that. We will use the timestamps provided by Canvas to designate assignments as late. It is your responsibility to submit assignments early enough to guarantee you will have no problems with network connection, printing your hand-in, etc. **There will be no deviation from this rule.**

Academic Integrity Policy

It is the responsibility of every student in this class to be familiar with and to adhere to the Academic Integrity Policies of Northwestern University and the McCormick School of Engineering. The policies are

at http://www.mccormick.northwestern.edu/students/undergraduate/academic-integrity.html. Any suspicion of violation of these policies will be reported immediately to the director of the program. If you are in doubt whether your actions constitute a violation of the above policies, **ASK THE INSTRUCTOR**.

Please, **do not share code or solutions**. You shouldn't even see each other's code or solutions. What you turn in must be your own (or your group's own) work. Copying code, solution sets, etc., from anywhere (e.g., other people, web, github) is strictly prohibited. If you discuss your work with another group, please list their names in your hand-in to avoid future problems.

Grading

Grades will be assigned according to the distribution below. There will be no grade curving. Letter grades will be assigned based on the default percentage-to-letter-grade mapping on Canvas.

	EECS-368	EECS-468
Lab assignments	80%	50%
Final project		30%
Class participation	20%	20%

Course Objectives

When a student completes this course, he or she should be able to:

(a) Apply knowledge of mathematics, science, and engineering (parallel programming techniques, kernel decomposition, synchronization, memory access optimizations, data layout transformations, branching/loop optimizations, algorithm cascading, Brent's theorem, input diagonalization, FP representations, regularization, compaction, binning, thread coarsening).

(b) Design and conduct experiments, analyze, and interpret data (design and conduct experiments on real massively parallel applications written using CUDA, utilize industrial tools to identify and overcome performance bottlenecks, measure execution time and speedup on GPU devices).

(c) Design a software system to meet desired needs within realistic constraints (design optimized massively parallel programs for GPUs by amplifying the utilization of constrained resources including PCIe bandwidth, global memory bandwidth, shared memory banks, texture and constant memory, warps, thread blocks, SMP registers, etc.).

(d) Ability to function on multidisciplinary teams (the collaborative term projects will ideally pair together computer scientists and engineers with domain experts).

(e) Ability to identify, formulate, and solve engineering problems using industry-strength CUDA tools.

(g) Ability to communicate effectively (research papers and reports, presentations, posters).

(i) Recognize the need for, and have the ability to engage in life-long learning (read recent research papers in an unfamiliar subject and assimilate knowledge without direct supervision).

(j) Gain knowledge of contemporary issues (state-of-the-art in GPU programming, energy-efficiency in computer architectures, high-performance parallel programming, heterogeneous computing).

(k) Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (GPUs, CUDA, occupancy calculator, parallel programming techniques).

ABET Content Category

100% Engineering (Design component)

Schedule (tentative, subject to change)

- **Week 1a-2a: Introduction to GPUs and their Programming Model**
  Grids, blocks, threads, memory model, execution model, software architecture, basic API
- **Week 2b-3a: GPU Architecture Overview and Execution API**
  Streaming processors, streaming multiprocessors, texture clusters, streaming arrays, block scheduling and execution, warps, scoreboarding, memory parallelism, register file, execution staging, subtyping, measuring time, compilation overview
- **Week 3b-4a: Memory Performance and Control Flow; Example application: Matrix Transpose**
  Coalescing, bank conflicts, DRAM memory controller organization, DRAM burst mode, tiling, padding, flow divergence
- **Week 4b: Performance and Occupancy**
- **Week 5: Putting everything to work: Reductions**
  Global synchronization, kernel decomposition, memory coalescing, non-divergent branching, eliminating shared-memory bank conflicts, loop unrolling, Brent's Theorem, algorithm cascading
- **Week 6: Work with atomics: Histograms; Vector Programming**
  Data races, atomics, input diagonalization, privatization, bank mapping, warp voting, vector loop semantics, order of evaluation, vectorization with forward dependencies, pragma directives for data temporal evolution, elemental functions, uniform/linear clauses, outer loop vectorization, vectorizing function calls
- **Week 7: Parallel Prefix Scan; Sparse Arrays**
  Inclusive scan, enforcing block ordering, exclusive scan, multi block/kernel parallel prefix scan, Warp voting, fences, compressed sparse array representations (CSR, diagonal, ellpack, coordinate, hybrid ell/coo, packet)
- **Week 8: Advanced Blocking/Tiling; Convolution; Textures**
  Thread coarsening, stencil, register tiling, two-pass methods, loop skewing, texture arrays, nearest-point sampling, linear filtering, clamping, CUDA arrays, 2D texture locality
- **Week 9: Input Binning, Parallel Programming Review**
  Binning, scatter/gather, cut-off summation, bin design, thread coarsening and binning
  Horizontal vs. vertical scaling, shared memory programming, message passing, data sharing models, algorithm structures vs. parallel programming coding styles, parallel program models vs. programming models, regularization, compaction, binning, data layout transformations, thread coarsening, scatter/gather, tessellation, limiter theory
- **Week 10, 11: Project Demos and Project Deliverables (Report Paper, Code, Presentation)**
- **Optional reading: PTX Assembly and Profiler; Floating Point Considerations; Introduction to OpenCL**
  Hardware counters, visual profiler, predicated execution, floating-point representation, flush-to-zero, denormalization, rounding, units-in-the-last-place, mixed-precision methods, OpenCL kernel structure and invocation, context, devices, command queues, memory allocation, argument construction