ECE 193: Parallel Computing
Fall 2017

Meeting time: MW 3:00-4:15.
Instructor: Joel Grodstein. Office hours: 1 hour before class or by appointment.

Prerequisites: ECE 126, graduate status or consent of the instructor.

Course Overview:
Parallel computing is a marriage of hardware and software. The hardware is of no use without software to run on it – but software writing without understanding the hardware is unlikely to perform well. We will look at concurrent and parallel computing, including multi-core CPUs and GPUs and concurrent programming. We will look at interactions between hardware and software, including using our knowledge of microarchitecture to write higher-performance and more robust parallel software.

Course Objectives:
Upon completion of the course, students will:

1. Become familiar with parallel hardware (cores, caches and memory) and software, and understand how to write efficient software based on understanding the hardware.
2. Write parallel programs using languages including the C++ threads and CUDA
3. Know how to parallelize algorithms
4. Know how to identify and optimize performance bottlenecks

Rough course sequence:

- Introduction to parallel programming and concurrence. Talk about some performance bottlenecks (memory bandwidth and CPU compute capability), and some issues involved in making a concurrent program work at all.
- C++ threads and CUDA: once over lightly. Introduce both programming languages. Compare and contrast them, using short examples.
- Review of architecture concepts. Caches (because making use of caches is key to getting high performance on a multicore processor). Out-of-order and speculative execution, branch prediction (the absence of these is a main difference between a CPU and a GPU, so we should know what they are).
- SIMD instructions. Pros and cons of SIMD vs. other approaches to parallelism. SIMD instructions to code matrix multiply, and gather/scatter problems.
- Multicore architecture. Ring caches and MESI. False sharing & CPU_breaker.
- Parallel algorithms and programming on multicore processors. Now that we understand the architecture, we will see why some algorithms run much faster than others. We'll discuss ways of getting lots of data to the cores, block-matrix algorithms, false sharing, and other issues.
- More concurrent-programming issues: mutex, semaphores, barriers.
- GPU architecture and programming. We'll discuss GPU architecture in detail, and understand why it works very well on some problems (but much less well on others). We'll code and experiment with several examples.

Grade Formula

- Quizzes – 30%
- Final or final project – 20%
- Programming assignments – 50%

Quizzes:
There will be at roughly five quizzes assigned during class. Dates will be posted on the course calendar, and communicated in class at least one week before each quiz. The lowest quiz grade will be dropped. There will be no make-up quizzes.

Programming Assignments:
There will be at roughly four programming assignments throughout the term, instructing students in high performance and parallel programming and parallel computer architecture. All assignments must be completed individually.

Final project or exam:
There will be either a final project or final exam. We will decide which partway through the course. If we do the project, then you may work in teams and choose any project that is related to the course materials. Some suggestions are:

- Program something interesting using SSE extensions.
- Read a paper on GPU acceleration of an interesting task and report on it.
- Learn to use Intel's VTune to definitively analyze your code from one of our assignments.
- Read a chapter of "A Primer on Memory Consistency and Cache Coherency" and report on it.
- Learn about the Google TPU report on it (there is a paper and a YouTube video)
- Use SetThreadAffinityMask for block matrix multiply; assign two threads in the same core to each block.
- Report on Magma (the LAPACK-equivalent library for multicore+GPU systems)

Late Assignments:
Each student will be allocated five late days per quarter that can be used for programming assignments. Please make a note on the top of the assignment if you would like to use a late day. After the allocated late days have been exhausted, the grade on a late assignment will be penalized by 10% per day. Any extensions due to extenuating circumstances (illness or family emergencies) must be arranged ahead of time with the instructor before the original due date.

Textbook and references
There is no official textbook for the course. However, there are several good references:

- "An Introduction to Parallel Programming" by Peter S. Pacheco (1st Ed 2011), online at Tisch. Good for concurrency problems , not so great for hardware, architecture. We're using C++ threads, which appeared after this book was written.
- "Computer Architecture: A Quantitative Approach," Fifth Edition, John L. Hennessy and David A. Patterson, ISBN: 978-0-12-383872-8. Online at Tisch.One of the bibles for computer micro-architecture. However, I don't think their chapter on GPUs is as clear as the rest of the book.
- Matrix Computations, Golub and Van Loan, 3rd edition. The bible of matrix math (including how to make good use of memory). Not always easy to read for a beginner. One copy on reserve in Tisch.
- Various GPU references:

- https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf White paper on a recent GPU. There are also white papers on Pascal and Tesla.
- http://docs.nvidia.com/cuda/cuda-c-programming-guide
- https://www.google.com/url?q=https%3A%2F%2Feecs.berkeley.edu%2Fresearch%2Fcolloquium%2F170315 The Google TPU
- CUDA by Example: An Introduction to General-Purpose GPU Programming, Sanders & Kantrot. A nice intro book; light on the hardware.
- Programming Massively Parallel Processors: A Hands-On Approach, 3rd edition, David Kirk and Wen-mai Hwu. A good book on both hardware and software.

Collaboration policy
Learning is a creative process. Individuals must understand problems and discover paths to their solutions. During this time, discussions with friends and colleagues are encouraged—you will do much better in the course, and at Tufts, if you find people with whom you regularly discuss problems. But those discussions should take place in English, not in code. If you start communicating in code, you're breaking the rules. When you reach the coding stage, therefore, group discussions are no longer appropriate. Each program, unless explicitly assigned as a pair problem, must be entirely your own work. Do not, under any circumstances, permit any other student to see any part of your program, and do not permit yourself to see any part of another student's program. In particular, you may not test or debug another student's code, nor may you have another student test or debug your code. (If you can't get code to work, consult a teaching assistant or the instructor.) Using another's code in any form or writing code for use by another violates the University's academic regulations. Do not, under any circumstances, post a public question to Piazza that contains any part of your code. Private questions directed to the instructors are OK. Suspected violations will be reported to the University's Judicial Officer for investigation and adjudication. Be careful! As described in the handbook on academic integrity, the penalties for violation can be severe. A single bad decision made in a moment of weakness could lead to a permanent blot on your academic record. The same standards apply to all homework assignments; work you submit under your name must be entirely your own work. Always acknowledge those with whom you discuss problems! Suspected violations will be reported to the University's Judicial Officer for investigation and adjudication. Again, be careful

Additional resources
Tufts University values the diversity of our students, staff, and faculty, and recognizes the important contribution each student makes to our unique community. Tufts is committed to providing equal access and support to all qualified students through the provision of reasonable accommodations so that each student may fully participate in the Tufts experience. If you have a disability that requires reasonable accommodations, please contact the Student Accessibility Services office at Accessibility@tufts.edu or 617-627-4539 to make an appointment with an SAS representative to determine appropriate accommodations. Please be aware that accommodations cannot be enacted retroactively, making timeliness a critical aspect for their provision. Tufts and the teaching staff strive to create a learning environment that is welcoming to students of all backgrounds. If you feel unwelcome for any reason, please let us know so we can work to make things better. You can let us know by talking to anyone on

the teaching staff. If you feel uncomfortable talking to members of the teaching staff, consider reaching out to your academic advisor, the department chair, or your dean.