**EC527: High Performance Programming with Multicore and GPUs -- Spring, 2015**

**Instructor:** Martin Herbordt, PHO 333
Office Hours: T, Th 3-5 and by appointment
Phone: x3-9850 Email: herbordt@bu.edu Web page: http://learn.bu.edu

**TFs:** As this is an advanced-undergrad/grad class with a large enrollment, the role of the TFs will be limited to grading and helping to find and solve system problems for the programming assignments.

**Mission Statement:** "Programming for performance using the capabilities of modern processors''

**Course Description (catalog):** Considers theory and practice of hardware-aware programming. Key theme is obtaining a significant fraction of potential performance through knowledge of the underlying computing platform and how the platform interacts with programs. Studies architecture of, and programming methods for, contemporary high-performance processors. These include complex processor cores, multicore processors, and graphics processors (GPUs). Labs include use and evaluation of programming methods on these processors through applications such as matrix operations and the Fast Fourier Transform.

**Prerequisites:** Computer Organization (EC413 of equivalent), programming in C, and academic maturity sufficient, e.g., to learn new programming tools from professional documentation.

**Course Motivation:**
For several decades programmers found the von Neumann (vN) model to be an adequate world view for obtaining most of the potential performance from target systems. This model is familiar: instructions are executed serially in a single stream and data are stored in a single image of memory. Instruction executions and memory accesses are assumed to be uniform with little penalty. Except for specialized processors--DSPs, Supercomputers, MPPs--good vN programming plus a good compiler meant taking advantage of most of a computer's capability.

Recent directions in processor architecture—complex memory hierarchies, superscalar and deeply pipelined CPUs, multicore, and accelerators such as SSE, GPUs and FPGAs—have made the vN approach to obtaining performance obsolete for all but the very simplest embedded processors. For many applications performance is secondary; in those cases current methods remain appropriate. But for applications requiring performance, a deeper level of machine understanding is required: the programmer must be aware of the underlying hardware at all stages of software development from algorithm selection, through coding, to interaction with system tools such as compilers and libraries, and finally debug, tuning, and maintenance.

**Texts and Organization (supplemented with additional articles, lecture notes, and tutorials):**
For complete (tentative) readings see "Readings" document.
Part 1 – Single core
This part of the course is based on sections of courses taught at CMU and ETH.

- "How to Write Fast Code," Markus Pueschel, Lecture Notes from CMU and ETH
- "How to Write Fast Numerical Code: A Small Introduction," S. Chellappa, et al.; CMU Tech Report
- *Computer Systems: A Programmer's Perspective,* Bryant & O'Hallaron, Chapters 5 and parts of Chapter 6
- Various Intel HW & SW Reference Manuals
- H&P 4th Edition -- Appendix G: (Mostly) SW methods for complex processors
- H&P 4th Edition -- Appendix F: Vector processors

Part 2 – Multicore

This part is a condensed and applied version of material from EC713 Parallel Computer Architecture.

- Computer Architecture (Chapter 4): Hennessy & Patterson 4e
- Parallel Computer Architecture (Chapter 5): D. Culler, et al.
- Parallel Programming (Chapters 2 and 3): D. Culler, et al.
- Threads Primer: A Guide to Multithreaded Programming (Chapters 2-5): Lewis & Berg
- OpenMP Lecture Notes: SCV at BU

Part 3 – GPUs

This part is based on a course taught at UIUC by Wen-mei Hwu.

- CUDA Reference Manual(s): NVIDIA
- Programming Massively Parallel Processors: Kirk & Hwu

**Course Mechanics**

- **Style:** One of the missions of this course is to be a practicum associated with the computer organization and architecture curriculum. As such we explore contemporary high-end processors in some depth and then practice using that knowledge to obtain high resource utilization with real programs. The emphasis is therefore on programming with lectures in support of the labs. Lectures will also introduce appropriate theory when necessary, especially with respect to performance evaluation.
- **Grading:** Exam: 30%
  Programming/Homework Assignments: 50%
  Final Project: 20%
  Please note that these percentages are tentative. Also, that the impact of an assignment/exam grade on the final grade depends on the expected variance in addition to the total.
- **Weekly Assignments:** Until the beginning of the project there will be weekly assignments, 10 in all (0-9). All involve programming, mostly exploring small amounts of code in great depth. Some assignments involve pencil-and-paper problems in addition to programming.
- **Late Policy:** Assignments must be submitted on time, usually on Tuesdays (by 23:59:59 local time). There is then a 20% per day penalty. You will get a total of 5 "free" late days to handle special (but common) occurrences such as illness, interviews, etc.
- **Academic Honesty versus Collaboration:** You are encouraged to work together to learn the material and to discuss approaches to solving problems. However, *you must come up with and write up the programs and other solutions on your own.*
- **Exams:** There will be at least one mid-term exam. There may also be a final exam**.** Exams are open readings and open notes.
- **Final Project:** The purpose is to add depth and to practice the concepts learned in an extended case study. Results will be written up conference paper style and presented to the class. You may work in teams of up to two students. I will consider larger groups with sufficient justification.
- **"Late" Classes:** Unfortunately, this class has been rescheduled so that it now conflicts with faculty candidate and distinguished lecture speakers. Therefore, on at least 2 days, class will be from 5:15-6:45. There may be a few more shifts -- these will be announced well in advance.

**Course Objectives**

Review

- Computer architecture including memory hierarchy and basic pipelining.

Learn about

- Various contemporary high-end processors, in particular the i7 core and memory hierarchy, multicore cache, and GPUs
- Methods of performance evaluation
- Methods of hardware-aware code development
- How to program complex hardware to obtain high utilization

Gain experience with developing efficient programs, including

- using extended instruction sets
- synchronization
- methods of parallel programming
- cache-aware optimizations
- CUDA for GPU programming.

**From the Course Requisition Form**

**Basic Goals**

1. Students should learn enough about processor architecture and programming to write fast code (code with high utilization of available resources) on contemporary processors.
2. The knowledge and experience should enable students to extend this capability to new processors and to large and varied applications.

**Detailed Goals**

Students should have a good understanding of theory and practice of

1. Measuring performance
2. Developing fast code (decomposition, mapping, load balancing)
3. Using advanced capabilities such as blocking, SIMD vector extensions, and other basic code optimizations
4. Parallel processing with small-scale (multicore) shared memory processors
5. Programming GPUs, including problems in getting good performance

Students should also develop a deeper understanding of one of the three technologies with an extended project. This will consist of examining a more complex numerical problem.

**Course Outcomes**

1. Sufficient knowledge of various processor architectures to be able to write high-performance programs
2. Basic knowledge of principles and practice of performance evaluation and writing high-performance programs with the goal of applying this knowledge to other processors.
3. Ability to use SSE instructions set extensions
4. Ability to write parallel programs using PThreads and OpenMP
5. Ability to write GPU programs in CUDA
6. Ability to formulate and design programs at a high level accounting for a target architecture