EC527: High Performance Programming with Multicore and GPUs -- Spring, 2025

Instructor: Martin Herbordt, PHO 333
Office Hours: T3-5, M,W after class, other days/times by appointment
Phone: x3-9850 Email: herbordt@bu.edu Web page: http://learn.bu.edu
TFs: Shining Yang (GTF), Vance Raiti, Sora Kakigi, Austin Jamias (UTFs)
Lab TA Hours: -- TBD Email: shiningy@bu.edu, ajamias@bu.edu, skakigi@bu.edu, vraiti@bu.edu

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 a program's 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 CPUs, and graphics processors (GPUs). Labs include use and evaluation of programming methods on these processors through application kernels such as matrix algebra and the Fast Fourier Transform.

Prerequisites: Computer Organization (EC413 or equivalent), programming in C, and academic maturity sufficient, e.g., to (i) learn new programming tools from professional documentation, (ii) design basic experiments, and (iii) perform simple data analysis, including using spreadsheets, plotting data, etc.

Course Motivation:
For several decades programmers found the von Neumann (vN) model to be an adequate worldview 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 have uniform performance, as do all memory accesses. Except for specialized processors--DSPs, Supercomputers, MPPs--good vN programming plus a good compiler meant taking advantage of most of a computer's capability.

Current directions in processor architecture—complex memory hierarchies, superscalar and deeply pipelined CPUs, multicore CPUs, and accelerators such as AVX extensions and GPUs—have made the vN approach to obtaining performance obsolete. For many applications performance is secondary; 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 and numerical analysis, through coding, to interaction with system tools such as compilers and libraries, run-time systems, and finally debug, tuning, optimization, and maintenance.

Texts and Organization (supplemented with additional articles, lecture notes, and tutorials):
For complete (tentative) readings see "Readings" document.

<u>Part 0 – Methods</u>

- Timing and Timers – See Lab 0 documentation
- Performance Models – Patterson & Hennessy 5e, Chapter 6.1, 6.2, 6.10, 6.11
- *Computer Systems: A Programmer's Perspective,* Bryant & O'Hallaron, Chapter 5.0, 5.2

<u>Part 1 – Single core</u>

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, Chapter 5 and parts of Chapter 6
- Hennessy & Patterson 4th Edition -- Appendix F: Vector processors

Part 2 – Multicore

The theory from this part is a condensed and applied version of material from EC713 Parallel Computer Architecture. The practice is from various sources.

- 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:** The primary mission of this course is how to create high performance code. Another mission is to be a practicum associated with the computer organization and architecture curriculum. For both 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(s): 35%
  Programming/Homework Assignments: 40%
  Final Project: 25%

  Please note that these percentages are tentative. Also, that the impact of an assignment/exam grade on the final grade depends on the variance in addition to the percentage.
- **Weekly Programming Assignments:** Until the beginning of the project there will be weekly assignments, 9 in all (0 - 8). All involve programming, mostly exploring small amounts of code in great depth. Some assignments include pencil-and-paper problems in addition to programming.
- **Lab Groups:** Programming assignments have been designed to be done by single students. But by popular demand, you may work in groups of at most two students.
- **Late Policy:** Assignments must be submitted on time, usually Wednesdays before class. There is a 20% per day penalty. However, you will get a total of 5 “free” late days to handle special (but common) occurrences such as illness and interviews. Otherwise the only acceptable excuses will be “uncommon” events such as long term disability or a family crisis.
- **Academic (Dis)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, or, if you are working with another student, only from you and your partner.*
- **Exams:** There will likely be two exams: one mid-term plus a final.
- **Academic Conduct Code:** Read the academic conduct code. Please note that the penalties are severe.
- **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. For the project only, you may work in teams of up to three students. Please note – the larger the group, the larger the expectations.

Course Objectives

Review

- Computer architecture including memory hierarchy and basic pipelined CPUs.

Learn about

- Various contemporary high-end processors, in particular, recent CPU cores and memory hierarchy, multicore cache, and GPUs
- Methods of performance evaluation
- Capabililities and limits of compiler optimizations
- 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 (AVX) with implicit code and intrinsics
- synchronization
- methods of parallel programming, including PThreads and OpenMP
- cache-aware optimizations
- CUDA for GPU programming (and possibly another framework such as OpenACC)

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 and analyzing performance
2. Determining effect of compiler optimizations including reading assembly language
3. Developing fast code using methods such as decomposition, mapping, load balancing, blocking, basic block optimizations, and many others
4. Using advanced capabilities such as SIMD vector extensions and use of intrinsics
5. Parallel processing with small-scale (multicore) shared memory processors with both PTreads and OpenMP
6. Programming GPUs, including dealing with the standard inhibitors to getting good performance

Students should also develop a deeper understanding of one of the three technologies (CPU, multicore, GPU) with an extended project. This will consist of examining a more complex numerical or data processing 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 AVX 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