Advanced Computer Architecture
ECE 6750 / CS 5420
Fall 2024

In compliance with Cornell University's policy and equal access laws, the instructor can discuss appropriate academic accommodations that students with disabilities may require. Requests for academic accommodations should occur during the first three weeks of the semester, except for unusual circumstances so that we can make appropriate arrangements. I encourage students to register with Student Disability Services to verify their eligibility for suitable accommodations.

Course website: Canvas

Faculty Name: José F. Martínez
Email: martinez@cornell.edu

Office Hours: **-** MoTh 1-2pm in 220 Carpenter Hall, or
- Zoom by appointment: https://fantastical.app/profmartinez/oh

Course Staff: Socrates Wong, Ph.D. TA

Credits and Credit Hour Options: 3.0 Credits (Letter grades only)

Prerequisites/Corequisites: ECE 4750/ECE 5740/CS 4420 or equivalent, or permission of instructor. Prerequisites are strictly enforced for undergraduate and MEng students. Concurrent registration with ECE 4750/ECE 5740/CS 4420 is not allowed.

Time and Location: MoWe 2:55-4:10pm in 102 Upson

Course Description
This course discusses advanced topics in computer architecture beyond the material that is covered in undergraduate courses such as ECE 4750/CS 4420. In particular, the course places special focus on multicore and multiprocessor architectures (coherence, consistency,

synchronization, interconnects, OS support, etc.), as well as advanced architecture techniques (simultaneous multithreading, speculative loads and stores, neural branch predictors, hardware resource management, memory scheduling, etc.) Students work on parallel programming assignments that emphasize hardware-aware performance optimization.

Course Objectives/Student Learning Outcomes

By the end of the course, a student will be able to:

- Design basic hardware support for multiprocessing, including but not limited to cache coherence (both snoopy- and directory-based), synchronization, memory consistency, and multiprocessor OS support
- Describe advanced microarchitecture solutions for branch prediction and memory disambiguation, including but not limited to neural branch and store set predictors
- Reason about architectural design trade-offs, especially regarding optimization for specific code patterns
- Construct efficient shared-memory parallel programming solutions to algorithmic problems
- Report quantitative results about parallel processing experiments

Course Materials

There are no required textbooks, although the following two sources are useful. The required reading for the course is at the end of this document.

Synthesis Lectures on Computer Architecture
Morgan & Claypool Publishers

Parallel Computer Architecture: A Hardware-Software Approach
by D.E. Culler and J.P. Singh – Morgan Kaufmann

Method of Assessing Student Achievement

Exams (dates TBD)

Exams are open notes and open books; however, please be aware that hoping to rely on on-the-spot browsing will probably backfire, as questions will be formulated to require some thinking.

Parallel Programming Assignments

There will be several programming assignments, which I will announce on Canvas. You will work with a partner. Solutions must be submitted by 5 pm EST on the due date, which always falls on a Friday; see Late Policy below. They must be submitted electronically in PDF format.

Peer Reviewing

All assignments will be peer-reviewed: Students will review their peers in a double-blind fashion (authors don't know who their reviewers are and vice versa), as assigned by the instructor using HotCRP, a web-based peer review service widely used in systems conferences. Grades will depend on the quality of student's work and their reviews of other people's work.

Grade Weights

Exams 50%
Programming assignments 35%
Peer-review effort 15%

I may boost your grade by up to 1/3 letter, e.g., from B+ to A-, at my discretion if I believe your contribution to the course has been remarkable in some (positive) way, e.g., very active in-class participation, or an unusually strong position paper. I will ask you to remind me of what (you believe) makes you qualify for this. Note that you can achieve the maximum grade without resorting to this aid. There is no other way of getting extra credit; in particular, there will be no extra work to boost your grade after the end of the semester.

Grading Basis

This course uses a letter grading system, which is the official grading system at Cornell University. Typically, about 20-30% of the students in this course receive a grade of A- or higher and the median is B or B+, although this may vary from semester to semester depending on the performance of the students.

Late Policy

ONE MINUTE LATE = NOT SUBMITTED = ZERO

I will not accept late submissions, nor will I grant extensions unless there is a well-justified reason (but see lifeline provision for projects below). Note that a high workload, other

assignments with similar due dates, and exams in other classes are not acceptable reasons for an extension. Planning for early submission is your best safeguard.

Lifeline provision

As an exception to the rule, and because "stuff happens," each student has one lifeline that they can use to secure a 24h extension (that's 1,440 minutes) on a homework or project submission. In the case of a group submission, no party must have used their lifeline previously, and it will use up all parties' lifelines. It may be wise to determine whether your partner(s) has (have) a lifeline well before a project deadline.

Academic Integrity

This section is particularly important; please read carefully.

The term "group" in this section refers to yourself if you work alone or to you and your partner(s) in a group project.

I expect your group's submission to be the result of your group's effort only. The use of a computer in no way modifies academic integrity standards expected under the Cornell University Code of Academic Integrity.

You are encouraged to study together and to discuss information and concepts covered in the lecture with other students. You can give "consulting" help to or receive "consulting" support from such students. However, this cooperation should never involve one group having possession of a copy of all or part of the work done by some other group, including work from previous years.

Should copying occur, both the student(s) who copied work from another student and the student(s) who provided the material will automatically receive a zero on the assignment. An extra penalty will be assessed, ranging anywhere from a deduction on the final grade to failure of the course and university disciplinary action. Please note that this implies that at no time are you allowed to grant anyone but your group partner access to your computer files. Be sure to master the use of *chmod* and *umask* before starting to work on your projects.

During exams, you must focus on your work. You are not allowed to communicate with other students, nor may you compare or borrow notes, copy from others, or collaborate in any way.

You may not submit any AI-generated code or text as your own, and you may not borrow AI-generated code or text even if you place it in quotes or cite it. However, you may use generative AI software to help you investigate a topic, find authoritative sources, or try out code segments as part of your design effort.

Please note that we routinely use plagiarism detection tools for both code and text, and this includes AI-generated work.

I urge you to read Cornell University's Code of Academic Integrity.

Required Reading Assignments

THE MULTICORE

Introduction
Amdahl's Law in the multicore era, by M. Hill and M. Marty, Tech. Rep. 2008

Cache Coherence
Using cache memory to reduce processor-memory traffic, by J.R. Goodman, ISCA 1983

A low-overhead coherence solution for multiprocessors with private cache memories, by M.S. Papamarcos and J.H. Patel, ISCA 1984

An evaluation of directory schemes for cache coherence, by A. Agarwal et al., ISCA 1988

The SGI Origin: A ccNUMA highly scalable server, by J. Laudon and D. Lenoski, ISCA 1997

Synchronization
Efficient synchronization primitives for large-scale cache-coherent multiprocessors, by J.R. Goodman et al., ASPLOS 1989

Memory Consistency
Memory consistency and event ordering in scalable shared-memory multiprocessors, by K. Gharachorloo et al., ISCA 1990

Two techniques to enhance the performance of memory consistency models, by K. Gharachorloo et al., ICPP 1991

OS Support
Scheduling and page migration for multiprocessor compute servers, by R. Chandra et al., ASPLOS 1994

THE CORE

Control Flow Speculation
Combining branch predictors, by S. McFarling, DEC WRL TR 1993

Neural methods for dynamic branch prediction, by D.A. Jiménez and C. Lin., ACM TOCS 2002

Piecewise linear branch prediction, by D.A. Jiménez, ISCA 2005

Memory Disambiguation
Memory dependence prediction using store sets, by G.Z. Chrysos and J.S. Emer, ISCA 1998

Composable Processors
Core Fusion: Accommodating software diversity in chip multiprocessors, by E. İpek, M. Kırman, N. Kırman, and J.F. Martínez, ISCA 2007

MEMORY CONTROLLERS

Self-optimizing memory controllers: A reinforcement learning approach, by E. İpek, O. Mutlu, J.F. Martínez, and R. Caruana, ISCA 2008

Improving memory scheduling via processor-side load criticality information, by S. Ghose, H. Lee, and J.F. Martínez, ISCA 2013

HARDWARE RESOURCE MANAGEMENT

Coordinated management of multiple resources in chip multiprocessors: A machine learning approach, by R. Bitirgen, E. İpek, and J.F. Martínez, MICRO 2008

XChange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures, by X. Wang, and J.F. Martínez, HPCA 2015

PARTIES: QoS-aware resource partitioning for multiple interactive services, by S. Chen, C. Delimitrou, and J.F. Martínez, ASPLOS 2019