

**CS 380P Parallel Systems**

Lectures:	MW 2:00-3:30 MEZ 1.120
Unique number:	54340
Class Web Page:	<a href="http://www.cs.utexas.edu/users/lin/cs380p/">http://www.cs.utexas.edu/users/lin/cs380p/</a>

	Instructor	Teaching Assistant	Admin. Assistant
Name:	Calvin Lin	Sibi Govindan	Gem Naivar
Email:	<a href="mailto:lin@cs.utexas.edu">lin@cs.utexas.edu</a>	<a href="mailto:sibi@cs.utexas.edu">sibi@cs.utexas.edu</a>	<a href="mailto:gem@cs.utexas.edu">gem@cs.utexas.edu</a>
Office:	ACES 3.424	ENS 31NQ	ACES 3.422
Office Hours:	WF 3:30-4:30	T 1:30-2:30, Th 10:00-11:00	

**Course Objectives.** This course will take a broad look at parallel computing, touching on topics in computer architecture, programming models, languages, performance evaluation, algorithms, and applications. We will consider both small-scale and large-scale parallelism, and we will consider both traditional (scientific) and modern (graphics) applications. The specific goals are (1) to learn fundamental principles about parallel computing, (2) to be able to write a parallel program and obtain good performance, (3) to be informed of trends in the area, and (4) to explore in depth one aspect of parallel computing in a project of your choosing.

**Text.** *Principles of Parallel Programming.* Lin and Snyder. Addison-Wesley, 2008.

**Prerequisites.** Familiarity with computer systems will be useful.

**Expectations.** There will be three small-to-modest programming assignments and one course project. The projects can be done in teams on any of a variety of topics. You will also read assorted papers and sections of the text book, and a few of you may be asked to present your project to the class. There will be no exams.

**Content.** Topics will include the following.

- Motivation
  - Traditional motivations for parallelism
  - The changing role of parallelism
- Evaluation
  - Sources of inefficiency
  - Metrics: Execution time, speedup, etc.
  - Throughput vs. latency
  - Scalability: massive parallelism, Amdahl's Law, Gustafson's Law

- Architectures
  - Parallel architectures
  - Trends in architectures, including CMPs, GPUs, and Grids
- Parallel Programming: Low Level Approaches
  - Threads
  - Message passing
  - Issues in scalability and portability
  - Transactional Memory
- Parallel Programming: Higher Level Approaches
  - ZPL
  - Automatic Parallelization and HPF
  - Chapel
  - MapReduce
- Parallel Algorithms
  - Models of parallelism: PRAM, CTA
- Applications

## Programming Assignments

We will have roughly three programming assignments, with the following tentative schedule. Each assignment will actually be broken into two pieces. For the first part of each assignment, you may work with others, as the goal is to familiarize you with the language and the environment. (I'm still considering whether to let you work in teams of two for the second part of each assignment.)

Programming Assignment Due Dates:

February 11	Programming with Threads
February 25	Programming with Message Passing (MPI)
March 11	Programming with Chapel

## The Project

The projects may take various forms and may involve design, implementation, experimentation, analysis, or evaluation. One broad class of projects (listed as the first bullet below) is to implement some non-trivial parallel program, but more research-oriented projects are possible as well. Below is a list of potential ideas, but my hope is that most of you will come up with other ideas of your own.

- Write a parallel program to solve some problem in the language of your choice. Hopefully, the problem is something that you're interested in. Ideally, you will also learn something from this project, perhaps one or more of the following:
  - What factors make it difficult to produce a correct program?
  - What factors make it difficult to achieve good performance?

- Does your solution require any of the following complications?
  - \* Dynamic parallelism
  - \* Pointer-based data structures
  - \* Task parallelism
  - \* Hierarchical data structures
  - \* Novel algorithms

Think about how you will evaluate performance.

Group projects that could compare different languages, approaches or hardware might be particularly interesting.

- CMP's: Take an application that was designed for a multi-chip parallel computer and retarget it for a CMP. Will drastic changes be required to obtain good performance out of the memory system?
- Rewrite an existing Astrophysics simulation in C++ (it's currently in Fortran 90), using a better domain decomposition than is currently used. We can get you the original code, along with some expertise about what it does.
- Transactional Memory vs. Lock Inference. TM provides optimistic concurrency. Locks provide pessimistic concurrency. We can envision a system that supports both for different parts of a program. Evaluate the performance difference for these two approaches using an existing TM infrastructure. (We can point you to some infrastructure.)
- CMP Simulation: Use Liberty to build a generic CMP simulator.
- Supercomputer Simulation: Modify a locally developed simulator for supercomputing clusters so that it realistically simulates network and file system contention.
- Massive Parallelism: Take an existing MPI application and simulate its performance on a cluster that has tens of thousands of nodes. Are significant changes required? Will we need to restructure the program's communication to be more hierarchical?
- GPU Programming: Using the CUDA language, implement and evaluate the performance of some parallel applications running on a GPU. What types of problems do the GPU present?
- Implement a ray tracer in CUDA for a GPU. A multi-person group could implement a ray tracer in both CUDA and in Pthreads and try to make some sort of meaningful comparison of the two in terms of both programmability and performance. There are many issues to resolve when comparing performance across different architectures.
- Assist Threads: Assist threads can be used to improve reliability. The basic idea is to create one fast thread and one slower thread that performs any necessary correctness checks. What kind of faults (software bugs, transient hardware faults, or hardware design faults) can be detected using assist threads? What are the performance implications of such an approach? Can we generate such threads semi-automatically using a compiler?

The various components of the project are listed below, and I am prepared to offer guidance and help at every step.

1. Pre-proposal: This is a rough idea for a project, including ideas about the project's scope, nature and personnel. This is essentially a starting point for defining the project, and may even be as vague as "I'm interested in parallel debugging, and I only work alone because I'm a sociopath."
2. Proposal: This is a well-thought out plan of action that includes target deadlines for sub-pieces of the project. Some non-trivial amount of thought and/or background reading will be required to move from the pre-proposal to this proposal.
3. Checkpoints: These checkpoints are project-specific goals intended to keep the projects on track.

4. Final Report: The final report will be evaluated based on its presentation as well as its contents.

The tentative schedule for these parts is shown below:

Project Checkpoints:

February 9	Project preproposal
February 23	Project proposal
March 25– April 29	Various project-specific checkpoints
May 8	Reports due

## Grading

Assignments:	40%
Project:	50%
Quizzes and Class Participation:	10%

**Web page.** All announcements and handouts will be placed on the class web page or on the TA's web page.

**Office Hours/Open Door Policy.** My door is usually open when I am in, so please feel free to come to my office whenever my door is open. (If the door is slightly cracked, consider it open.) In the rare case that my door is completely closed, **please do not knock** unless you have an appointment with me.

**Administrative Assistant.** My administrative assistant is Gem Naivar, whose office is across the hall from mine. *See Gem when you need to retrieve unclaimed assignments or handouts.*

## Scientific Ethics

Basically, we'll apply the usual standards of the scientific community: collaboration is encouraged but must be cited. Cheating, including plagiarism, will lead to failure of the course.