CS 735/835: Introduction to Parallel and Distributed Programming

Catalog description

Programming with multiple processes and threads on distributed and parallel computer systems. Introduces programming tools and techniques for building applications on such platforms. Course requirements consist primarily of programming assignments. Prereq: CS 520.

Overview

Parallel and distributed computing have become mainstream and demand to be covered in both the undergraduate and graduate curricula. Revised curricula from the NSF (2012)—“we must now prepare [students] for the very dynamic world of parallel and distributed computing”—and the ACM (2013)—“parallel and distributed computing has moved from a largely elective topic to become more of a core component of undergraduate computing curricula”—have given a more prominent place to these topics.

This course is an introduction to parallel and distributed computing from a programmer’s standpoint. It covers fundamental concepts (e.g., locking, atomicity, parallelism) and a few more advanced ones (e.g., non-sequentially consistent memory models, non-blocking algorithms) through examples. Students complete several programming assignments and a multi-threaded client-server project. The course uses Java, its java.util.concurrent library and its support for Remote Method Invocation, but similar concepts are found in most modern programming environments.

The course does not assume prior knowledge of Java threads. However, it assumes that students can read and write Java code, understand basic concepts of object-oriented programming and can implement, test and debug a medium-sized Java application. Students should also be at least familiar with the concept of thread (or process) and parallelism (through an operating systems course, for instance). Coursework is identical for graduate (CS-835) and undergraduate (CS-735) students, but graduate students need to achieve at least a B− grade to pass the course.

Attributes

- This course is one of the CS electives designated as implementation intensive.
- This course can be taken as CS735H by honors students.

Outcomes

Primary

- **programming**: multi-threaded programming, concurrency support at the language and library levels, distributed systems (sockets, client-server paradigm, serialization, remote method invocation).
- **algorithms**: concurrent algorithms, synchronization, non-blocking algorithms.

Secondary

- **system & hardware**: performance of multi-threaded applications, low-level costs of threads, memory models and non-sequential consistency.
Evaluation

Five programming assignments (50%), one project (30%) and one exam (20%).

Minimum score for each grade: A: 90, A−: 87, B+: 83, B: 80, B−: 77, C+: 73, C: 70, C−: 67, D+: 63, D: 60, D−: 57.

Topics

- **Basic concepts of concurrent and distributed programming:**
  - parallelism, threads, scheduling, nondeterminism
  - blocking operations, deadlocks, livelocks, timeouts
  - atomicity, synchronization, data races
  - memory models, memory barriers, visibility
  - serialization, remote procedure call
- **Client-server paradigm:**
  - sockets, socket servers
  - Remote Method Invocation
- **Synchronizers:**
  - locks, semaphores, latches, barriers, conditions, futures
  - blocking and bounded queues
- **Concurrent data structures:**
  - `synchronized` vs `concurrent`, parallel operations
  - atomic operations, client-side locking
  - non-blocking algorithms
- **Executor services:**
  - tasks, futures, thread pools, failures, cancellations, shutdown
  - parallelizing recursive computations
  - timers and scheduled executors
- **Engineering concurrent programs:**
  - correctness, thread-safety
  - state, invariants, sharing, locality, mutability, immutability
  - decomposition, tasks, execution services
  - thread-safe and concurrent data structures
  - interrupts, poison pills, cancellation, termination, abortion, non-interruptible blocking
  - non-blocking functional concurrency (futures as monads)
  - actors
- **Performance:**
  - Amdahl’s law
  - contention, thread creation, parking/unparking, I/O
  - throughput, responsiveness
  - resource management
- **Programming language support (Java):**
  - `synchronized` blocks, `final` fields, `volatile` fields
  - `wait`, `notify`, `notifyAll`
  - `java.lang.ThreadLocal`
  - `java.util.concurrent` (usage and implementation)
  - `java.net`, `java.io.Serializable`, `java.rmi`
- **Advanced topics:**
  - `compare-and-set`, non-blocking algorithms
  - Java Memory Model
  - `java.util.concurrent.lock`
  - `java.util.concurrent.atomic`
  - building custom synchronizers
Textbooks

Required:


Additional (for students new to Java):