Lab 4 Review
Final Exam Review

Christopher Mitchell, Ph.D.
cmitchell@cs.nyu.edu || http://z80.me
Outline

- Lab 4 Assignment
- Final Exam Topic Review
  - Lectures
  - Labs
Lab 4 Assignment

• Genetic Algorithm Maze Solving
• Due Tuesday, August 13, 2019
  • Day of final exam; accepted until midnight Wednesday
• Adapts your Lab 3, adds different types of threads for solving mazes
Lab 4: Maze Representation

\[
\begin{bmatrix}
1,1,1,1,1,1,1,1, \\
1,0,0,0,0,0,0,0, \\
1,0,1,1,1,1,1,1, \\
1,0,0,0,0,0,0,0, \\
1,0,1,1,1,1,1,1, \\
1,0,0,0,0,0,0,0, \\
1,1,1,0,1,1,1,1, \\ 
\ldots
\end{bmatrix}
\]
Lab 4: Fitness & State

• Energy minimization
  • Cost: taxicab distance from ending point to desired end point

• State
  • Population: Candidate “genomes”: 5 directions
  • Offspring: good candidates waiting to be mutated
  • Futility Counter: thread-safe counter
Lab 4: Threads

- Mixer Threads
  - Splice genomes from Population, add to Offspring

- Mutator Threads
  - Pull an offspring from Offspring
  - Potentially mutate the offspring
  - Compute fitness
  - Add to population
  - Truncate population
  - Update futility counter
1. The Multicore Revolution
2. C++ Crash Course
3. Parallelism, Concurrency, and Performance
4. Programming Models
5. Understanding Hardware
6. Mutexes, Semaphores, and Condition Variables
7. Reader-Writer Locks
8. Barriers and Thread Pools
9. Synchronized Structures I-III
10. Multicore Correctness & Performance
11. Heterogeneous Multicore
12. Work Distribution
13. Transactional Memory
Lecture 1: The Multicore Revolution

• Do know:
  • The history of CPUs, especially trends in transistors, area, power usage, number of cores, and capabilities
    • Why we stopped making CPUs faster and started making them more parallel instead
  • What an assembly program and an assembly instruction is
  • How pipelining works and what it’s for
    • Pipeline acceleration features, pipeline hazards
  • Why we need parallel programming

• Don’t:
  • Try to memorize exact transistor counts, names, years, and other minutiae that you could just look up in a book. This isn’t a test of your ability to memorize.
Lecture 2: C++ Crash Course

- Memory
- Classes and structs
  - Visibility, inheritance, and polymorphism
- Data structures: Strings, Vectors, Maps, Lists
- Basic templates
- Const correctness
Lecture 3: Parallelism, Concurrency, and Performance

• The difference between parallelism and concurrency
• Linearizability and serializability
• What Amdahl’s law is, means, and how to apply it
• How to model a program as a DAG
  • How to compute work, span, and parallelism
• Types of parallelism
Lecture 4:
Programming Models

• Flynn classifications, what they mean, and what SIMD, MIMD, SISD, and MISD.

• Parallel programming models and their properties, including (specifically) PRAM, LogP, and BSP

• Brent’s Theorem
Lecture 5: Know Your Hardware

- Do Know:
  - What the memory wall is, what a von Neumann architecture is, and what the bottleneck in a von Neumann architecture is.
  - What spatial and temporal locality are
  - Why we need CPU cache, how it works, and its history in brief
  - Consistency and coherence in CPU caches, including the protocols we discussed

- Don’t memorize:
  - Number of AGUs in Intel Haswell architecture
  - The exact MESI coherence protocol (but do understand why each transition make sense and where each comes from!)
Lecture 6: Parallel Programming Primitives

• How a program model needs to define running tasks
• What a coroutine is and how it works
• The basics of the fork/join paradigm
• Types of synchronization and communication: message passing and shared memory
• Threads: What they are (and are not)
  • The stack, the heap, and threads
  • What data races, critical sections, and locks are
• What a mutex is and how to use it
• How we built a simple concurrent queue with locks
Lecture 7: Mutual Exclusion

- Peterson’s Algorithm
  - How it works
  - Why it’s the minimal implementation

- Filter Algorithm
  - How it works
  - How it relates to Peterson’s Algorithm
Lecture 8: Semaphores and Condition Variables

- How semaphores work
  - How to make a mutex with semaphores
  - The producer-consumer problem
- How condition variables work
  - How to make a condition variable with semaphores
- What a lost wakeup condition is, and how to fix it
Lecture 9: Reader-Writer Locks

- Do know:
  - What reader-writer locks are, how they help performance, and how to use them
  - Review our evolution of creating a reader-writer lock: the naïve design, solving writer starvation, then solving reader starvation
  - How pthread reader-writer locks are used
  - How to build data structures with reader-writer locks

- Don’t memorize
  - The exact code for our reader-writer lock implementation
Lecture 10: Barriers and Thread Pools

• Do know:
  • What barriers are
  • How our implementation of barriers with semaphores works
  • The basics of thread pools (see also lecture 16)

• Don’t memorize:
  • The exact code for our barrier implementation
Lecture 11: Concurrent Queues

• How to build a concurrent queue
  • What linearizability is and how it helps
    • How a sentinel gives a locking or lock-free queue linearizability
  • How simultaneous enqueueing and dequeuing works in a locked, sentinel-based concurrent queue

• Making the queue lock-free
  • What CAS is, and how to use it
  • How we maintain linearizability

• What the ABA problem is, and how to solve it
Lecture 12: Concurrent Ordered Lists & HashTables

- How a concurrent ordered list can be built
  - Locking version: how to use lock coupling
  - How to make insertion and deletion lock-free

- How a concurrent hash table can be built
  - How ordered lists can be turned into a hash table
  - What a split ordered list is
  - How split ordered lists make concurrent resizing easy
Lecture 13: Multicore Correctness

• Know about the different types of real-world multithreaded programming bugs, and what each one is.
  • Remember that many bugs can fit into multiple categories!

• Software approaches to detecting concurrency bugs
  • How each approach works, pros and cons
  • What a lockset is, and how it helps

• Preventing concurrency bugs: understand the argument for deterministic execution (and why it’s hard in practice)
Lecture 14: Multicore Performance Evaluation

• Do:
  • Understand why it might be important to measure latency and/or throughput
    • How do more cores or faster cores affect each one?
  • Understand the relationship between execution time, clocks per instruction, instruction count, and cycle time
  • Know what an ISA is, and how different CPUs with the same ISA or different ISAs relate to each other
  • Understand why it may be hard to get a single value for how long a multicore program takes to run
  • Understand ways to measure multicore program throughput and turnaround time
  • Know why benchmarks are important
  • Understand why multicore program bottlenecks can be time-dependent

• Do not:
  • Memorize the notation for multicore program throughput and turnaround time: understand why that notation is used instead
  • Memorize the applications in each benchmark
Lecture 15: Heterogeneous Multicore

- Understand how to classify heterogeneous multicore CPUs based on the number of core types and variation in ISA types
- Understand how heterogeneous multicore CPUs can complicate scheduling
- Remember properties of a few interesting real-world and research heterogeneous multicore CPUs
  - Understand how the Cell CPU schedules jobs on SPEs
  - Understand FUSE and SPLIT in the Core Fusion
  - Understand caching and communication challenges
- Understand how you might consider the problem of load balancing (scheduling) for heterogeneous multicores, and how this applies to homogeneous multicores
  - Thread costs, thread dependencies, and locality
Lecture 16: Work Distribution

- Work stealing
- Work dealing
- Lock-free work stealing with a double-ended queue
  - Do know how it works and what operations it has
  - Do not memorize the lock-free implementation
  - Review how it fixes the ABA problem
Lecture 17: Transactional Memory

- Understand the properties of transactions (for databases and transactional memory)
  - What ACID is and what it provides

- How transactions can be used in C++
  - The difference between atomic and relaxed transactions

- What optimistic and pessimistic concurrency control are, and the properties of each
  - How to detect and handle conflicts

- How software transactional memory works
- How hardware transactional memory works
- The pros and cons of HTM and STM
Lab Review

• Pro tip: review your write-ups and lab notes, not the lab assignments. Sample questions to ask yourself:

• Lab 1: What were the limitations of using a lock to maintain correctness in a simple concurrent queue? A key-value store?

• Lab 2: How did you implement a thread worker pool? How else could you have done it? How did you maximize performance?

• Lab 3: How were you able to measure performance? How does performance measurement affect the measured performance (and how can you minimize this)? What are the relative merits of reporting median, average, minimum, and maximum latency? Throughput?

• Lab 4: How did having two types of worker threads (and the ratios of those threads) contribute to performance? How did synchronization overhead affect performance?