Lecture 24: Shared Memory Concurrency

CS 62
Fall 2016
Kim Bruce & Peter Mawhorter

Some slides based on those from Dan Grossman, U. of Washington

Exponential Growth

- "How many times would I have to fold a sheet of paper for the height of the folded paper to reach the moon?"
- Human beings have terrible intuition for exponential growth. If I asked you how many times you would have to fold a single sheet of US Letter paper to reach the moon, it would be difficult to intuitively comprehend that it only takes twenty folds to reach Mount Everest, forty-two folds to the moon, and fifty to reach the sun.

Uday I/O

For Lab

- Will be using in-line tools for Java
  - Must do the reading before lab!!!!!
- If want to use your Mac
  - type: "whereis java" at command line
  - If no response then must download xcode
  - Once downloaded, in preferences, select Downloads
    - Select "Command Line Tools" and click "install"
- If want to use Windows
  - Install Cygwin (with vim package) & puTTY

Assignment

- AI'ish program to play simple chess-like game, Hex-A-Pawn.
- Build game tree
  - Players move from root to leaves (win/lose configs)
- Smart Player:
  - Trim sub-tree corresponding to last move when make a losing move.

A Last Example: Sorting

- Quicksort, sequential, in-place, expected time $O(n \log n)$
  - Pick pivot elt $O(1)$
  - Partition data into $O(n)$
    - A: less than pivot
    - B: pivot
    - C: greater than pivot
  - Recursively sort A, C $2*T(n/2)$
    - Now do in parallel, so $T(n/2)$
    - $n + n/2 + n/4 \ldots = 2n$, which is $O(n)$
  - With work, can improve more and get $O(\log^2 n)$

Parallel Streams in Java 8

Streams in Java 8

- (Lazy) Streams added in Java 8 to enable simpler list processing
  - Similar to functional languages
- Example:
  - names.stream().filter(name -> name.startsWith("B")
    .count()
  - Returns count of number of elements of names starting with "B"
  - Compare with how write with loops.
  - arl.stream().reduce(0,((m,n) -> m+n));

Stream Operations

- Construct: Most collection classes have stream() method
- Filtering Operations:
  - Stream<T> filter(Predicate<T> f)
  - Stream<T> distinct()
  - Stream<R> flatMap(Function<T,Stream<R>> f)
- Terminal Operations:
  - int count()
  - boolean allMatch(Predicate<T> f) anyMatch

Parallel Streams

- Stream<T> parallelStream()
- Tries a divide and conquer approach to solving problem.
  - Requires no explicit effort by programmer if data structure set up properly (Spliterator)

Shared Memory Concurrency

Sharing Resources

- Have been studying parallel algorithms using fork-join
  - Reduce span via parallel tasks
- Algorithms all had a very simple structure to avoid race conditions
  - Each thread had memory "only it accessed"
    - Example: array sub-range
  - On fork, "loaned" some of its memory to "forkee" and did not access that memory again until after join on the "forkee"

But ...

- Strategy won't work well when:
  - Memory accessed by threads is overlapping or unpredictable
  - Threads are doing independent tasks needing access to same resources (rather than implementing the same algorithm)
- How do we control access?

Concurrent Programming

- Concurrency: Allowing simultaneous or interleaved access to shared resources from multiple clients
- Requires coordination, particularly synchronization to avoid incorrect simultaneous access: make somebody block
  - join is not what we want
  - block until another thread is "done using what we need" not "completely done executing"

Non-Deterministic Computation

- Even correct concurrent applications are usually highly *non-deterministic*: how threads are scheduled affects *what* operations from other threads they see and *when* they see them.
- Non-repeatability complicates testing and debugging

Examples

- Multiple threads:
  - Processing different bank-account operations
    - What if 2 threads change the same account at the same time?
- Using a shared cache of recent files
  - What if 2 threads insert the same file at the same time?
- Creating pipeline w/ queue for handing work to next thread in sequence?
  - What if enqueuer and dequeuer adjust a circular array queue at the same time?

Threads again?!?

- Not about speed, but
  - Code structure for responsiveness
    - Example: Respond to GUI events in one thread while another thread is performing an expensive computation
  - Processor utilization (mask I/O latency)
    - If 1 thread "goes to disk," have something else to do
  - Failure isolation
    - Convenient structure if want to interleave multiple tasks and don't want an exception in one to stop the other

Sharing is the Key

- Common to have:
  - Different threads access the same resources in an unpredictable order or even at about the same time
    - But program correctness requires that simultaneous access be prevented using synchronization
  - Simultaneous access is rare
    - Makes testing difficult
    - Must be much more disciplined when designing / implementing a concurrent program
    - Will discuss common idioms known to work

Canonical Example

- Several ATM's accessing same account.
  - See ATM2