Lecture 21: Parallelism & Concurrency

CS 62
Fall 2016
Kim Bruce & Peter Mawhorter

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

Parallelism & Concurrency

• Single-processor computers going gone away.
• Want to use separate processors to speed up computing by using them in parallel.
• Also have programs on single processor running in multiple threads. Want to control them so that program is responsive to user: Concurrency
• Often need concurrent access to data structures (e.g., event queue). Need to ensure don't interfere w/each other.

History

• Writing correct and efficient multithread code is more difficult than for single-threaded (sequential).
• From roughly 1980-2005, desktop computers got exponentially faster at running sequential programs
  • About twice as fast every 18 months to 2 years

Darwin

• Should have made significant progress by now!
• Don’t forget to submit species program (using standard commands)
More History

- Nobody knows how to continue this
- Increasing clock rate generates too much heat
- Relative cost of memory access is too high
- Can keep making “wires exponentially smaller” (Moore’s “Law”), so put multiple processors on the same chip (“multicore”)
- Now double number of cores every 2 years!

What can you do with multiple cores?

- Run multiple totally different programs at the same time
  - Already do that? Yes, but with time-slicing
- Do multiple things at once in one program
  - Our focus – more difficult
  - Requires rethinking everything from asymptotic complexity to how to implement data-structure operations

Parallelism vs. Concurrency

- Parallelism:
  - Use more resources for a faster answer
- Concurrency
  - Correctly and efficiently allow simultaneous access
- Connection:
  - Many programmers use threads for both
  - If parallel computations need access to shared resources, then something needs to manage the concurrency
Analogy

• Typical CS1 idea:
  • Writing a program is like writing a recipe for one cook who does one thing at a time!

• Parallelism:
  • Hire helpers, hand out potatoes and knives
  • But not too many chefs or you spend all your time coordinating (or you’ll get hurt!)

• Concurrency:
  • Lots of cooks making different things, but only 4 stove burners
  • Want to allow simultaneous access to all 4 burners, but not cause spills or incorrect burner settings

Models Change

• Model: Shared memory w/explicit threads

• Program on single processor:
  • One call stack:
    • each stack frame holds local variables and references to parameters
  • One program counter (current statement executing)
  • Static fields
  • Objects (created by new) in the heap (nothing to do with heap data structure)

Multiple Threads/Processors

• New story:
  • A set of threads, each with its own call stack & program counter
  • No access to another thread's local variables
  • Threads can (implicitly) share static fields / objects
  • To communicate, write somewhere another thread reads

Shared Memory

Threads, each with own unsbared call stack and current statement (pc for “program counter”) local variables are primitives/null or heap references

Heap for all objects and static fields
Other Models

- **Message-passing:**
  - Each thread has its own collection of objects. Communication is via explicit messages; language has primitives for sending and receiving them.
  - Cooks working in separate kitchens, with telephones.

- **Dataflow:**
  - Programmers write programs in terms of a DAG and a node executes after all of its predecessors in the graph.
  - Cooks wait to be handed results of previous steps.

- **Data parallelism:**
  - Have primitives for things like “apply function to every element of an array in parallel.”

Parallelism in Java

Parallel Programming in Java

- **Creating a thread:**
  1. Define a class `C` extending `Thread`
     - Override `public void run()` method
  2. Create object of class `C`
  3. Call that thread’s `start` method
     - Creates new thread and starts executing `run` method.
     - Direct call of `run` won’t work, as just be a normal method call.
     - Alternatively, define class implementing `Runnable`, create thread w/it as parameter, and send start message.
       
       *Allows class to extend a different one.*

Parallelism Idea

- **Example: Sum elements of an array**
  - Use 4 threads, which each sum 1/4 of the array

- **Steps:**
  - Create 4 thread objects, assigning each their portion of the work.
  - Call `start()` on each thread object to actually run it.
  - Wait for threads to finish.
  - Add together their 4 answers for the final result.
First Attempt

class SumThread extends Thread {
    int lo, int hi, int[] arr;//fields to know what to do
    int ans = 0; // for communicating result
    SumThread(int[] a, int l, int h) { ... }
    public void run(){ ... }
}

int sum(int[] arr){
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++){// do parallel computations
        ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
        ts[i].start(); // use start not run
    }
    for(int i=0; i < 4; i++) // combine results
        ans += ts[i].ans;
    return ans;
}

What’s wrong?

Correct Version

class SumThread extends Thread {
    int lo, int hi, int[] arr;//fields to know what to do
    int ans = 0; // for communicating result
    SumThread(int[] a, int l, int h) { ... }
    public void run(){ ... }
}

int sum(int[] arr){
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++){// do parallel computations
        ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
        ts[i].start(); // start not run
    }
    for(int i=0; i < 4; i++)// combine results
        ts[i].join(); // wait for helper to finish!
    ans += ts[i].ans;
    return ans;
}

See program ParalledSum

Thread Class Methods

- void start(), which calls void run()
- void join() — blocks until receiver thread done
- Style called fork/join parallelism
  - Need try-catch around join as it can throw exception
  - InterruptedException
- Some memory sharing: lo, hi, arr, ans fields
- Later learn how to protect using synchronized.

Actually not so great.

- If do timing, it’s slower than sequential!!
- Want code to be reusable and efficient as core count grows.
  - At minimum, make #threads a parameter.
- Want to effectively use processors available now
  - Not being used by other programs
  - Can change while your threads running