Parallelism & Concurrency

CS 62 - Fall 2015
Michael Bannister

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

This Week's Assignments

- Lab: Java without Eclipse and Darwin Tournament
  - Submit your species by 11:59pm on Tuesday
- Assignment: HexaPawn
  - A simple tree based AI

Parallelism & Concurrency

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.

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

Models Change

- Model: Shared memory w/explicit threads
- Program on single processor:
  - One call stack (w/ each stack frame holding local variables)
  - One program counter (current statement executing)
  - Static fields
  - Objects (created by new) in the heap (nothing to do with heap data structure)

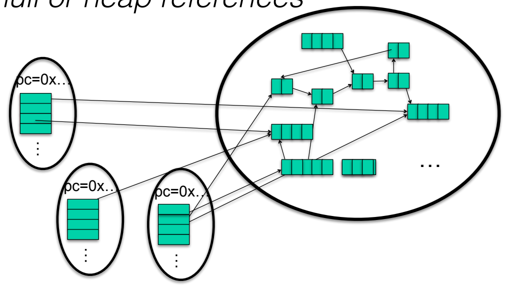
Multiple Threads & Processors

- 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 to shared memory another thread reads from shared memory

Shared Memory

Threads, each with own unshared call stack and current statement (pc for "program counter") local variables are numbers/null or heap references

Heap for all objects and static fields



Parallelism in Java

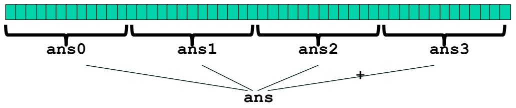
Parallel Programming in Java

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

Parallelism Example



- 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(){ … }
}
```

What's wrong?

```
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;
}
```

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 ParallelSum

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.