Lecture 28: Parallelism II

CS 62
Spring 2018
Alexandra Papoutsaki & William Devanny

Some slides based on those from Dan Grossman, U. of Washington
Example behind Parallelism Idea

- Method to calculate sum of 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
How to Create a Thread in Java

1. Define class `C` extends `Thread`
   - Override `public void run()`
   - `Thread` in `java.lang`

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, similarly to the issue as `paint-repaint`.
   - Alternatively, define class implementing `Runnable`, create thread with it as parameter, and send start message
     - Allows class to extend a different one.
First Attempt

class SumThread extends Thread{
    int lo, int hi, int[] arr
    int ans = 0; // for communicating result
    SumThread(int[] a, int l, int h) {
        lo=l; hi=h; arr=a;}
    public void run(){
        for(int i=lo; i < hi; i++) ans += arr[i];}
}
//some other class
static 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;
}
(Semi) Correct Version

```java
class SumThread extends Thread {
    int lo, int hi, int[] arr
    int ans = 0;
    SumThread(int[] a, int l, int h) { ... }
    public void run(){ ... }
}
//some other class
static int sum(int[] arr){
    int len = arr.length;
    int ans = 0;
    SumThread[] ts = new SumThread[4];
    for(int i=0; i < 4; i++){
        ts[i] = new SumThread(arr,i*len/4,(i+1)*len/4);
        ts[i].start();
    }
    for(int i=0; i < 4; i++){
        ts[i].join(); // wait for helpers to finish!
        ans += ts[i].ans;
    }
    return ans;
}
```
Thread class methods

- `void start()`, which calls `void run()`
- `void join()` which blocks until receiver thread is done
  - Style called fork/join parallelism
  - It needs a try-catch around `join` as it can throw `InterruptedException`
- Some memory sharing:
  - `lo`, `hi`, `arr` fields written by “main” thread, read by helper thread
  - `ans` field written by helper thread, read by “main” thread
- Later, we will learn how to protect data (race conditions) using synchronized
Great, right? Actually, no!

- If we time it, it’s slower than sequential!!
- We want our code to be reusable and efficient as core count grows (“forward-portable”).
  - At minimum, make #threads a parameter (e.g., in the sum method)
- Want to effectively use processors available now
  - Not being used by other programs or threads in your program
  - Can change while your threads are running
Problem

• Suppose we have a computer with 4 processors and a problem of size $n$
  • We can solve the problem with 3 processors, each taking time $t$ on $\frac{n}{3}$ elements.

• Suppose linear in size problem:
  • We want to use all 4 processors, but one is busy playing music
  • First 3 threads run, but $4^{th}$ waits
    • First 3 threads scheduled and take time $\left(\frac{n}{\frac{4n}{3}}\right) \cdot t = \frac{3}{4}t$
    • After first 3 are finished, run $4^{th}$ which takes another $\frac{3}{4}t$
    • Total time ends up $\frac{3}{4}t + \frac{3}{4}t = 1.5t$
    • Runs 50% slower than with 3 threads!
More problems

• Subproblems can take significantly different amounts of time
  • Apply method $f$ to every array element, but maybe $f$ is much slower for some data items. e.g., is a large integer prime?
  • If unlucky, all slow operations may be assigned to the same thread
    • Certainly, won’t see $n$ speedup with $n$ threads
    • May be much worse, due to load imbalance
Toward a solution

• To avoid having to wait too long for any one thread, instead create lots of threads, far more than #cores
• Schedule threads as processors become available.
• If a thread is very slow, many others will get scheduled on other processors while that one runs.
• Will work well if the slow thread is scheduled relatively early
Divide and Conquer

1. Divide problem into pieces recursively:
   - Start with full problem at root - Halve and make new thread until size is at some cutoff

2. Combine answers in pairs as we return from recursion
   - If have numProc processors then total time $O\left(\frac{n}{\text{numProc}} + \log n\right)$
In practice

• Creating so many threads and synchronizing their communication swamps savings

• Instead, use sequential cutoff about 500-1000
  • Eliminates almost all the recursive thread creation (bottom levels of tree)
  • Exactly like quicksort switching to insertion sort for small subproblems, but more important here

• Don’t create two recursive threads: create one thread and do the other piece of work “yourself”
  • Cuts number of threads in half
ForkJoin Framework to the rescue

- Java’s threads are too heavyweight
- ForkJoin Framework addresses the need for divide-and-conquer fork-join parallel programming
- Part of Java 7
Java Threads VS ForkJoin

- Create a ForkJoinPool
- Don’t subclass `Thread` → Subclass `RecursiveTask<V>`
- Don’t override `run` → Do override `compute`
- Do not use an `ans` field → Do return a `V` from `compute`
- Don’t call `start` → Do call `fork`
- Call `join` that returns answer
- To optimize, call `compute` instead of `fork` (rather than `run`)
Getting good results in practice

• Sequential threshold
  • Library documentation recommends doing approximately 100-5000 basic operations in each “piece” of your algorithm

• Library needs to “warm up” – May see slow results before the Java virtual machine reoptimizes the library internals

• Wait until your computer has more processors
  • Seriously, overhead may dominate at 4 processors, but parallel programming is likely to become much more important
Examples

• Maximum or minimum element
• Is there an element satisfying some property (e.g., is there a 47)?
• Left-most element satisfying some property (e.g., first 47)
• Smallest rectangle encompassing a number of points
• Counts; for example, number of strings that start with a vowel
• Are these elements in sorted order?
• Create a Histogram of test results from a much larger array of actual test results
CPU vs GPU

From Mythbusters:

https://www.youtube.com/watch?v=-P28LKWTzrl&feature=youtu.be

In a bit more detail:

https://www.youtube.com/watch?v=1kypaBjJ-pg