Parallel and Distributed Algorithms

University of Massachusetts Amherst
ECE 242 – Data Structures and Algorithms
Lecture 33

Moore’s law and processors

Figure from C. Moore
Limits on processor performance

- Processor performance increases with
  - Higher clock rate (e.g., 3 GHz vs. 2 Ghz)
  - Instructions with more functionality
  - Wider data path (e.g., 64-bit vs. 32-bit)
  - More memory (less disk access)
  - Etc.
- Physical limits on processor speed
  - Implementation complexity goes up non-proportionally
- Other approaches to higher performance?

Parallel processing

- Multiple processor cores on single chip
  - Each processor runs slightly slower
  - Total processing power higher than single core
- Challenge: how to make use of multiple cores?
  - Operating system allows multiple programs in parallel
  - Operating system does not automatically run single program on multiple cores!
- Need “parallel” versions of programs/algorithms
  - Variants: concurrent/parallel/distributed/etc.
Parallel processing examples

• Large scale simulations (e.g., protein folding, weather forecast)
  – Computations for each element done on core
  – Interactions between elements communicated between cores
• Google’s PageRank
  – Distributed computation of URL importance
• Entertainment
  – Virtual environment computations
• Etc.

Parallel processing example

• Our example for class:
  – Count primes in array of random numbers
• How can we parallelize this program?
Parallel processing example

• Our example for class:
  – Count primes in array of random numbers
• How can we parallelize this program?
  – Count primes in different ranges of array in parallel
  – Sum counts from each range
  – Report total

Concurrency problem

• Main program: int count=0;
• Two concurrent threads
  – Each thread: count = count + numberOfPrimes;
• Example
  – Thread 1: count = count + 5;
  – Thread 2: count = count + 2;
• What could go wrong?
Concurrent problem

- Concurrent process may lead to incorrect result because steps can be interleaved
  - Thread 1: reads count (0)
  - Thread 2: reads count (0)
  - Thread 1: stores 0+5=5 in count
  - Thread 2: stores 0+2=2 in count
  - Final value of count: 2
- Solution: “lock” certain regions of code so that only one thread can access them at a time

Parallelism in Java

- “Thread” are code that can be executed in parallel
  - Multiple threads can exist within one program
  - Threads share memory
- Multithreading support in Java
  - Thread class (or Runnable interface)
  - run() method for thread execution
  - start(), isAlive(), join() methods to control threads
  - Synchronized methods
Example code

- Normal implementation:

```java
for (int i=0; i<size; i++) { // count primes
    if (isPrime(array[i])) {
        count++;
    }
}
System.out.println("found "+count+" prime numbers");
```

- Multithreaded implementation:

```java
Thread t1 = new Thread(new CountPrimesInRange(0, size/2));
// create first thread with its range
Thread t2 = new Thread(new CountPrimesInRange(size/2, size));
// create second thread with its range
t1.start(); // start first thread
t2.start(); // start second thread
...
```
Example code

- Multithreaded implementation:

```java
public void run() { // this is executed when thread starts working
    try {
        int count=0; // local counter
        for (int i=start; i<stop; i++) { // count primes in range
            if (isPrime(array[i])) {
                count++;
            }
        }
        addToCounter(count); // report count to main class
    }
    ...}
```

Result

- Program output:

  normal execution:
  found 78797 prime numbers in 1000000 random numbers range 0..1000000
  Elapsed time: 640.905 milliseconds.
  multi-threaded execution:
  main: waiting for threads to finish
  Thread-1: starting at 0
  Thread-2: starting at 500000
  Thread-1: finishing at 500000
  Thread-1: added 39226 to counter
  main: t1 done
  Thread-2: finishing at 1000000
  Thread-2: added 39571 to counter
  main: t2 done
  main: both t1 and t2 done
  found 78797 prime numbers in 1000000 random numbers range 0..1000000
  Elapsed time: 338.263 milliseconds.

- Speedup of ~2x with two threads
Many more interesting problems

• Parallel computers differ in architecture
  – Memory sharing
  – Interconnect type
  – Synchronization between cores
  – Etc.
• Algorithms for parallel and distributed apps
  – Types of interactions between parallel instances
  – Etc.
• Parallelism is becoming important even for small, embedded systems
  – Multicore systems are everywhere...

Next Steps

• Lecture on Friday (Final Exam Review)
• Project 5 due Thursday