Lecture 1

January 12, 2009

Required background: computer architecture, programming languages, data structures, discrete math

Dictionary definitions:

Concurrent - 1. operating or occurring at the same time 2a. running parallel 3. acting in conjunction 4: exercised over the same matter or area by two different authorities

Concurrency - simultaneous occurence

Working definitions:

Sequential program - sequence of actions (statements) that produce a result (final value or sequence of outputs). Variously called a *process* or *task* or *thread (of control)*

Concurrent program - two or more sequential programs, cooperating to accomplish some goal, and running (at least conceptually) at the same time.

Concurrent programming - the art and science of creating concurrent programs

Hardware terms: *uniprocessor, shared-memory multiprocessor, multicomputer* (separate memories)

Parallel program - a program intended to exploit the capabilities of a parallel computer to achieve increased performance

What makes concurrent programming worthwhile? What situations give rise to concurrent programs?

- true parallelism for performance
- operating systems - device drivers, time sharing
- embedded systems - concurrency *plus* real-time
- distributed systems
- client-server systems
- user interface programs

- re-use existing applications (e.g. Unix pipes)
- true parallelism

We are going to ignore true parallelism as a motivation during much of this class. The techniques we study will be applicable to true parallelism but finding and exploiting parallelism in problems is different from what we will be doing. At the end we will perhaps revisit parallel programming.

What makes CP challenging? Why do we study it as an independent topic?

- exponential state growth - the blessing and the curse of concurrent programming. On the one hand CP provides compact and natural representation of real-world world problems with complicated state, yet on the other we have to reason about that complicated state to understand our concurrent programs
- interference
- common patterns and paradigms for dealing with the ensuing complexity
  - concurrency mechanisms
  - programs
  - reasoning

State Explosion

Consider the program

```
x = 1; y = 2; z = 3
```

How many states does it have?

– discuss

Now consider (// means concurrently with, which implies arbitrary interleaving of steps of the components)

```
x = 1; y = 2; z = 3 // a = 1; b = 2; c = 3
```

How many states does it have?

– discuss

If there are n states in each of m concurrent processes, the program has n^m states.

Now consider

```
x = x+1
```

How many states?

– discuss

To answer this we have to know something about how such a statement would be compiled, e.g. in a simple accumulator machine it might be

```
LD x,A
ADD 1
ST x,A
```

Now consider

```
x = x+1 // y = x+1
```

How many states? If x is initially 0 what is the final value of y?

– discuss

As you can see, writing correct concurrent programs is going to be very hard if understanding what a tiny given program does is this difficult! So why isn't the whole notion of a concurrent programming a folly?

– discuss

The world is complicated – many states must be represented; if these states evolve along more-or-less independent trajectories, a representation as concurrent threads is more compact, and *easier* to understand and reason about than the alternative. (Which is not to say that the reasoning is *easy!*)

What we ask then of our systems for reasoning about concurrent programs, our programming notations and mechanisms for concurrent programming, and our patterns and paradigms for concurrent programs is that they help us manage the complexity that arises due to arbitrary interleaving and interference.

Plan for the semester: we will begin by working with *shared-state concurrency* which will seem the most familiar to you but in which some of the thorniest problems arise. We will primarily use Java as the basis for this discussion but also discuss *pthreads* in C. We then move on to *message passing concurrency* in a *function language* where the our goal is to see how these characteristics ensure that the complications of concurrency don't impact parts of programs where they are really unnecessary.

Reading and problems for next time:

- Read Chapters 1 and 7 in the Erlang book and Chapter 1 in the Java book.