Parallelism

Week 9

What is parallelism?

- Executing multiple things at the same time
- In software:
  - Multiple processes
  - Multiple threads
- In hardware:
  - Instruction level parallelism

Multiple Processes

- Run two or more independent program instances at the same time
- Can be the same program or multiple programs
- Examples:

```
find ~ | grep myFile
grep pattern1 myFile | grep partern2
```

Multithreading

- Threads are software-level parallelism running inside a single process.
- Since it's one process, threads share memory (but not their stack and local variables)
- Each thread has an associated priority level
- Threads should "play-nice" by using `yield` and `sleep` functions.

Multithreading Example

Download at http://www.cs.ucla.edu/~cleak/cs35l

(demo9.tar.gz)

What can be achieved from multithreading?

- Better throughput
  - Example: Multiple threads serving up webpages
- Lower latencies
  - Example: A dedicate thread polling input devices
- Better designs (sometimes)
  - Example: A chat server with one thread dedicated to each client.

What about without multicore and hyperthreading?

- Designs can still be improved with threads
- Systems can improve performance by switching threads when one is stalled
- Logically, it still makes sense to partition many designs up into threads.

How much of a speed up can we get?

- If you go from 1 thread on an 8 core machine to 8 threads on an 8 core machine, do we get 8x the throughput? Do we get 1/8 the latency?

Amdahl's Law

- Given *P*, the parallelizable fraction of a program and *S* the number of parallel instances of that portion, we can achieve a speed up of:

$$\frac{\mathbf{1}}{(\mathbf{1}-\boldsymbol{P})+\frac{\boldsymbol{P}}{\boldsymbol{S}}}$$

Refactoring (useful to read about)

- List of refactorings can be found at:
  http://www.refactoring.com/catalog/index.html

The Lab

- Some reminder about `sed`
- On Wednesday we'll cover refactoring