Principles of Software Construction: Objects, Design and Concurrency

The Perils of Concurrency

Can't live with it.
Can't live without it.

Jonathan Aldrich

Charlie Garrod
• Homework 6a design presentations tomorrow
  - See the Piazza note or Sign Up Genius for your presentation room
Last time: Stream I/O and Networking in Java

- Basic I/O Streams in Java
- Distributed systems
- Networking in Java
  - Communication via network sockets
  - Java RMI

1: bind Foo -> Bar
2: OK!
3: locate Foo
4: It's over there.
5: Bar.bar(x)
6: baz
Today: Concurrency, part 1

- The concurrency backstory
  - Motivation, goals, problems, ...

- Basic concurrency in Java
  - Synchronization

- Coming soon (but not today):
  - Higher-level abstractions for concurrency
    - Data structures
    - Computational frameworks
Processor speeds over time
Power requirements of a CPU

• Approx.: \textbf{Capacitance} * \textbf{Voltage}^2 * \textbf{Frequency}

• To increase performance:
  - More transistors, thinner wires: more \textbf{C}
  - More power leakage: increase \textbf{V}
  - Increase clock frequency \textbf{F}

• Problem: Power requirements are super-linear to performance
  - Heat output is proportional to power input
One option: fix the symptom

- Dissipate the heat
One option: fix the symptom

- Better: Dissipate the heat with liquid nitrogen
  - Overclocking by Tom's Hardware's 5 GHz project

http://www.tomshardware.com/reviews/5-ghz-project,731-8.html
Another option: fix the underlying problem

- Reduce heat by limiting power input
  - Adding processors increases power requirements linearly with performance
  - Reduce power requirement by reducing the frequency and voltage
  - Problem: requires concurrent processing
Aside: Three sources of disruptive innovation

- **Growth crosses some threshold**
  - e.g., Concurrency: ability to add transistors exceeded ability to dissipate heat

- **Colliding growth curves**
  - Rapid design change forced by jump from one curve onto another

- **Network effects**
  - Amplification of small triggers leads to rapid change
Aside: The threshold for distributed computing

- Too big for a single computer?
  - Forces use of distributed architecture
  - Shifts responsibility for reliability from hardware to software
    - Allows you to buy cheap flaky machines instead of expensive somewhat-flaky machines
      - Revolutionizes data center design
Aside: Network effects

- Metcalfe's rule: network value grows quadratically in the number of nodes
  - a.k.a. Why my mom has a Facebook account
  - $n(n-1)/2$ potential connections for $n$ nodes

- Creates a strong imperative to merge networks
  - Communication standards, USB, media formats, ...
Concurrency

• Simply: doing more than one thing at a time
  • In software: more than one point of control
    • Threads, processes

• Resources simultaneously accessed by more than one thread or process
Concurrency then and now

- In the past multi-threading was just a convenient abstraction
  - GUI design: event threads
  - Server design: isolate each client's work
  - Workflow design: producers and consumers

- Now: must use concurrency for scalability and performance
Problems of concurrency

• Realizing the potential
  ▪ Keeping all threads busy doing useful work

• Delivering the right language abstractions
  ▪ How do programmers think about concurrency?
  ▪ Aside: parallelism vs. concurrency

• Non-determinism
  ▪ Repeating the same input can yield different results
Realizing the potential

- **Possible metrics of success**
  - **Breadth**: extent of simultaneous activity
    - width of the shape
  - **Depth (or span)**: length of longest computation
    - height of the shape
  - **Work**: total effort required
    - area of the shape

- **Typical goals in parallel algorithm design?**
Realizing the potential

- **Possible metrics of success**
  - **Breadth**: extent of simultaneous activity
    - width of the shape
  - **Depth (or span)**: length of longest computation
    - height of the shape
  - **Work**: total effort required
    - area of the shape

- **Typical goals in parallel algorithm design?**
  - First minimize depth (total time we wait), then minimize work
Amdahl’s law: How good can the depth get?

- Ideal parallelism with $N$ processors:
  - Speedup = $N$

- In reality, some work is always inherently sequential
  - Let $F$ be the portion of the total task time that is inherently sequential
  - Speedup = $\frac{1}{F + (1 - F)/N}$

  - Suppose $F = 10\%$. What is the max speedup? (you choose $N$)
Amdahl’s law: How good can the depth get?

• **Ideal parallelism with \( N \) processors:**
  - Speedup = \( N \)

• **In reality, some work is always inherently sequential**
  - Let \( F \) be the portion of the total task time that is inherently sequential
  - Speedup = \( \frac{1}{F + (1-F)/N} \)

**Suppose \( F = 10\% \). What is the max speedup? (you choose \( N \))**
- As \( N \) approaches \( \infty \), \( 1/(0.1 + 0.9/N) \) approaches 10.
Using Amdahl’s law as a design guide

• For a given algorithm, suppose
  ▪ \( N \) processors
  ▪ Problem size \( M \)
  ▪ Sequential portion \( F \)

• An obvious question:
  ▪ What happens to speedup as \( N \) scales?

• Another important question:
  ▪ What happens to \( F \) as problem size \( M \) scales?

"For the past 30 years, computer performance has been driven by Moore’s Law; from now on, it will be driven by Amdahl’s Law."
— Doron Rajwan, Intel Corp
Abstractions of concurrency

- **Processes**
  - Execution environment is isolated
    - Processor, in-memory state, files, ...
  - Inter-process communication typically slow, via message passing
    - Sockets, pipes, ...

- **Threads**
  - Execution environment is shared
  - Inter-thread communication typically past, via shared state
Aside: Abstractions of concurrency

- What you see:
  - State is all shared

- A (slightly) more accurate view of the hardware:
  - Separate state stored in registers and caches
  - Shared state stored in caches and memory
Basic concurrency in Java

- **The java.lang.Runnable interface**
  
  ```java
  void run();
  ```

- **The java.lang.Thread class**
  
  ```java
  Thread(Runnable r);
  void start();
  static void sleep(long millis);
  void join();
  boolean isAlive();
  static Thread currentThread();
  ```

- **See IncrementTest.java**
Thursday:

• More concurrency