Lecture 20: Parallelism & Concurrency

CS 62
Spring 2015
Kim Bruce & America Chambers

Some slides based on those from Dan Grossman, U. of Washington

---

Splay Tree

- Idea behind splay tree.
  - Every time find, get, add: or remove an element x, move it to the root by a series of rotations.
  - Other elements rotate out of way while maintaining order.
- Splay means to spread outwards

How to Splay in Words

- if x is root, done.
- if x is left (or right) child of root,
  - rotate it to the root
- if x is left child of p, which is left child of g,
  - do right rotation about g and then about p to get x to grandparent position. Continue splaying until at root.
- if x is right child of p, which is left child of g,
  - rotate left about p and then right about g. Continue splaying until at root.

Results in moving node to root!

Splay Tree

- Modify tree operations:
  - When do add, contains, or get, splay the elt.
  - When remove an elt, splay its parent.
- Average depth of nodes on path to root cut in half on average!
- If repeatedly look for same elements, then rise to top -- and found faster!
- Splay code is ugly -- but follows ideas given

Example of modified operation

```
public boolean contains(E val)  {
  if (root.isEmpty()) return false;

  BinaryTree<E> possibleLocation = locate(root,val);
  if (val.equals(possibleLocation.value())) {
    root = possibleLocation;
    splay(root);
    return true;
  } else {
    return false;
}
```

Parallelism & Concurrency

Object-Oriented Design

What are objects?

- Objects have
  - State/Properties — represented by instance variables
  - Behavior — represented by methods
    - accessor and mutator methods

Calculator

- Calculator class: User interface
  - including buttons and display
  - No real methods — construct & associate listeners
- State class: Current state of computation
  - Methods invoked by listeners
  - Communicate results to user interface
- Listener classes: Communicate from interface to state

Model-View-Controller

State

- Instance variables:
  - partialNumber, numberInProgress?, numStack, calcDisplay
- Methods:
  - addDigit(int Value)
  - doOp(char op)
  - enter, clear, pop

Model-View-Controller

- Dissociate user interface with the “model”
  - “model” represents actual computation
  - May have multiple alternate user interfaces
    - Mobile vs laptop versions of UI
- Model should be unaffected by change in UI.
- In Java UI generally served by “event thread”
  - If tie up event-thread with computation then user-interface stops being responsive.

Designing Programs

- Identify the objects to be modeled
  - E.g., Frogger game, Shell game
- List properties and behaviors of each object
  - Model properties with instance variables
  - Model behavior with methods (*write spec*)
- Refine by filling in the details
  - Hold off committing to details of representation as long as possible.

Implementation

- Write in small pieces. Test thoroughly before moving on.
- Solve simpler problem first — use "stubs" if necessary.
- Refactor as code becomes more complex.

Reading on Object-Oriented Design

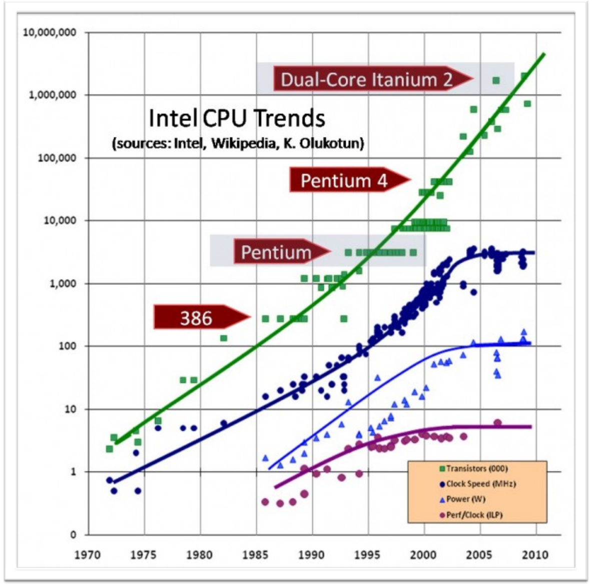
- **Practical Object-Oriented Design in Ruby: An Agile Primer** by Sandi Metz, 2013
- **Design Patterns: Elements of Reusable Object-Oriented Software** by "Gang of Four", 1994

Parallelism & Concurrency

- Single-processor computers ~~going~~ gone away.
- Want to use separate processors to speed up computing by using them in parallel.
- Also have programs on single processor running in multiple threads. Want to control them so that program is responsive to user: Concurrency
- Often need concurrent access to data structures (e.g., event queue). Need to ensure don't interfere w/each other.

History

- Writing correct and efficient multithread code is more difficult than for single-threaded (sequential).
- From roughly 1980-2005, desktop computers got exponentially faster at running sequential programs
  - About twice as fast every 18 months to 2 years



More History

- Nobody knows how to continue this
- Increasing clock rate generates too much heat
- Relative cost of memory access is too high
- Can keep making “wires exponentially smaller” (Moore’s “Law”), so put multiple processors on the same chip (“multicore”)
- Now double number of cores every 2 years!

What can you do with multiple cores?

- Run multiple totally different programs at the same time
  - Already do that? Yes, but with time-slicing
- Do multiple things at once in one program
  - Our focus – more difficult
  - Requires rethinking everything from asymptotic complexity to how to implement data-structure operations

Parallelism vs. Concurrency

- Parallelism:
  - Use more resources for a faster answer
- Concurrency
  - Correctly and efficiently allow simultaneous access
- Connection:
  - Many programmers use threads for both
  - If parallel computations need access to shared resources, then something needs to manage the concurrency

Analogy

- Typical CS1 idea:
  - Writing a program is like writing a recipe for one cook who does one thing at a time!
- Parallelism:
  - Hire helpers, hand out potatoes and knives
  - But not too many chefs or you spend all your time coordinating (*or you'll get hurt!*)
- Concurrency:
  - Lots of cooks making different things, but only 4 stove burners
  - Want to allow simultaneous access to all 4 burners, but not cause spills or incorrect burner settings

Models Change

- Model: Shared memory w/explicit threads
- Program on single processor:
  - One call stack (w/ each stack frame holding local variables)
  - One program counter (current statement executing)
  - Static fields
  - Objects (created by new) in the heap (nothing to do with heap data structure)

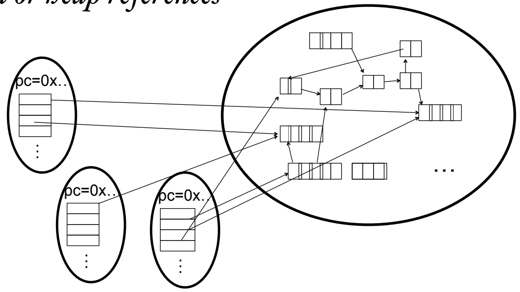
Multiple Theads/Processors

- New story:
  - A set of threads, each with its own call stack & program counter
  - No access to another thread's local variables
  - Threads can (implicitly) share static fields / objects
  - To communicate, write somewhere another thread reads

Shared Memory

Threads, each with own unshared call stack and current statement (pc for "program counter") local variables are numbers/null or heap references

Heap for all objects and static fields



Other Models

- Message-passing:
  - Each thread has its own collection of objects. Communication is via explicit messages; language has primitives for sending and receiving them.
  - Cooks working in separate kitchens, with telephones
- Dataflow:
  - Programmers write programs in terms of a DAG and a node executes after all of its predecessors in the graph
  - Cooks wait to be handed results of previous steps
- Data parallelism:
  - Have primitives for things like “apply function to every element of an array in parallel”