Lecture Topics

- an introduction to the challenges
  - atomicity
  - precedence/dependence [stopped here last time]
  - the inheritance anomaly
  - optimizing for processor/system architecture
  - determinism
- [a sequential task specification]

Administrivia

- midterm return & review
  - mean 65
  - median 64
  - stdev 13
  - max 87.5

- Have a nice break!

- previous issues
  - challenges for expressing parallelism correctly and efficiently
  - I see as the main difficulties

- remaining issues are more practical
  - expressing parallelism for robust performance
  - avoiding debugging nightmares

- algorithm vs. system/processor architecture
  - some basic tenets of parallelism
    - communication time is wasted time (not needed on a sequential machine)
    - sometimes I can trade extra work for less communication frequency and/or volume
  - also note
    - resources per processor going down with future generations
    - less memory, less memory bandwidth, etc.
    - remember algorithmic space/time tradeoffs?
  - What's the right mix of algorithm and parameters for today's machine?
  - What about tomorrow's?
  - The other N chips? (Rigel, Intel x86, Intel Larrabee, AMD x86, AMD GPU, IBM P7, IBM Cell, NVIDIA, TI, Stretch, Xilinx, Altera)
  - And their versions in 2015?
  - What about the multi-chip platforms? Heterogeneous chips? SoCs?
  - Exactly how many times are you going to write this code?
  - (some efforts now trying to address this problem)
  - don't forget…
    - you need to plan for extensibility
    - (hard enough to predict & incorporate on a sequential system)

- an old business model (now vanished?)
  - buy ~1,000 PCs combining
    - various mother boards
    - various processors
    - various disk drives
    - various graphics cards
    - various other devices
  - sell time on your machines
    - company produces software
    - you run software on all PCs
    - give results: your sequential code fails on the following combinations…

- a new business model (to fund or to found…) [one word change from above]
  - buy ~1,000 PCs combining
    - various mother boards
    - various processors
    - various disk drives
    - various graphics cards
    - various other devices
  - sell time on your machines
    - company produces software
    - you run software on all PCs
    - give results: your parallel code fails on the following combinations…

- determinism/reproducibility … is it too much to ask?

- determinism/reproducibility
  - relatively simple problem: human-controlled debugger
    - human sets a breakpoint
      - may want to inspect state on other processors
      - understand potential precedence issues (races)
    - code on one processor hits breakpoint
      - Do others stop?
      - When, exactly?
      - Should in-flight memory ops be squashed?
      - Should you squash in-flight instructions?
      - (You're hitting relativistic limits here. Simultaneity has **no meaning** at this level.)
  - What about races that aren't being tracked?
    - as the program runs
    - might go differently on a customer's platform
    - motivation for Marie Conte's Ph.D. thesis (Wen-mei's student; some informal co-advising from me)
      - HP developed a good dynamic optimizer for Java
      - some company wrote a Java server: 200k lines
      - optimizer changed a race condition, so code broke
      - company told customers: don't buy HP; buy Sun!
  - can provide (probably correct!) illusion of constrained order
    - separate stores/ownership of data
    - runtime support to avoid data races
    - hardware to track and re-create race directions
    - language to express race-free mini-programs
    - or even language incapable of expressing races (e.g., parallelism obeys strict sequential language ordering)
  - What's the cost in performance?
  - Do we care?

- related problem: intermediate state
  - if we do decide to constrain allowed orderings
    - still want to push performance
    - probably need to make constraint an illusion
      - as with out-of-order execution
      - as with sequential consistency
      - as with sequential optimizer? Oh, no.
  - when user "probes" system
    - collapses into state corresponding to allowed ordering
    - dangerous path
      - debug mode: guarantee ability to probe
      - normal mode: may not always be able to probe
    - see any analogy with general software testing? [ask]
      - nice if code also works in normal mode, no?
      - hard to debug, though

- summary of challenges
  - atomicity
  - precedence/dependence
  - inheritance anomaly
  - algorithm vs. system
  - determinism: to be or to more or less just postulate one's existence