Lecture Topics

- parallelism!
  - why bother with it? (including some review)
  - is it hard to find?
  - is it hard to implement?

Administrivia

- HW#2 returned

Parallelism: Why Bother with It?

- few people ever do (fraction of software writers is small); why not?
  - gain is essentially constant, not scaled with problem
  - want a fixed gain?
    - wait a few years
    - or build your application in hardware instead
  - few people are motivated by "no gain, just pain"
- so why has anyone ever done it?
  - sometimes the pain is vanishingly small
    - little or no actual interaction between parts of the application
    - by design in database systems (my hypothesis from grad. school: for enough $, any application can be made embarrasingly parallel)
  - more resources
    - memory
      - some commercial CFD codes in late 90s ran as separate processes primarily due to limits on virtual addr. space
      - large problems often don't fit in the memory available with a single-chip machine
    - memory bandwidth, I/O bandwidth
      - many applications limited by engineering decisions based on generic workloads
      - e.g., for database sorting benchmarks, the NOW cluster (100 machines) beat a commercial system designed for the server market because the NOW cluster had more disks
  - sometimes you need the fixed gain
    - supercomputing/computational science
    - grand challenge applications
    - h/w also works, but it's more expensive and restrictive (e.g., limits utility of algorithmic advances)

- recall first lecture…What has changed about exploiting parallelism?

- want a fixed gain?
  - wait a few years…sorry, that trend has ended
  - or build your application in hardware…only $50-100M for a typical modern chip process!
- sometimes the pain is vanishingly small
  - maybe only use free/easy parallelism?
  - 2-processor system cost was the sweet spot briefly in mid-90s
  - before Intel/AMD entered server market
  - small SMP's are again popular as servers, but multi-core is now also ubiquitous on desks/laps
- more resources? no, not with one machine
  - in fact, fewer resources per processor
  - we'll need to be careful not to overtax resources
    - resource use/response time as a function of load
    - generally has a sharp rise as utilization nears 100%
  - methods for simplifying parallelism tend to exacerbate by temporally correlating resource use

- new viewpoint on parallelism
  - parallelism may be attractive to more people
    - zero- or tiny-pain variants
    - small fixed gains
  - particularly if "fixed" gain
    - can be made to scale with time
    - that is, process generations/density/number of cores
- How far can we push the envelope with good engineering?
  - research topic, but one that needed to be solved 10 years ago
  - potential for immediate impact

Is Parallelism Hard to Find?

- my view: almost never
  - nearly always trivial from a task specification viewpoint
  - took me substantial effort
    - to find a problem that is fundamentally serial
    - or at least requires you to solve a hard math problem
  - as a result, we get industry speakers telling us how easy it is to use parallelism (e.g., IBM Cell compiler author, IBM researcher, and others)
- note: easy to see does not imply easy to use
- some practical concerns
  - may be hard to see parallelism by looking at code
  - may be hard to derive task specification from code
  - task specification may not make sense for code
    - e.g., library used for many purposes
    - what if best parallelism source for a task is not within library?
  - may be hard to rewrite sequential code cleanly to expose parallelism
- all of these issues refer to existing code
- one view
  - leveraging parallelism in existing code is hard
  - writing parallel code is not hard
  - it's just a matter of education
- fairly common view
  - both among people who have and haven't written parallel code
  - ressurrected roughly every five years for the last 20
  - resulting in introduction of undergraduate parallelism classes
  - draw your own conclusions
  - but keep the new landscape in mind w.r.t. current generation

Is Parallelism Hard to Implement?

- my view: generally, yes
  - the joke version (it's funny because there's a great deal of truth in it): Only graduate students write parallel programs.
  - Pfister's version, in case the undergraduates don't fully get it: "parallelism is the wave of the future, and graduate students are inexpensive, intelligent, and motivated." (p. 221)
- is there a fundamental reason for the difficulty? maybe…
  - note
    - the following observation was made based on a talk by Matt Frank
    - credit to him, blame for being overly trite to me
  - parallelism (even "data parallelism") is a control abstraction:
    - two (or more) operations can be done at the same time
    - in a way in which they don't interfere with each other
  - everything we've talked about so far in this class has been data abstraction!
  - That difference should worry you.
  - (Yes, people have tried to create data-centric expression of parallelism: dataflow, pipeline parallelism, stream processing; it works sometimes.)