Lecture Topics

• an introduction to the challenges
  – atomicity
  – precedence/dependence [got to here]
  – the inheritance anomaly
  – optimizing for processor/system architecture
  – determinism

Administrivia

• …
An Introduction to the Challenges

• define a framework to help us focus…
  – start with some task and find one or more sources of parallelism
  – each source of parallelism is like a bunch of mini-programs
  – mini-program scale can vary
    • could be as big as a whole program
      (e.g., run RigelSim with different cache sizes to measure the impact of cache size on performance)
    • could be as small as an instruction
      (e.g., add an array of integers)

• now let’s think about using these mini-programs
atomicity

  atomicity is ALWAYS with respect to something
  To a high-energy photon, atoms are not.
  that said
  
  do a pair of mini-programs need to execute atomically with respect to one another?
  that is, does the end result have to appear as though the programs’ execution did not interleave at all?
  or, are there pieces of the mini-programs that must not appear to interleave?
  note the “appear” part; to whom or to what?

  for example
  
  mini-programs that update your bank account
  each pair of updates (read, change, write back) needs to be atomic with respect to one another
  not a major issue for mini-programs with no shared data
  Do two mini-programs share data?
  sometimes easy to know, but sometimes not
    Do two insertions into a hash table share data?
    Are two graph node updates based on all of the nodes’ neighbors atomic?
    Can I make them atomic with a bipartite graph?

  common failure types: algorithmic errors
  programmer thinks that operations are independent
  simply hasn’t considered input data for which they are not
  or some other programmer may reuse code but not understand assumptions that imply independence
• precedence/dependence: what is it?
  – atomicity does not constrain relative order
  – some mini-programs may have to finish before others start (data dependence)
  – results from one mini-program may eliminate the need to execute another mini-program (control dependence)

• common issues with precedence
  – programmer fails to express constraint
    • simple example: bipartite graph + discretized diff. eq’s.
    • do updates need to alternate strictly?
    • or can I let them run ahead so long as they’re atomic?
    • depends...
      – static problem (heat)? Yes! Run, run, run…
      – dynamic problem (em3d)? No way: physics matters!
  – speculation can be dangerous
    • TimeWarp discrete event simulator
    • try to execute state machine even if inputs may be en route
    • state machine emulation in software may cause crashes
    • is isolation + rollback ready for bizarre exceptions?
  – language makes expression difficult
    • What is the element of computation?
    • How do I name it, when many may be created dynamically?
    • If I can’t name it, how can I specify ordering constraints?
    • consider parallel heap insertions
      – a second insertion may be in a different part of the tree
      – but has to defer to the first insertion at the root node
      – of course, not all insertions reach the root…
• common issues with precedence (cont’d)
  – semantics of operations may be confusing, misleading programmer
    • e.g., barrier synchronization crossed when all processors reach it
    • What exactly is guaranteed to have finished?
      – Are memory operations before a barrier (e.g., stores)
        guaranteed to have completed? For all processors?
      – Are network messages sent before a barrier guaranteed to
        have arrived?
      – Are I/O operations performed before a barrier guaranteed to
        have been sync’d to disk?
      – answer: sometimes, on some systems…
    • my first parallel bug: network messages aren’t guaranteed with a
      separate network for barriers (e.g., on CM-5)
  • synchronization and the inheritance anomaly
    – S. Matsuoka and A. Yonezawa, OOPSLA/ECOOP’90 Workshop on
      Reflections and Metalevel Architectures in Object-Oriented Languages.
    – 1993 version (book chapter) usually cited instead
    – synchronization (atomicity, precedence, and dependence)
      at odds with inheritance
      • hard to encapsulate synchronization such that
        – derived class author need know nothing about it
        – derived class author need not rewrite base class
      • simple example
        – bounded buffer with atomic get/put
        – extend to extract first two elements atomically
        – extracting two MUST synchronize with get/put
    • hundreds of papers trying to solve
    • yet solution missing/broken in most common parallel interfaces
      (e.g., POSIX, Java, etc.)