

Lecture Topics

- challenges for exploiting parallelism
 - precedence/dependence (wrap-up)
 - the inheritance anomaly
 - optimizing for processor/system architecture

Administrivia

- Send me lab 2 by e-mail (several have already).
- Have a nice break!

- common issues with precedence (cont'd)
 - 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...
 - 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. ACM, Aug. 1990.
 - 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.)

- 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)