CMPSCI 691W Parallel and Concurrent Programming **Spring 2006**

Lecture 10: March 13

Lecturer: Emery Berger *Scribe: John Burgess*

10.1 Overview

This lecture describes common parallel language constructs and features as well as the explicitly parallel language Clik.

10.2 Parallel Language Taxonomy

- Fully abstract parallel languages perform all parallelization in the compiler (Haskell, Unity), where programmer is oblivious of parallelism. A compiler capable of converting single threaded C code to maximal parallel threads of execution would be the holy grail.
- Explicitly parallel languages (Multilisp, Fortran+, NESL) use constructs like futures to initiate parallelization or special comments for compiler to interpret. NESL can operate on distributed machines or on multiple CPU platforms.
  - Explicit decomposition (CODE, Flux) separates independent tasks from one another.
  - Explicit Mapping (Linda) assigns/retrieves data in an abstract pool of available data repositories that may take arbitrary amounts of time to respond to requests (this abstract pool of tuples similar to JINI's abstract pools of tuples).
  - Explicit communication is simply static dataflow between worker threads.
  - Synchronization (MPI, fork(), Java, POSIX threads, Ada, occam) systems for explicit communication provide friendlier methods of sharing data like message passing, shared memory, rendezvous, etc.

10.3 Cilk

- Explicit everything, but threads are abstract and later assigned to machine threads via the compiler
- Shared memory thread coordination
- Provably efficient work stealing scheduler (guaranteed efficiency is rare in parallel languages)
- C language with some additions to notify compiler of parallelism (spawn and sync commands)
- Clean programming model with C language programmer familiarity and portability
- Divide and conquer algorithms are perfect for Cilk (like Fibonacci, but with more work per recursive call)

- Constrained to nested parallelism where children are spawned and then synched. No goto like activity to move between functions while spawned children remain running
- Supports inlets, which flag code segments as atomic with respect to one another
- Children may be aborted prior to execution, which kills spawned threads which have not run yet (good for applications like search that should stop upon finding something). Running threads must be signaled manually for termination, who must, in turn, abort their running children, else their children must be manually terminated.
- Unfortunately, requires locking to manage access to shared data. Deadlocking is still a problem in Cilk
- Compile with Cilk to output giant mess of C code, then compile with gcc, then link with Cilk runtime to produce an executable
- On an architecture with an infinite number of available CPUs, Cilk will execute a program in the time it takes for the program's critical path to execute.
- Cilk can determine a program's critical path length
- Parallel Slackness: $P_{average}/P >> c$ such that parallel slackness is the reduction in execution time achieved when maximally parallel execution occurs versus single threaded execution.
- Work-First Principle: Since the majority of execution speedup occurs via parallelism, maximizing parallelism can be achieved by elongating the critical path when necessary. Why? Because the critical path is only a constant c, wile all parallel work are part of the asymptotic execution time, which must be minimized.
- Work-Stealing: Workers normally take work from the bottom of their own deck. If out of work, they take work from the top of other threads queues such that the stolen work is the oldest work that can be stolen and will spawn the most work. This is asymptotically optimal.
- Fast clone, employed by workers taking work from their own queue, does not lock a work deck before beginning thread setup, but later checks to see if that work was stolen before actually beginning work. If one item exists and the worker and thief modify the deck concurrently, the corrupted deck top and bottom are detected and the thief backs off. Slow clone, employed by thieves, locks a deck to grab work (this is required to move state to another machine or CPU worker deck), but thievery is not the common case and so locking is uncommon.
- Nondeterminator: detects race conditions, but is no longer supported as part of Cilk