

Lecture Topics

- more complex approaches (wrap-up)
- a few basic concepts
- some common misconceptions
- a few useful ideas
- performance reasoning for shared memory threads (Lab 3)

Administrivia

- topics
 - detailed overview of message-passing and shared memory threads
 - 12 ways to fool the masses (assigned reading?)
- possible topics
 1. replay and race detection, theory and practice
 2. message-passing optimization (short)
 3. synchronization: theory, general approaches, and some algorithms
 4. parallel primitive operations and optimization
 5. parallel data structure and library examples
 6. an “application”: parallel garbage collection
 7. other approaches to parallelism
 - streams
 - TM & guarded atomic execution
 - thread-level speculation/control independence
 - data-centric high-level models
 8. communication topology: embeddings, randomization, fairness, avoiding deadlock
 9. auto-tuning (mostly current research? There is some older stuff...)

- fork-join (e.g., Cilk)
 - mostly academic approach for expressing parallelism
 - shared memory systems, with some efforts to apply more broadly
 - dynamic number of threads
 - fork new set of threads, join to wait for completion
 - forked threads may also fork new threads (hierarchical)
 - explicit parallelism (threads usually based on functions)
 - asynchronous sharing, but sometimes cast into functional language

- task models (e.g., Rigel Task Model)
 - many small tasks bounded by synchronization
 - a common approach within other models (e.g., implementation of OpenMP loops)
 - less common as a self-contained model
 - Rigel software stack also allows SPMD thread model
 - dynamic number of threads (task = thread)
 - explicit parallelism (threads usually based on functions)
 - sharing model varies
 - Rigel Task Model is synchronous
 - but relies on programmer to follow rules

A Few Basic Concepts

- high-performance computing definitions...
- parallel speedup, or just “speedup”
 - When I run my program in parallel
 - it finishes X times faster than when I run it sequentially
 - $X = T(\text{sequential}) / T(\text{parallel})$
 - X is the speedup
 - finding T(sequential)
 - best algorithm for sequential machine
 - optimized for sequential machine
 - no parallelism support remnants
- note that speedup assumes a fixed problem size
- parallel efficiency, or just “efficiency”
 - How well am I using my parallel resources?
 - efficiency on P processors = speedup on P processors / P
- scalability
 - For how many processors is speedup linear, or is efficiency flat?
 - at some value of P, with fixed problem size
 - speedup will flatten out
 - later it will drop (unless you leave processors idle)
 - “good” scalability means
 - no falloff on your machine
 - maximum measurable value of P

- other variants of speedup
 - see J. P. Singh, J. L. Hennessy, A. Gupta, “Scaling Parallel Programs for Multiprocessors: Methodology and Examples,” *IEEE Computer*, 26(7):42-50, July 1993
 - fixed size per node (scaled speedup)
 - using parallel system to run bigger problems, so why use measure fixed size?
 - patently false for most applications at some scale
 - memory constrained speedup
 - biggest problem that fits in memory
 - looks really bad unless the problem runs in $O(N)$
 - time constrained speedup
 - biggest problem that finishes by the time I return from lunch
 - sometimes reasonable...
 - ...but we could wait overnight for a grand challenge application?

- parallel grain size (work per parallel task)
 - each possible source of parallelism has “natural” grain size
 - loop body
 - objects in a container
 - rows/columns/blocks in a matrix
 - elements in a matrix
 - graph nodes/connected components
 - some may exhibit higher variance than others
 - conditionals/inner loops in loop body
 - complex per-object methods
 - rows in upper/lower diagonal matrix
 - matrix elements usually roughly constant
 - degree of nodes, size of connected components