Lecture Topics

• simple approaches to parallelism
• dimensions for parallel models

Administrivia

• example of sequential task on web page (since before break)
• hand out Lab #1: due Th next week (Thursday 5 April)
• opportunity for interdisciplinary internships at Intel – see web board for details; contact Jeff Cook if you’re interested
The Simpler Approaches to Parallelism

- starting simple and working towards more complex models

- “embarrassingly parallel”
  - exploiting parallelism poses no challenge whatsoever
  - examples
    - different jobs
    - parameter variations
      - architectural/network simulations
      - ground state quantum chem./materials design
    - independent parameters / input data sets
      - e.g., thumbnail all of my images
      - disk can serialize this process!

- parallel under the hood
  - parallel implementations of libraries called from sequential code
  - domain-specific languages with parallel execution
    - e.g., Structured Query Language, or SQL
    - may include dynamic optimization (as with databases)
    - note that user does not “know” about parallelism

- read-only sharing
  - e.g., ray tracing sections of a screen based on a shared 3D model
  - minor variation from embarrassing parallelism
  - may optimize distribution of replicated data to enhance performance
  - starts to break when dataset does not fit in one node’s memory
    - (possibly replicated) hashed distribution coupled with caching
    - may retain simplicity of embarrassing parallelism
  - at some point the problem is more similar to complex models, though
• master-slave model (minor sharing)
  – large and possibly dynamically-generated set of inputs
  – results
    • must be integrated, tallied, filtered, etc.
    • e.g., reduced by finding the minimum value or summing
  – has some nice properties
    • security/reliability through replication
    • doesn’t finish in certain time? send elsewhere instead
    • don’t trust the result? run on independent alternative and compare
  – many systems leverage
    • SETI@home, Folding@home, etc.
    • Condor (idle cycle harvesting)
      – had special “parallel” package
      – for master-slave application support
      – (only parallel model with Condor!)
    • S. Mitra’s reliable implementation of Intel’s RMS benchmarks
      – RMS = recognition, mining, and synthesis
      – also uses iterative nature of benchmarks to squash errors

• independent by programmer assertion / language constraint
  – execution contexts may share read-only data
  – but are guaranteed to produce separate results
  – examples
    • DOALL loop parallelism (programmer assertion of indep.)
    • similar constructs in OpenMP (shared memory parallelism API)
    • side-effect-free functional languages
      (dependent code cannot be expressed in language)
we’re not done, but already some options arise
  – task: Given a graph, find shortest paths between all pairs.
    • Do you start with a good algorithm and parallelize it?
    • Or share/copy the graph and have each task find all paths from a single source?
  – Floyd-Warshall is $O(V^3)$, Johnson’s algo. is $O(V^2 \lg V + VE)$
  – Dijsktra is $O(E \lg V)$, but we have to run $V$ times…
  – we lose about a factor of $\lg V$
    • using Dijkstra $V$ times not work efficient (same asymp. total work)
    • but we get easy parallelism
  – consider…
    • Do you want to pay the complexity cost?
    • How big is the graph?
    • How often do you plan to run your code?

**Dimensions for Parallel Models**

static/dynamic number of execution contexts
  – let’s call them threads
  – and call hardware execution contexts processors
  – static/fixed pool of threads
    • present at start of program, and around until the end
    • each has a unique identifier (0 through $P – 1$)
  – dynamic creation/destruction of threads
    • sometimes at start/end of block-structured language concepts
    • sometimes under direction of existing threads
    • each thread
      – may only execute on one particular processor
      – or it may execute on many (vary over time)
• implicit vs. explicit parallelism (a range of possibilities)
  – “purely” sequential language / inherently parallel (e.g., side-effect-free functional language)
    • compiler may still find parallelism
    • particularly if language is type-safe or lacks support for pointers
  – explicit vector/SIMD/VLIW/EPIC in ISA or language
  – parallel loop annotations
  – same program, multiple data (SPMD); differentiate by unique IDs
  – distinct “main” thread functions (when done, thread terminates)
  – distinct binaries