Course Summary

Mark Greenstreet

CpSc 418 – Apr. 5, 2017

- The things we have done:
  - parallel algorithms
  - parallel architectures
  - parallel performance
  - parallel paradigms
- and the things we have left undone.
- An exam, but first, a party.

Parallel Algorithms 1

- map, reduce, and scan: simple patterns
  - Easy to parallelize.
  - Learn to recognize when a problem can be solved by these simple methods.
- sorting networks
  - oblivious computation: the control flow doesn't depend on data values.
    - oblivious algorithms are good candidates for parallelism because we can determine the control flow in advance.
    - This lets us identify the data dependencies, and find a parallel solution.
  - The 0-1 principle
  - Bitonic sorting: it's merge sort with an oblivious merge.

Parallel Algorithms 2

- Matrix operations
  - matrix multiplication dividing the matrix into b
    - Dividing the matrix into blocks
    - Analysis of the compute and communication costs.
  - BLAS and cuBLAS: use a library when you can!
- Map-Reduce
- Model checking
  - We only had two lectures on the topic and no HW.
  - Won't ask any detailed questions, but if there might be some high-level questions with one sentence answers in the review questions, in which case similar questions could be on the exam.
  - Know what model checking is: verifying properties of a hardware or software design, using a finite-state-machine model, finding the reachable states.
  - The idea of distributing work by hashing values and sending each to its owner process.

Parallel Architectures

- pipelining and instruction level parallelism
- shared memory multiprocessors
  - Know what a cache coherence protocol is.
  - Explain the idea of shared-reader or exclusive writer.
  - Be able to point out that real cache coherence protocols aren't as "consistent" as the simple (e.g. MESI) model from class.
- message passing architectures
  - Rings, tori, hypercubes
  - Latency, bisection width.
- data parallel architectures
  - SIMD (and SIMT)
  - instruction execution: why so many threads
  - GPU memory hierarchy

Parallel Performance

- λ
  - Communication costs are critical to understanding parallel performance
  - This is true across all parallel architectures.
- Overheads and losses
  - Communication, synchronization, extra memory, extra computation
  - Idle processors, resource contention, non-parallelizable code
- Speed-up and Amdahl's Law.
- **Understanding real world performance requires experiments and measurements.**

Parallel Programming Paradigms

- Message passing: Erlang
- Data Parallel: CUDA
- We've mentioned shared-memory.

. . . and the things we have left undone

- More paradigms and programming frameworks
  - shared memory: Java threads, pthreads.
  - futures: e.g. Scala
  - MPI and OpenMP (for scientific computing)
  - many big-data, machine-learning, and scientific computing frameworks
- Do a bigger project.
- **The good news:**
  - You've got what you need to lean new paradigms, new frameworks, and take on realistic projects.
  - From my experience with research projects that have moved into industry in the past few years, you've got the critical knowledge and skills.
  - Writing industrial-strength, parallel-code with good performance is still more than a homework assignment, but when my students have done it, they've built on the foundation you now have.