Midterm Review/Objectives CpSc 448B

1 Proposed learning objectives submitted to the Computer Science "Program Experience" committee

1. Familiarity with Parallel Program Design
   (a) The student can implement methods such as map-reduce, scan, block-allocation, and work queues to implement small (up to a few hundred lines of code) parallel programs.
   (b) The student can describe large-scale parallel programming application that are based on these methods.
   (c) The student is aware that
      i. parallel program design often centers around identifying the communication and synchronization patterns of the computation;
      ii. a parallel program is much more than a parallel algorithm – there are important issues such as software engineering and performance tuning that are not covered in this course.
2. Familiarity with Parallel Programming Paradigms
   (a) The student can implement simple parallel programs using message-passing and shared-memory programming models.
   (b) The student can describe how parallelism, communication, and synchronization are expressed in these models.
   (c) The student is aware that there is much ongoing work on finding better approaches to parallel programming and that new methods are likely to emerge.
3. Familiarity with Parallel Algorithms
   (a) The student can implement existing algorithms that are distinctively parallel such as bitonic sort, producer-consumer, and termination detection.
   (b) The student is aware that
      i. there is a large body of prior work on parallel algorithms;
      ii. much of this prior work is for large-scale scientific computing but these algorithms can be applied to other problems as computers with a large number of CPU cores become widely available.
4. Familiarity with Parallel Correctness
   (a) The student can
      i. write a proof for simple invariants;
      ii. identify races that occur for simple examples of incorrectly modifying existing parallel algorithms.
   (b) The student can describe races, deadlock, and livelock errors.
   (c) The student is aware that
      i. most programs only need systematic use of standard synchronization and communication primitives;
      ii. if they *really* need something new, then need to find (or become) someone who can make effective use of formal methods beyond what is covered in this course.

5. Familiarity with Parallel Architectures
   (a) The student can describe message passing and shared-memory architectures.
   (b) The student is aware that
      i. there is a wide range of parallel computing platforms from cell-phones to super-computers;
      ii. many experimental architectures are commercially available, and the long-term survivors can't be predicted today;
      iii. real shared-memory multiprocessor don't provide sequential memory consistency (nor does the Java memory model);
      iv. power consumption is the critical constraint driving parallel architectures. Other physical constraints such as reliability and communications bandwidth and latency are important as well.
6. Familiarity with Parallel Performance
   (a) The student can
      i. implement code to identify and quantify bottlenecks;
      ii. reconcile experimental measurements with analytical predictions to get a realistic performance model.
   (b) The student can describe the most common causes of performance loss: communication overhead, synchronization overhead, idle processors.
   (c) The student is aware that
      i. system software has a major impact on communication and synchronization costs;
      ii. locality is critical to achieving good performance;
      iii. hardware performance is determined mostly by power consumption, but we don't yet have practical models that make these trade-offs visible to the programmer.

2 Oct. 13 "snapshot"

At this point, we're pretty well on-schedule. We're generally working from the end-of-the objectives back to the beginning. That's because the list that I sent to the committee starts with the "big-picture" that we want to complete by the end of the term, and progressively expands on the details of what's needed to make that picture work. So far we've covered:

1a: Program design. We've covered the the "reduce" part.

- Sept. 22 lecture, slides 16 & 17.
- Lin & Snyder, Chapter 1 → Examining Parallel and Sequential Programs → A Paradigm Shift → Pair-Wise Summation (p. 12).
- Lin & Snyder, Chapter 5 Schwartz' Algorithm . . . The Reduce and Scan Abstractions → Structure for Generalized Reduce (pp. 113–121).

We'll be covering much of the rest of this in the next two weeks, but that's not material for the midterm.

1(c)i: the central role of communication has been a theme throughout the course.

2a: Paradigms. We've done message-passing computing in Erlang. We'll do a bit more in Erlang, and we'll then finish up message passing using MPI (first week of November). We've mentioned shared-memory programming in the context of shared-memory multiprocessors, for example, Dekker's algorithm.

3 Algorithms. In addition to the simple "count-3's" example, we've looked at matrix-multiplication from several angles, a parallel version of a prime sieve, and a mutual exclusion algorithm. We'll cover sorting, dynamic programming, producer-consumer, termination detection, and maybe others during the rest of the course.

4 We've mentioned issues of correctness when discussing sequential consistency (Oct. 4), and Dekker's algorithm (also Oct. 4). More coming up in the rest of the term.

5 We've covered everything on the "official" learning objectives list for architectures.

1. Sept. 29 & Oct. 11: Superscalars (e.g. the R10000)
2. Oct. 4: Shared memory machines
3. Oct. 6: Message passing multiprocessors
4. Lin & Snyder, Chapter 2 → A Look at Six Parallel Computers (pp. 31-44).

I'm planning on going beyond the official list to include GPU architectures and programming sometime in November.

6 We've covered everything on the "official" learning objectives list for performance.

1. Sept. 20: Overhead
2. Sept. 22: Quantifying Performance
3. Lin & Snyder Chapter 3 (all of it).
4. Homework 2.

Performance and being able to measure it is the critical point where theory meets real-world application. We'll be continuing to measure "how fast" our programs go and figuring out what the key issues are throughout the term.

3 What kinds of questions could be on the midterm?

That's a good question. The first two homework sets have been 100% programming problems. That was intentional to get the basic pieces in place for exploring other aspects of parallel computation. But, I realize that it could leave you guessing about what kinds of questions to expect on the midterm. Here are some (definitely representative, reasonably thorough, but I might think of something else in the same spirit as these) of the topics that I'll be drawing questions from:

Performance:

- Speed-up: what does it mean that a parallel program is 25 times faster than a sequential program?
- throughput and latency. Be able to explain what they are, how they are related, and how they are different.
- Understand the various kinds of overhead that can prevent parallel programs from realizing ideas speed-up. Be able to give examples of each kind.
- Dependences: Be able to define and give examples of a flow-dependence, an anti-dependence, and an output dependence. (Lin & Snyder, Chapter 3, Parallel Structure → Dependences ... How Dependences Limit Parallelism, (p. 68-71)).
- Amdahl's law: what is it? Be able to give and use the formula. Also, be able to express the intuition behind it. Be able to give examples of how it applies and of situations where it does not apply.

- Super-linear speed-up: What is super-linear speed-up? Give some examples of things that can cause it?
- What is an "embarrassingly parallel application"?
- From the homework: what have you learned about measuring the execution time of a program? Is it easy to get one, clear number? If not, why not?

Architecture

- Shared-memory multiprocessors
  - What is a snooping cache? The MESI protocol.
  - What's directory-based coherence?
  - What's is sequential consistency?
  - Do most chip multiprocessors provide shared memory?
  - Do most chip multiprocessors guarantee sequential consistency?
- Message passing computers
  - Describe some of the key design-issues for the communications network: latency, bandwidth, robustness.
  - What are some typical network topologies? Name one advantage and one disadvantage for each.
  - Describe some of a small- or medium-sized compute cluster.
  - Describe some of the key features of a large supercomputer.
- Superscalar computers
  - Register renaming. The life-cycle of a register.
  - How renaming handles "false dependencies" (see Lin & Snyder, Chapter 3, Parallel Structure, for a more general discussion of dependences).
  - Branch prediction: what it is. How mispredicted branches are handled.
  - Precise exceptions: what they are. How a superscalar provides precise exceptions.
  - Scaling issues: why not make a super-scalar that issues 200 instructions/cycle?

Programming

- Be able to read a short (10-20) lines of Erlang code and explain what it does (this will **not** use past winners from some "obsfucated Erlang" contest).
- Be able to write up 3-10 lines of Erlang to complete a function.
- Given a sketch for an algorithm, comment on its communication requirements and other efficiency and performance issues.
- Looking over the lectures, readings, and homework, I could include something from the examination of various ways to parallelize matrix multiplication.

DLS talk

- What was the main claim of the talk?
- What problem was he trying to solve?
- How did the speaker substantiate his claim?
- How does this fit into the past 20 years of computer systems work?
- What conclusions did the speaker make about how computer systems are likely to evolve?

4 Is this a theory course or an applied course?

Yes. My goal is to show how parallel computation changes how we reason about algorithms and software (i.e. theory), how these changes are driven by what computers are possible to build (architecture), and getting experience with how that changes how we program (software, applied). The course should prepare you for taking advantage of the opportunities that are created by wide-spread parallel computing. I don't have a crystal ball; so, I can't tell you what architectures, languages, programming paradigms, or companies will succeed or fail. This course aims to give you experience with key concepts that will be part of any application of parallel computing.