Multicore Computing – CS 5966 / 6966 – Week 1 (1/12/09)

`http://www.eng.utah.edu/~cs5966/LECTURES`
`http://www.cs.utah.edu/formal_verification`

Contents

1 INTRODUCTION

Multiprocessor machines or "multi-cores" as they are known are quickly taking over every aspect of computing. The central questions in this area are:

- Why do we even need them?
- How to proceed with their hardware design?
- How to program them?

1.1 Why do we need multi-core CPUs?

A very good answer is provided in "Amdahl's Law in the Multicore Era" by Mark D. Hill and Michael R. Marty, IEEE Computer, July 2008, available from `http://www.cs.wisc.edu/multifacet/amdahl`.

Assignment 1 (due 1/18): These questions are best answered after discussing amongst you, and/or with me. Again, as noted in the class webpage

`http://www.eng.utah.edu/~cs5966`

your submissions must, by default, be in the form of discussions posted in the class Google Group. As to the length of your answers, I will leave it to your imagination. Clearly, inadequate and/or cryptic and/or content-free answers are not what I am looking forward to. On the other hand, I also don't want to see a repeat of what's in the paper – tastefully summarize, hitting all highlights. Please precede each answer with the keyword describing the question.

Hill, Amdahl Paper: Read this paper

`http://www.eng.utah.edu/~cs5966/LECTURES/PAPERS-AND-SLIDES/ieeecomputer08_amdahl_multicore`

and summarize the main ideas in a page (single-spaced 11pt font).

Hill, Speed-up Calculator: The calculator is available at

`http://www.cs.wisc.edu/multifacet/amdahl`

What is the maximum speed-up predicted by their model for fraction parallel (F) = 0.95, BCEs (N) = 64, for the symmetric, asymmetric, and dynamic multi-core organizations.

Hill, Redo for different perf(x): Redo for another plausible $perf(x)$ function (I don't care what it is, so long as it is not super-linear; provide some justifications for your choice).

Hint: Can you justify a $ln(x)$ function?

Hill, on Algorithm Selection: How can we obtain higher F for a given problem (such as sorting)? Justify in a few sentences, suggesting an approach to increase F.

Hill, on the Existence of Parallel Algorithms: Can we obtain higher F for *all* problems? Justify in a few sentences. If it helps you, think about breadth-first search and depth-first search as two specific problems – does it seem that both are parallelizable with equal ease?

Hint: Read-up on P-Completeness from somewhere (e.g. Wikipedia)

1.2 Insights into Software

Jack Dennis provides an interesting historical view of computing in his paper Toward the computer utility

avaliable from

`http://csg.csail.mit.edu/Users/dennis/essay.htm`.

He emphasizes the role of functional programming.

Assignment 1 (due 1/18):

Dennis Historical Review: What are Dennis's main observations?

Given that parallel algorithm design is fundamentally important, the importance of expressing the problem without committing to a sequential implementation comes out in Blelloch's work, and his talk on Parallel Thinking.

Assignment 1 (due 1/18):

Blelloch Talk: This talk is available from

under "Parallel Thinking."

What are Blelloch's main observations concerning parallelization? What are his views on describing problems at a high level that makes it easier to identify parallelism?

It is noteworthy that his work on the NESL language and parallel algorithm animation occurred nearly 13 years ago (we are attempting to build NESL). He proposes a machine independent performance model which is worth comparing against that provided by Cilk.

In an article in

CACM November 2008,

available from

`http://portal.acm.org/citation.cfm?doid=1400214.1400227`

Cantrill et al. claim that the advances in parallel software design over the decades of the 70s and 80s created the need for multi-core CPUs. This is a contentious point because it is widely acknowledged that these decades predate a number of advances in computer hardware design, including the move towards RISCs, advances in compilers, and the steady progress in memory technology, VLSI design, and CAD tools. In fact, clock frequency doubling was a perfectly feasible way of obtaining higher performance. See

Software and the concurrency revolution by Sutter and Larus,

available from

`http://portal.acm.org/citation.cfm?id=1095421`

and of course the famous

"Free Lunch is Over" article by Sutter

available from

`http://www.gotw.ca/publications/concurrency-ddj.htm`.

Yet, Cantrill's paper gives many valuable insights, especially on how concurrency is likely to affect different parts of a complex software system.

Assignment 1 (due 1/18):

Cantrill Summary: Summarize the main points in Cantrill's paper. Describe what he lists under "concurrency is for performance." Describe what he lists under "illuminating the black art."

A recent whitepaper of UPCRC

available from

`http://www.upcrc.uiuc.edu`

points out that whereas previous parallel computing research had focussed on server and high performance computing (HPC) applications, the shift of of focus toward client parallel applications escalates the scope and magnitude of the questions – *i.e.*, how many people are affected by the various outstanding problems in multi-core computing.

You may also download and begin understanding many of these issues from

the free e-Book

available from

`http://www.cilk.com/ebook/download5643`

by Leiserson.

Assignment 1 (due 1/18):

Leiserson Summary: Describe the basic definitions in Leiserson's e-book pertaining to work, span, and parallelism.

2 DETAILED SYLLABUS

Week1, 1/12-14: Introduction, Hill's papers, Amdahl's law in the multi-core era, Leiserson's e-book, Cilk overview.

Week2, 1/19-21: Programming in Cilk, Exercises using Cilk, understanding span, work, parallelism.

Week3, 1/26-28: Overview of MPI, Verifying MPI Programs using ISP. Emphasis is on what progress is being made in establishing correctness of concurrent systems using automated tools.

Week4, 2/2-4: Writing state transition semantics of APIs, Illustration on MPI (paper/pencil). The ability to write state transition specifications is another fundamental aspects of concurrent programming.

Week5, 2/9-11: Overview of PThread Programming, Verifying PThread Programs using Inspect. This is to illustrate how thread programming basically works, and how tools can help detect bugs in thread programs.

This class placs heavy emphasis on projects. Consequently, you are required to select, and begin working on a project from Week5.

Projects may be selected from one of these:

- The Herlihy/Shavit book ("The Art of Multiprocessor Programming" or TAMP) There are plenty of concurrent data structures described in the book - studying and experimenting with them (verification using JPF, testing them using ConTest are all good projects).
- Porting locking protocols described in TAMP into PThreads, and verifying using Inspect.
- Studying and writing different work-stealing queues, and verifying them.
- Writing Cilk or Cilk++ programs and studying various concurrent algorithms and their performance.
- Studying something we won't cover, but is very popular – such as Intel's TBB – is good.
- A hardware project, perhaps using FPGAs (say, build a directory based cache controller).
- Writing larger MPI / PThread programs for solving various applications, and using our Raven cluster for actual experimentation.

  One might write a simple software distributed shared memory system using MPI for instance. Another might be a memory re-distribution mechanism (motivated by Siegel's MADRE paper).
- Studying and re-implementing the Goldilocks Java Runtime for online race checking (based on the work of Elmas/Tasiran/Qadeer)
- Understanding and experimenting (TBD) with the latest proposals for the Java and C++ memory models.

Week6, 2/16-18: Ganesh away. Overview of MCAPI by Subodh, Exercises using MCAPI

Week7, 2/23-25: Programming in Erlang, Executing state transition semantics of MPI (exercises). This gives students an intro to functional programming, and its use for rapid prototyping. Erlang is also receiving serious attention for multi-core programming, as it supports threads.

Week8, 3/2-4: Overview of Promela and SPIN. Understanding safety and liveness verification.

Week9, 3/9-11: Shared memory consistency models, Exercise coding memory model in Promela, and verifying using SPIN.

This exercise will also involve the use of Vector Clocks – an important conceptual tool fundamental to many aspects of distributed computing!

If possible, we may obtain and run ARCHTEST also.

Spring Break 3/16-18

Week10, 3/23-25: Locking protocols, effect of weak memory models on simple locking protocols. Illustration using SPIN.

Week11, 3/30 and 4/1: Ganesh away on 3/30. Students get help in class on 3/30. On 4/1, will present Chapters from TAMP

Week12, 4/6-4/8: TAMP, Illustration using JPF.

Week13, 4/13-4/15: TAMP (Ganesh away, will be presented by others.)

Week14, 4/20-4/22: TAMP

Week15, 4/27-4/29: TAMP. **Student project submission.**