Teaching Parallel Computing to Lower-Division Undergraduates

Peter Pacheco
Departments of Computer Science and Mathematics
University of San Francisco
peter@usfca.edu
Introduction

• Parallel hardware has arrived.
• But most CS students can’t program it.
• The vast majority of CS students have limited exposure to parallelism
• Most see concurrency in operating systems
• A small minority take upper division classes in parallel and/or distributed computing
• Ideal solution: integrate parallelism throughout the CS curriculum
Introduction, cont.

- Problem: This could take a while
- Our solution:
  a) Upper division/master’s level class in parallel computing
  b) Project-based courses and directed study for upper-division students
  c) A required lower division class in parallel computing
Introduction, cont.

• This talk: Focus on the required lower division course
• First offered fall 2004 -- last fall we offered it for the sixth time
• Fundamental Idea: Students need to start writing parallel programs early
• Formalism and rigor are less important than hands-on experience and starting to “think in parallel”
Outline

• Course organization
• Infrastructure
• What we teach
• What we don’t teach
• Issues
• What we’ve learned
• Conclusions
Course Organization

• Prerequisites: B or better in Intro to CS 1 (Python) or C or better in Intro to CS 2 (Java)
• Most students: first semester sophomores. A few freshmen.
• No more than 20 students, usually 15-18.
• 200 minutes per week
• Main topics: C, MPI, Pthreads, OpenMP
• Kernighan and Ritchie is the only required text
Course Organization, cont

• Coursework:
  • Weekly homework assignments: typically “modify this program so that it does . . .”
  • Five programming assignments: two serial C, two MPI, one Pthreads
  • Two midterms and a final (short answer, find output, write code)
Infrastructure

- 24 node Infiniband connected cluster
- Cluster nodes: two dual core Opterons
- Three 8-core shared memory systems, one 48-core system. Two Opteron-based, two Xeon-based.
- All systems run versions of Fedora Core
- Use gcc for Pthreads and OpenMP
- Use Mvapich2 for MPI
- Home grown scheduler for interactive cluster sessions
- Great sys admin
Infrastructure, cont.

• Special classroom:
• Amphitheater structured, seats a max of 30
• Two large plasma screens, whiteboard, projector, document camera
• Small LCD’s for code display when students are facing forward
• LCD’s, keyboard and mouse behind students
• Typical uses: discuss program, have students turn around and modify it. Have each student run one set of input to generate timing data.
What We Teach

- Very brief overview of parallel algorithms and architectures; speedup, efficiency, Amdahl’s law
- Basic C (very quickly)
- Dynamic data structures in C
- Basic MPI: Init, Inquiry, Send, Recv, Finalize
- Most MPI Collectives
- Basic Pthreads: create, join, rank kludge
- Mutexes, semaphores, condition variables, barriers, read-write locks
What We Teach, cont

• Thread safety
• Cache coherence, false sharing
• OpenMP: parallel, parallel for, critical directives
• OpenMP: num_threads, reduction, default, shared, private, static, dynamic clauses
• OpenMP: library functions: get_num_threads, get_thread_num
What We Don’t Teach

- Derived datatypes
- Group, communicator, topology functions
- MPI-2
- Do discuss communication semantics, but don’t require use of alternate forms of Send and Recv
- No Pthreads attributes, thread scheduling, cancellation
- No OpenMP section(s), single, . . .
What We Don’t Teach, cont

• No formal discussion of correctness
• Discussions of performance are empirical
Issues

• Dynamic data structures in C
• Programming assignments: very hard to come up with assignments that are accessible to the students
• Debuggers (or lack thereof)
• Order of topic coverage: e.g., distributed memory or shared memory first?
• No time for significant project in last parallel system (e.g., OpenMP)
What We’ve Learned

- At USF it definitely helps that there’s another class that’s the acknowledged killer.
- At this level, most students have problems understanding abstract discussions of topics such as race conditions: it really helps to have them see the problem happen in a real program.
- Don’t expect them to discover how to write parallel programs: give them lots of guidance.
- Stick to standards.
Conclusions

- Does it work?
- None of our students has gone on to win a Gordon Bell Prize
- However, the students are very enthusiastic
- LLNL and LBL are very happy to hire our students as employees and interns
- Several have gone to grad school to study parallel computing
- The OS prof is delighted
More Information

• http://cs.usfca.edu/peter/cs220