CS 309: Parallel programming

1/4/16
Why parallel programming?

• Essentially all computers are now parallel computers
  – multicore = multiple processing elements
  – need parallel computing to harness these abilities

• It’s fun!
High-Performance Computing (HPC)

- Traditional domain of parallel computing
- Specialists using large, expensive computers for important problems
- Largely science motivated
- Typical problems: matrix operations
- Typical tools: MPI w/ OpenMP and/or CUDA
Tension between HPC and “new” parallel computing

• HPC has developed the tools, but most applications are different

• Smaller systems and all programmers

• Issue of programmer productivity
What we’ll do

- Cover basic principles for thinking about parallel programming
- Apply these to a variety of paradigms and problems
- Program in C, Java, and Chapel (productivity-oriented parallel language with HPC roots)
- Run code at (slightly) larger scale using Amazon Web Services (AWS)
- Include a project with which you can explore a topic of your choice in more detail
Class details

- Meet MWF (here)

- Textbook: Parallel programming for multicore and cluster systems (2nd ed) by Rauber and Rünger

- Also need to get a clicker
Read the rest of your syllabus

• Late work policy
  – 2 free “late days”

• Absences
  – Attendance is more explicitly required than in the past because of participation credit

• Policy on collaboration
Looking ahead

• Wednesday: Read 1.1-1.3, 2.1-2.2, and 3.1-3.2
  – Lots of definitions; try to understand the relationships between them
  – Note things that we should talk about
• Friday: Read 3.3-3.4
“The free lunch is over: A fundamental turn toward concurrency in software”
by Herb Sutter
Dr. Dobbs Journal, 2005.

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

Read to and ideally thru the section “Obstacles, and why you don’t have 10 GHz Today”
Since the article
How willing should chip designers be to change the instruction set to improve performance?

A. They should only add features that users (programmers/compiler writers) can ignore if they want
B. They should only make changes that help users
C. They should cause only minor user inconvenience and only if it gives significant gains
D. Changes and inconvenience should be weighed carefully and it’s possible that big changes could be justified
E. Bring it on!