Administrivia

- (How many will be here Thursday?)
- Homework 2 officially due today at midnight. If that's not possible for you as long as it's in by class time Thursday, okay. Don't worry if performance is not great. This is a first pass — goal is to get something that produces more or less the same results as a sequential program, experiment a bit with different seeds for RNG.

Slide 1

More Administrivia About Exam 1: Would anyone object if I proposed just dropping it? both exams? Grade then would be determined by homeworks (of which there would be slightly more), the to-be-discussed project, and attendance/participation. (This is a big enough change that I won't do it if at least one student strongly objects. If you do but don't want to speak up now, send me e-mail. If I get no such e-mail by midnight tonight, no exams.)





- So far we've talked about some "big picture of parallel computing" stuff, plus nuts and bolts of three specific programming environments the low-level stuff you need in order to be able to write programs.
- Now we're going to recap that in a more general way as a discussion of "implementation mechanisms". Why do this? Could express in design pattern terms:

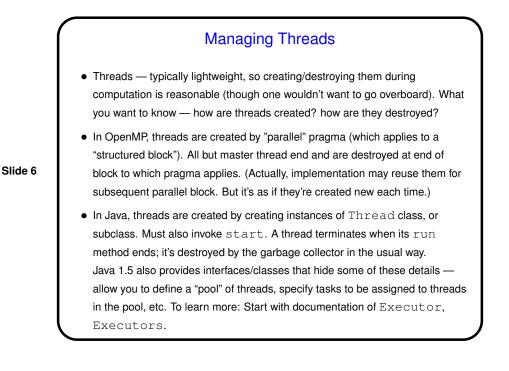
Problem: Where do you start in learning how to use a new parallel programming "environment" (language, language extension, or library)? *Context:* In learning a new sequential programming language, ask about writing assignment, if/then/else, loops, etc. Are there analogous "building blocks" for parallel programming? *Solution:* General discussion of (1) UE management, (2) synchronization, and

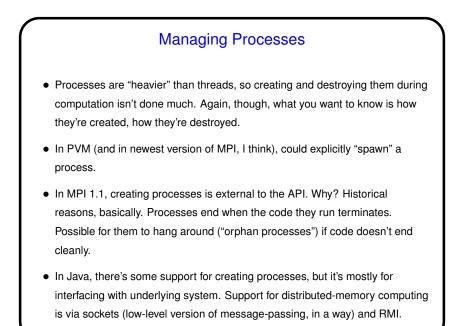
(3) communication.

UE Management

 "UE"? In MPI we have processes. In OpenMP we have (implicit) threads. In Java we have threads. Common theme — something that carries out computations. Generally have several of these running concurrently. Our generic term — "unit of execution" (UE).

- In general, what you want to know is how these are created and destroyed.
- Discuss separately for processes and threads ...





Synchronization" — very generic term, idea is to enforce constraints on order in which things execute in different UEs. Examples:
If one thread holds a particular lock, all other threads wanting the lock must wait.
A process executing a blocking "receive a message" operation must wait until the message arrives (which implies that it's been sent, etc.).
Different systems/environments provide different ways of doing this — locks, message-passing, other "synchronization mechanisms" discussed in operating systems courses/texts (semaphores, monitors, etc.). What you want to know, when learning a new language/library, is what it provides along these lines. Look at categories of mostly-commonly-needed functionality ...

Slide 8

Slide 9

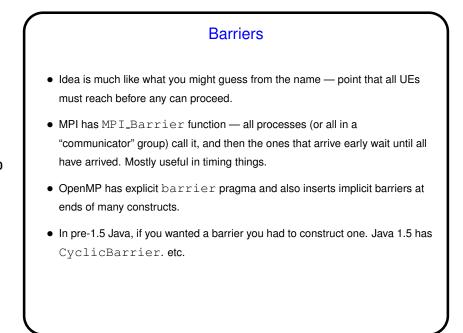
Memory Synchronization and Fences

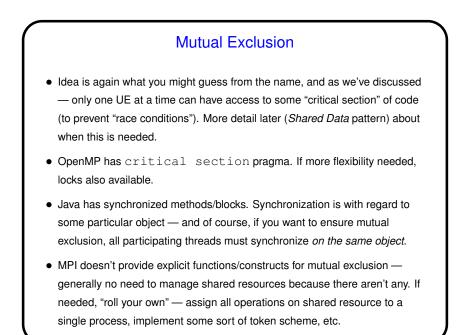
• Additional complication in shared-memory systems:

In the simple classical model of how things work, reads/writes to memory are "atomic" (execute without interference from other UEs).

Reality these days is somewhat different — hardware may cache values, compiler may do interesting optimizations, etc., etc.

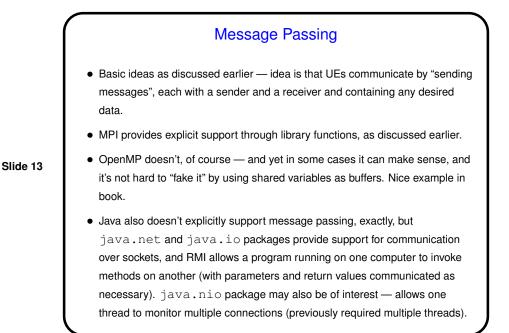
- How to know when there's a consistent view of memory all UEs share? "Memory fence" idea — writes before the fence visible to reads after it, etc.
- Memory fences usually implicit in higher-level constructs, but you could need to know about them if threads share variables that change during execution, and access to the variable isn't controlled by some sort of synchronization (OpenMP critical section, Java synchronized block, etc.).
- More details in chapter 6, with examples ...

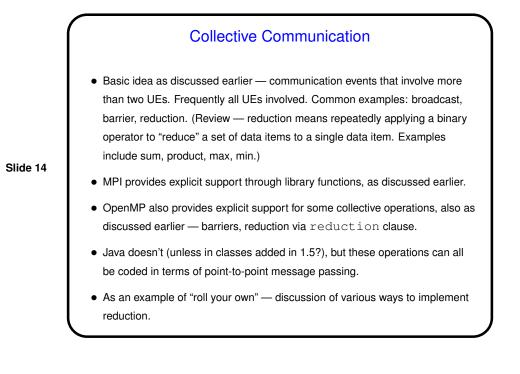


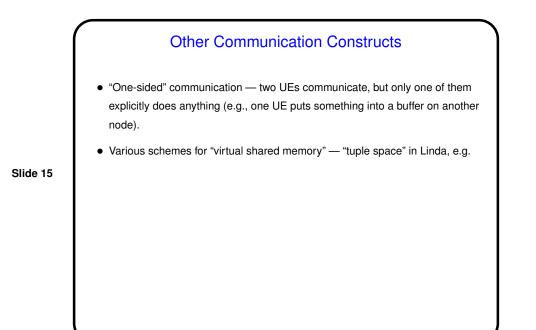


Slide 11

Communication
In the shared-memory model, communication (sharing information) among UEs is easy (trivial, really) but synchronization is difficult.
In the distributed-memory model, other way around.
Look at two basic categories of functionality ...







Minute Essay • Tell me about your experiences doing Homework 2. What was difficult? What was easy? What was interesting? Which of the three parallel programming environments (MPI, OpenMP, Java) do you feel most comfortable with at this point?