Slide 1

Administrivia

- Reminder: Homework 1 due today (MPI part) and Thursday (Java part).

Slide 2

Minute Essay From Last Lecture

- (Review sample answer.)
- Are race conditions a problem if methods are synchronized? (Not if it's done right!)
- Could one thread update a variable and another thread not "see" the updated variable? (Not if all access is in synchronized methods.)
- "What happens if threads get out of order?" (You're supposed to make that not happen.)

Slide 3

Synchronization in Java, Continued

- `synchronized` methods/blocks can be used to ensure that only one thread at a time accesses some shared variable.
- For more complex synchronization problems, can use `wait` and `notify` (or `notifyAll`):

  `wait` suspends executing thread (adds to "wait set").

  `notify` wakes up one thread from the wait set. `notifyAll` wakes up all threads. Waked-up thread(s) then compete to reacquire lock and continue execution.

  Can only be done from within synchronized method/block.

  Typical idiom — loop to check condition, `wait`.
- Example — bounded buffer class (`BoundedBuffer.java`, `TestBoundedBuffer.java` on sample programs page).

Slide 4

Controlling Threads in Java

- Preferred method of controlling one thread from another uses "interrupted" status. (Early version of Java provided other methods, e.g., `stop` — now deprecated.)
- Set status with `interrupt` (instance method).
- Check status with `isInterrupted` (instance method) or `interrupted` (static method), or by catching `InterruptedException` thrown by `wait`, `sleep`, `join`, etc.
- Example — bounded buffer test program (`TestBoundedBuffer.java` on sample programs page).

Slide 5

Features for Multithreading Added in Java 5.0

- Lots of stuff for concurrent programming added in Java 5.0 (a.k.a. 1.5). Short examples in later versions of "hello world" program (`Hello3.java`, `Hello4.java`, `Hello5.java` on sample programs page.
- Look at API for `java.util.concurrent` for more . . .

Slide 6

Not-So-Simple Point-to-Point Communication in MPI, Again

- For not-too-long messages and when readability is more important than performance, `MPI_Send` and `MPI_Recv` are probably fine.
- If messages are long, however, buffering can be a problem, and can even lead to deadlock. Also, sometimes it's nice to be able to overlap computation and communication.
- Therefore, MPI offers several other kinds of send/receive functions, including:
  - Synchronous (`MPI_Ssend`, `MPI_Recv`) — blocks both sender and receiver until communication can occur.
  - Non-blocking send/receive (`MPI_Isend`, `MPI_Irecv`, `MPI_Wait`) — doesn't block, program must explicitly test/wait.
  - Which is faster/better? probably best to try them and find out. (Sample programs `exchange*`.)

Slide 7

Minute Essay

- This wraps up the quick PAD I-level tour of our three environments. Any questions at this point?