Slide 1

Administrivia

- Sample programs page updated to include all code shown in class (except today's example, coming soon).
- Reminder: Homeworks not accepted past Wednesday at 5pm. Grades/comments coming soon by e-mail.
- Projects due at scheduled final time (Tuesday 12/12 8:30am). We need a bit more than an hour for project presentations. Start at 10am?
- Office hours this week to be announced by e-mail.
- Questions about grading?

Slide 2

Distributed-Memory Programming in Java — Review

- Most common situation is probably client/server. Two basic approaches:
  - Regular network communication using sockets.
  - RMI (Remote Method Invocation).
- First a quick review of basic ideas, then review/finish example from last time.

Slide 3

Client/Server in Java — Sockets

- Server sets up "server socket" specifying port number, then waits to accept connections. Connection generates socket. Often start a new thread to communicate with client.
- Client connects to server by giving name/IPA and port number — generates a socket.
- On each side, get input/output streams for socket. Must define some "protocol" for two sides to communicate (basically, details of what each side will send and receive, in what order).

Slide 4

Client/Server in Java — RMI

- Motivation — for client/server applications, can be annoying to have to design your own protocol.
- Instead, idea is to define "remote objects" that can be treated (at program level) like any other objects — invoke methods.
- Typical use in client/server program:
  - Server creates some remote objects and "registers" them.
  - Clients look up server's remote objects and invoke their methods.
  - Both sides can pass around references to other remote objects.
- Dynamic code loading possible too.

Slide 5

RMI, Quick How-To

- Define a class for remote objects:
  - Define interface that extends `Remote`
  - Define class that implements that interface, extends a Java remote object class. Can also include other methods, only available locally.
  - Write code using classes — if using as remote object, reference interface; otherwise can reference class.
- Compile and execute:
  - Compile as usual, *plus* run `rmic` to generate "stubs" to be used in communicating with remote objects as remote objects.
  - Make classes network-accessible.
  - Start `rmiregistry`.
  - Run server and clients as usual.

Slide 6

Review of Course

- "PAD I for parallel programming"? We covered:
  - Three languages/libraries — OpenMP, MPI, Java.
  - How to find and exploit concurrency in programs.
- We also did several running examples and some homeworks . . .

Slide 7

Review of Homeworks

- Homeworks 1 and 2 — estimating π with Monte Carlo methods. Basic structure is *Task Parallelism*. Complication is that you need a thread-safe RNG.
- Homework 3 — Conway's game of life. Basic structure is *Geometric Decomposition*. Basic idea easy, details a bit messy (especially in C).
- Homework 4 — quicksort. Basic structure is *Divide and Conquer*.
- For all programs, probably need large problem sizes to get any benefit from multiple UEs.

Slide 8

Minute Essay

- How did the course compare with your expectations/goals? Did you learn what you hoped to learn?