Slide 1

Administrivia

- Homework 2 writeup to be on Web later today or early tomorrow. Due next Thursday. Some background/discussion today in class.

Slide 2

Multithreaded Programming in Java — A Bit More

- Corrected/expanded bounded-buffer example (`BoundedBuffer.java` and `TestBoundedBuffer.java` on sample programs page).
- Typical use of threads in Java is in a GUI program, to hide latency and/or do some sort of background activity (e.g., animation). Probably can do most things with newer Swing features rather than with explicit threads. Review "How to Use Threads" in Sun's Swing tutorial.
- Other uses of threads — hide latency (e.g., in a server program, to start a new thread for each client), improve performance on multiprocessor machines (was rare, maybe will be less so). Many new features in Java 1.5 (`java.util.concurrent` package).

Slide 3

Homework 2 Background

- In Homework 2, you will make a first pass at writing a set of programs (one using MPI, one using OpenMP, and one using Java) to solve the following problem. (We'll talk more about it in class after you've tried it.)
- We talked about computing π using numerical integration. Another interesting (surprising?) approach uses a "Monte Carlo" method:

  Consider a square with sides of length 2 (any unit you like), enclosing a circle of radius 1.

  Approximate the area of the circle by "throwing darts" at the square, counting how many fall within the circle, and calculating the ratio of those within the circle to the total number.

  Model "throwing darts" by using pseudorandom number generator to generate coordinates of a point.

Slide 4

And Now For Something Somewhat (not completely) Different

- Early in the semester I said — this is "PAD I for parallel programming". In PAD I you learn
  - Details — what programs look like, how to write/compile/run them, simplified mental model of how this relates to hardware.
  - "Principles of algorithm design."
- Material so far has been mostly in that first category — but we need it as background for the "principles" part.
- Now we'll turn to the more abstract stuff about algorithm design — our "pattern language".

Slide 5

A Few Words About Design Patterns

- Idea originated with architect Christopher Alexander (first book 1977). Briefly — look for problems that have to be solved over and over, and try to come up with "expert" solution, write it in a form accessible to others. Usually this means adopting some "pattern format" to use for all patterns.

  A good pattern has an "aha!" aspect.
- First used in CS in OOD/OOP, about 1987. Really started to take off in OO community with "Gang of Four" book (Gamma, Helms, Johnson, and Vlissides; 1995). Now can find people writing patterns in many, many areas.
- To give you the idea — look at some simple patterns, e.g.,

  `http://www.cs.duke.edu/~ola/patterns/plopd/loops.html`.

Slide 6

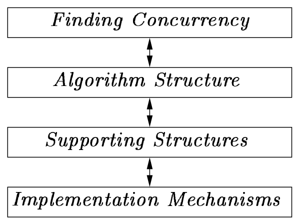
"A Pattern Language for Parallel Programming"?

- Goal of our book (and preceding work) — apply this idea in parallel computing.
- We started out looking for patterns representing high-level structures for parallel programs, thinking there might be a dozen of them.
- At some point we realized we also wanted to talk about how you get from the original problem to one of these structures — i.e., how do expert parallel programmers think about how to decompose a problem, etc.? and also about commonly-occurring data structures and program structures, and how to map high-level designs/structures into real programming environments.
- Eventually — four-layer "pattern language". (Notice that "pattern language" connotes common vocabulary more than grammatical structure. Not a programming language!)

Slide 7

Overall Organization of Our Pattern Language

- Four “design spaces” corresponding to phases in design:



- Idea is that you start at the top, work your way down, possibly with some backtracking.

Slide 8

Overall Organization of Our Pattern Language, Continued

- *Finding Concurrency* patterns — how to decompose problems, analyze decomposition.
- *Algorithm Structure* patterns — high-level program structures.
- *Supporting Structure* patterns — program structures (e.g., SPMD, fork/join), data structures (e.g., shared queue).
- *Implementation Mechanisms* — no patterns, but generic discussion of “building blocks” provided by programming environments. Details of three specific environments in appendices.

Minute Essay

- Have you heard of design patterns before? worked with them / studied them? (e.g., in Dr. Lewis's OOP seminar class?)