Slide 1

Administrivia

- (None.)

Slide 2

Minute Essay From Previous Lecture (9/29)

- What people found interesting about Homework 1:
  - Comparison of OpenMP, MPI, Java.
  - How using more UEs stops helping at some point.
  - How accuracy sometimes decreased with more UEs.
  - Setting up for passwordless ssh.
- What people found difficult about Homework 1:
  - Getting into the right mindset (but not very difficult).
  - Getting good time results.
  - Getting familiar with OpenMP / MPI / Java.
  - Details of setting up for MPI.

Slide 3

Homework 1 Revisited — Sequential Programs

- First step is probably to run sequential programs a few times. (Using what machines? what parameters?)
- (Look at some results I generated . . .)
- Do results vary depending on seed? (Yes.)
- Are results better for more samples? (Sometimes.)
- Are results the same for C and Java programs? (No.)
- Does execution time make sense — fairly consistent from run to run, scales with number of samples? from machine to machine? (Yes.)

Slide 4

Homework 1 Revisited — Parallel Programs

- My idea was that you would do something very similar to what we did with numerical integration:
  - Consider each “throw a dart” operation as a task.
  - Divide tasks among UEs, with each of them computing a local count.
  - Combine local counts at the end, and then compute pi.

  (Most people did this, pretty much, with a few variations.)
- Recall that for numerical integration we got different results for different numbers of UEs because floating-point addition is not associative. Will that happen here? (It shouldn’t!)

Slide 5

Homework 1 Revisited — Parallel Programs, Continued

- Probably should repeat sequential-program experiments, right? with same inputs, but varying numbers of UEs. (How many UEs should we use?)
- And if we do that, results can be — "interesting"?
  - Different answers depending on number of UEs. (How can that be? Is the answer the same for OpenMP, Java, and MPI?)
  - Disappointing performance (but maybe not for all three versions?)
- What's going on? well, maybe we should step back and talk about "generating random numbers" . . .

Slide 6

A Little About Random Numbers

- (Canonical reference — discussion in volume 2 of Knuth's *The Art of Computer Programming*. Very mathematical. Other references may be easier.)
- Many application areas that depend on "random" numbers (whatever we mean by that) — simulation (of physical phenomena), sampling, numerical analysis (Monte Carlo methods, e.g.), etc.
- Early on, people used physical methods (currently still in use in lotteries), and thought about building hardware to generate "random" results. No good large-scale solution, though, and besides it seemed useful to be able to repeat a calculation.
- Hence need for "random number generator" (RNG) — way to generate "random" sequences of elements from a given set (e.g., integers or doubles). Tricky topic. Many early researchers got it wrong. Many application writers

aren't interested in details.

Slide 7

Desirable Properties of RNG — "Randomness"

- Obviously a key goal, if tricky to define. A thought-experiment definition: Suppose we're generating integers in the range from 1 through d, and we let an observer examine as much of the sequence as desired, and ask for a guess for any other element in the sequence. If the probability of the guess being right is more than $1/d$, the sequence isn't random.
- Also want uniformity — for each element, equal probability of getting any of the possible values.
- For some applications, also need to consider "uniformity in higher dimensions": Consider treating sequence as sequence of points in 2D, 3D, etc., space. Are the points spread out evenly?

Slide 8

Slide 9

Minute Essay

- What kind of experiments might be useful in figuring out whether a random sequence is “good” for the Monte Carlo pi problem?