Slide 1

Administrivia

- Reminder: Homework 1 OpenMP program due today at 11:59pm. This really is meant to be a first pass at producing a good program, so if you have somthing that gives correct results but performs very badly — turn it in and plan to fix it after class discussion next week.

  And in general . . . If you're not finished with a programming assignment by the deadline, *turn in what you have* with a note that you'll be submitting a beter version later.

  Please remember to mention the course and the assignment in the subject line. No Google-Drive shares please! If working remotely, consider using `mail-files` script (see "sample programs" page) to send mail from command line (so attaching a file is easy even if it's on the remote system).
- Reminder: Homework 1 MPI program due Monday.

Slide 2

More Administrivia

- At least one copy of textbook on reserve in the library. 1-day reserve, which I hope will give those without their own copies a reasonable chance . . . ?
- Readings for this class and the next updated.

Slide 3

Running OpenMP and MPI Programs, Revisited

- As mentioned last time, for most if not all programs we write for this class, we'll be interested in finding out how they "scale" with varying numbers of "units of execution" (processes or threads).
- To make this interesting for OpenMP, you need a machine with as many cores as possible. It's AOK to develop anywhere, but really it's probably a good idea to do final timing experiments on `Dione`, which has — well, supposedly 64 processing elements, though I wonder.
- To make this interesting for MPI, you need to use multiple machines; you should probably try both with one process per machine and more than one per machine (up to the number of cores on the machine). Of course, if you use multiple machines they all have to be running Linux. Machines good for this purpose are `PandoraNN` (NN from (`01` to (`08` — there is also a `Pandora00` but it's meant as a file server).

Slide 4

MPI — Review/Recap

- You can do a lot with just a few things — initialize/finalize, simple send/receive, simple collective communication.
- To execute programs you need `mpirun`, sometimes with the `--prefix` flag. To run on multiple machines, either use `-host` and list their names (separated by commas not spaces) or put names in a file and use `-hostfile`.
- (Review numerical-integration example.)

Slide 5

Multithreaded Programming in Java — Overview

- We'll look next at basic multithreaded programming in Java, mostly focusing on lower-level approaches.
- Why Java? When we wrote the book it was a language many people already knew, including Trinity CS students (after CS2). Now still popular but not used in our required courses. We'll use it this year partly so the examples make sense but also as a way of giving you some exposure to another popular language.

Slide 6

Introduction to Java for Scala Programmers

- Scala is built on top of Java and shares a common runtime environment (plus Scala has access to Java's huge library).
- Syntactically, however, Scala is more like C++.
- Unlike Scala (but like C/C++), Java has no REPL environment (alas). You must compile (to "byte code") and then execute using runtime system. (This two-step approach works for Scala too.)
- Unlike Scala (but like C/C++), programs all have to include some "boilerplate" lines that set things up for the main program.
- Unlike either Scala or C++, the compiler and runtime system are picky about filenames.

Slide 7

“Hello World” in Java

- Define class `Hello` in file `Hello.java`.
- (You can use Eclipse for Java, but for short programs I don’t, and sometimes (especially for this class) it’s better to run from the command line. So I’ll show command-line tools only.)
- Compile with `javac Hello.java`. If it succeeds, generates a file `Hello.class`. (To reduce clutter, add `-d objectdir`.)
- Execute with `java Hello`. (If you compiled with `-d`, add `-cp objectdir`.)

Slide 8

Java for Scala Programmers, Continued

- Unlike either Scala, C, or C++, everything in Java is part of some class. Regular (non-local) variables and methods are associated with instances of the enclosing class `static` variables or methods are associated with the class as a whole. (Scala’s “companion objects” provide similar functionality.)
- As in C, variables have to be declared, with a type, and declarations look more like C/C++. No `var` or `val`.

  Variable types include “primitives” (lowercase type name, similar to C variables) and “references” (uppercase type name corresponding to a class, similar to Scala variables). Why oh why? Attempt at efficiency.
- Syntax for function declarations is more like C/C++ than Scala.
- Much low-level syntax is the same as C/C++. Classes are the same idea but with slightly different syntax, more similar to C++.
- (Simple example(s)?)

Java for Scala Programmers, Continued

Slide 9

- Like Scala, Java has a notion of grouping classes into packages, and syntax is similar (maybe not a huge surprise?). Unlike Scala, the compiler and runtime system are picky about file names and directory structure.
- At this point we can look at some ways to write multi-threaded “hello world” . . .

Minute Essay

Slide 10

- Anything interesting to report about the part of Homework 1 you’ve done? no need to repeat (in detail anyway) what you said in the discussion you turned in with your code.