Slide 1

Administrivia

- Reminder: First installment of Homework 1 (OpenMP program) due today (at 11:59pm). Next installment (MPI program) due Tuesday.

  (If you started from program `num-int-par-spmd-*.c` on the sample programs page — that was not the code we looked at in class, and it's not the code I intended you to start from. I've temporarily removed it and suggest that you try again (and you can turn the code in tomorrow for full credit).)

Slide 2

Minute Essay From Last Lecture

- (Review.)
- Comments/questions from previous minute essays:

  How do multithreaded programs adapt to varying numbers of cores? Do these programs run worse on single-core systems? (It depends, and sometimes!)

  Why was parallel computing "on the horizon" for so long? (Good question! probably has something to do with continued increase in single-processor speed, though.)

Slide 3

Parallel Programming in Java

- Java supports multithreaded (shared-memory parallel) programming as part of the language — `synchronized` keyword, `wait` and `notify` methods of `Object` class, `Thread` class. Programs that use the GUI classes (AWT or Swing) multithreaded under the hood. Justification probably has more to do with hiding latency than HPC, but still useful, and recent versions (5.0 and beyond) includes much useful library stuff.
- Java also provides support for forms of distributed-memory programming, through library classes for networking, I/O (`java.nio`), and Remote Method Invocation (RMI).

Slide 4

What Does A Multithreaded Java Program Look Like?

- Easy answer: Like a regular Java program. (In fact, any program with a GUI . . .)
- Programming model: All threads share a common address space. Programmer is responsible for creating threads, providing synchronization, etc.

Slide 5

Creating Threads in Java

- Threads are all instances of `Thread` class (or a subclass). Pre-5.0, two ways to create threads:
  - Create a subclass of `Thread` (frowned on by o-o purists).
  - Create a `Thread` using an object that implements `Runnable` (preferable).

  Either way, `run` method (of subclass of `Thread`, or of `Runnable`) contains code for thread to execute.
- Start thread with `start` method. Can wait for it to finish with `join`.
- "Hello world" example (`Hello1.java` and `Hello2.java` on sample programs page). (Other methods in `java.util.concurrency` — see sample programs `Hello3.java`, `Hello4.java`, `Hello5.java`.)

Slide 6

Java from the Command Line

- Most of you probably use Eclipse to write Java programs. You can do that for this course too, but for this course you might prefer to run them from the command line (to make it easier to supply command-line arguments if nothing else). Command to use is `java`, followed by class name and any arguments.
- You can also write them using your favorite text editor compile from the command line. Command to compile is `javac`.

Slide 7

Shared Variables in Java

- Code executed by a thread is some object's `run` method. Access to variables is consistent with usual Java scoping — class/instance variables, parameters, etc.
- As we noted before, though, simultaneous access to shared variables can be risky, however. So . . .

Slide 8

Synchronization in Java

- Interaction among threads in Java based on "monitor" idea (Hoare (1975) and Brinch Hansen (1975)).
- Every object has implicit lock; `synchronized` keyword means "only run this when you have the relevant lock" — if another thread has the lock, wait. Can be used to ensure one-at-a-time access to critical variables.

  "Relevant lock"? For synchronized methods, lock for object (instance methods) or class (static methods). For synchronized blocks, you specify the object.

  Example — `HelloSynch*.java` on sample programs page.
- `wait` and `notify` methods allow more interesting kinds of coordination. But first . . .

Slide 9

Numerical Integration Example, Revisited

- How to parallelize using Java? well, first must rewrite in Java (`NumIntSeq.java` on sample programs page).
- Now rewrite to use multiple threads, based on same strategy we used for OpenMP — split loop iterations among threads, give each its own copy of work variables, compute sum based on "reduction" idea. Some things must be done more explicitly in Java (making the program in some ways more like MPI's SPMD model). See `NumIntPar.java` on sample programs page.

Slide 10

Minute Essay

- `synchronized` has benefits — avoiding multiple threads changing a shared variable at the same time. What risks/disadvantages can you imagine that it might have?

Slide 11

Minute Essay Answer

- Among the possible problems are the possibility of deadlock, and performance that's worse than it needs to be.