Slide 1

Administrivia

- Sample programs under "sample programs" on course Web site. ZIP file for each programming environment containing all programs for it. These files includes examples we haven't looked at yet; ignore for now.

Slide 2

Multithreaded Programming in Java, Recap/Revisited

- Basic functionality — threads, synchronization, wait/notify — has been in the language from the start.
- At some point in Java's evolution, `java.util.concurrent` added to the library. *Many* useful features. Some we won't use much in this class since focus is on details / low level.
- But "thread pools" and "futures" can be pretty useful . . .

Slide 3

Java "Thread Pools" and "Futures"

- One limitation of the original design of threads: No way to have `run` return a value.

  To change that — `Callable`, `Future`. (Note that Scala `Future`s seem to be fairly different from Java's — much more asynchronous.)

- Also, explicitly creating and starting threads a nuisance, doesn't provide an easy way to set up "worker" threads to which work can be assigned.

  "Thread pools" address that: `ExecutorService` provides various ways to set up a pool of threads and assign them work.

- (More versions of "hello world" and numerical integration.)

Slide 4

Compiling and Executing Java Programs, Recap/Review

- Sample programs include `compile-all` script. Not the most efficient way to rebuild but effective — recompiles everything, into directory `obj`.

  ```
  ./compile-all csci3366
  ```

  to compile all examples. (Can specify a subdirectory to just compile some.)

- Execute program with `java -cp obj` and full class name and arguments, e.g.,

  ```
  java -cp csci3366.sample.hello.Hello1 4
  ```

Slide 5

OpenCL — Review/Recap

- Explicitly defines computation in terms of some parts that execute on a "host computer" and others that execute on a "compute device" (typically a GPU but doesn't have to be).
- Intended to be very portable but also to not hide too much from the programmer.
- Result is that programmers must deal with a lot of low-level details. However, many of those details the same from program to program and can be encapsulated in a library. I wrote one for my own use; you can use it too.

Slide 6

OpenCL — Review/Recap, Continued

- Computation on compute device packaged as "kernels".
- Basic compute step: Execute a kernel concurrently on many "work items", one for each value in an "index range".
- Cna have more work items than processing elements, so group items into "work groups" for execution. Conceptually all work items in a kernel execution happen concurrently, but in fact they're executed in "waves".

Slide 7

Numerical Integration in OpenCL

- What does our familiar example look like in OpenCL?
- Note first: OpenCL only guaranteed to provide a single-precision floating-point type.
- Doesn't seem quite right, then, to compare results with code that computes using double precision. So I wrote a version of the sequential code that uses `float` rather than `double`, and . . . .
- Results were astonishingly bad! and in fact, they got worse with increasing numbers of samples! why . . .

Slide 8

Digression: `double` versus `float`

- It took a while to figure out, but problem turns out to be that at some point in computing `sum`, increments being added are so much smaller than the current value that they just disappear. (Recall "floating point addition not associative" example from CSCI 1120?)
- What to do?! I found a clever algorithm that helps quite a bit. (Wikipedia reference in source code.) Will this be a problem in OpenCL? maybe or maybe not; assume not to start with.

Numerical Integration in OpenCL, Continued

Slide 9

- Basic strategy — split iterations of main processing loop among UEs — same. (Remember: “UE” our generic term for thread of control, e.g., thread or process. For OpenCL, nearest analog is work item.)
- *Could* make each loop iteration a work item (as in vector addition example), but very “fine-grained”, and overhead of setting up so many work items seems like it might swamp any any performance improvement, plus there’s the problem of synchronizing access to shared sum. So adopt similar strategy as for other programmming environments we’ve discussed — compute local sums in each UE and then combine.
- But it may be tricky . . .

Numerical Integration in OpenCL, Continued

Slide 10

- Unlike OpenMP and MPI, OpenCL doesn’t have anything built in to help with reduction. So we’ll have to write our own, as we did in Java. Basic idea of computing partial sums and combining them seems reasonable, but . . .
- Synchronizing among work items can be difficult: “Barrier” synchronization available within each work group, but no way to apply it across work groups(!).

Slide 11

Numerical Integration in OpenCL, Continued

- So our strategy will be multi-level . . .
- First compute a partial sum in each work item (similar to what we did in MPI and Java, and implicitly with OpenMP).
- Then combine these into one partial sum for each work group (using barrier synchronization — wait for all work items to compute their partial sums and then have one work item combine them).
- Then have the host combine these per-work-group sums into the final sum.

Slide 12

Numerical Integration in OpenCL, Continued

- To do this we'll need to work with different levels of memory:
- Sums for work items can go in an array shared among work items but local to a work group (this is the "local memory" previously mentioned).
- Sums for work groups need to be in "global memory" (accessible to host as well).

Slide 13

Numerical Integration in OpenCL, Continued

- One other thing to know about before looking at code: When executing kernel, the number of work items (indices) has to evenly divide the workgroup size.
- That said, look at code . . .
- Unlike the other programming environments, not clear what “scalable” means, if anything.

  However, do have to choose number of loop iterations per work item and work group size. My code allows experimenting with different values through command-line options, and also has an option to run the kernel on the CPU.

Slide 14

Minute Essay

- Does Java’s version of `Future`s make sense to you? Would you agree or disagree with me that it’s easier to understand, and to use for problems such as the numerical integration example?
- Does the OpenCL version of numerical integration make (some?) sense?
- Questions?