Slide 1

Administrivia

- (None?)

Slide 2

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 3

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.

Slide 4

OpenCL — Recap/Review

- May be worth recapping / reviewing how you specify, in OpenCL, work to be done on GPU:
- Execute "computational kernel" on every element of an "index range". Each individual execution is a "work-item", which can compute for a single point of the data or many.
- Work-items are grouped into "work groups"; within a work group, all work-items execute concurrently, but work groups can be executed in sequence if necessary.
- Effect is as if all work-items were operated on concurrently, *except* that synchronizing/communicating within a work group is fine, but not possible between work groups.

Slide 5

Numerical Integration in OpenCL, Continued

- 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 . . .

Slide 6

Numerical Integration in OpenCL, Continued

- 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 7

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 8

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 9

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.