Slide 1

Administrivia

- About the RNG for Homework 2, I have code in both C and Java that I believe generates the constants needed to separate the global sequence into subsequences, one per UE. I will package it up and post it. (I don't know that this would even have been possible in strictly standard C, but it's doable with the use of the GMP library.)

Slide 2

OpenCL — Review/Recap

- OpenCL, like OpenMP and MPI, defines concepts and an API.
- In some ways it's more like OpenMP than MPI, because it defines source-code syntax as well as library functions. Why? Well . . .
- At runtime, OpenCL programs do some computation on the "host" (main processor) and some computation on the "compute device" (often the GPU — but doesn't have to be! everything designed to allow it to work with many kinds of compute devices).
- Specification, therefore, includes a library of functions and definitions of constants and "opaque" data types, as MPI does, but also a syntax for writing source code for the compute device — a somewhat modified version of C. So this means compiler changes? not exactly . . .

Slide 3

OpenCL — Compiling and Executing Programs

- C programs using OpenCL are compiled more or less like other C programs, into object code that's linked to produce an executable — *for the host computer*.

  (Some implementations, such as the AMD SDK, just call a regular C compiler, possibly requiring you to tell it where to find library files. Others, such as NVIDIA's Cuda, include a compiler, but one that calls a regular C compiler to do a lot of its work.)

- What happens to the source code for the compute device? Often (usually? normally?) it's compiled at runtime by OpenCL library functions on the host. (And yes, *they* have to compile a variant of C.)

Slide 4

OpenCL on a "Compute Device"

- Computation on an OpenCL compute device is done using a command queue and "kernels".
- A "command queue" for the device holds commands for the device to execute — data transfers between host and device memories, kernel executions, etc.

Slide 5

OpenCL on a "Compute Device", Continued

- In terms of the SIMD model, a kernel defines a single instruction stream, which will be executed effectively-in-parallel on multiple data items.
- An "index space" defines a range on which to execute a kernel.
- A "work item" is one execution of a kernel.
- "Work groups" are used to group work items: A compute device can only operate on some finite number of data items truly in parallel; to operate on a larger index space it must break it up into "work groups" and execute them one at a time. (But the effect is the same as if they all executed in parallel.)

Slide 6

OpenCL on a "Compute Device", Continued

- Memory accessible to kernels comes in different varieties:
  - Global memory — accessible to all work items, allocated and written/read by host.
  - Constant memory — similar to global memory, but read-only to work items.
  - Local memory — accessible to all work items, can be allocated by host or (statically!) in kernel.
  - Private memory — accessible to a single work item. Also static allocation only.
- All distinct from host memory.

Slide 7

OpenCL — Vector Addition Example

- As an example, consider a somewhat silly program to add two vectors, producing a third vector, with the compute device doing the actual addition. So . . .
- Sequential code for this problem is simple — a `for` loop over all elements in the vectors' index range.
- In OpenMP this would be simple. In MPI it would be a little harder. In OpenCL, it's . . . "messy"?

Slide 8

OpenCL — Vector Addition Example, Continued

- The host has to create the three vectors (as arrays), fill in the input values, copy them to device memory, tell the device to do the addition, copy the results back from device memory, and print them.
- The "kernel" the compute device will execute — it should be done once for each element of the vectors' index range (so, these are the "work items"), and what it does is basically the body of the sequential-code loop.

Slide 9

OpenCL — Program Start-Up

- You'll recall that MPI programs are supposed to start with a call to `MPI_Init`, to set up the MPI environment. OpenCL programs also start with some setup, but it's much more involved. Why? probably because designers wanted to support lots of options/environments. (Semi-aside: Also like MPI, there are functions you should call at the end of the program. See example — function in my utility library.) So . . .
- First step is to find a suitable device. Devices are typically grouped by "platforms" (e.g., CPU, NVIDIA GPU). Library functions let you find available platforms and then within each one look for devices of whatever type you're interested in. (See example — more functions in my utility library.) Result is a "device ID".

Slide 10

OpenCL — Program Start-Up, Continued

- Once you have a device ID, you can start setting things up to use it:
- Create a "context" — information about device, associated memory, etc.
- Create a command queue for the device.
- Create a "program object" — basically a dynamically-generated library of kernels, built by compiling at runtime from source, possibly defined as a (rather long!) string within the host source program.

Slide 11

OpenCL — Data Transfer

- A complication in OpenCL programming is the need to transfer data from host to compute device, and vice versa. Details of doing this are, well, detailed.
- First step is to create “buffers”. (See example.)
- To actually do the copying, you put on the device’s command queue a command to write to or read from a buffer, specifying the destination or source location in the host. (See example.)

Slide 12

OpenCL — Executing Kernels

- To execute a kernel — well, again, it’s messy! (See example.)
- First step is to build the kernel from the program object.
- Next step is to set up arguments.
- And then to execute the kernel, you again put a command on the command queue to do so, specifying the kernel, the index range, and the size of the “workgroup”. (An inquiry function lets you find out the maximum for that.)
- Finally, you wait for the command to finish.

Slide 13

Minute Essay

- Anything particularly unclear today?
- How are you doing with Homework 2?