Introduction to OpenCL and GPU Programming

Katharine Hyatt

February 12, 2012

Outline

What is OpenCL?

What is OpenCL?

- Extension of the C programming language

What is OpenCL?

- Extension of the C programming language
- Allows control of heterogenous systems

What is OpenCL?

- Extension of the C programming language
- Allows control of heterogenous systems
- Code runs on CPU, GPU, Cell.

What is OpenCL?

- Extension of the C programming language
- Allows control of heterogenous systems
- Code runs on CPU, GPU, Cell.
- Open standard developed by OpenCL group

Why use a GPU?

Why use a GPU?

- GPUs designed for massively parallel computing

Why use a GPU?

- GPUs designed for massively parallel computing
- Multiple-instruction-multiple-data architecture

Why use a GPU?

- GPUs designed for massively parallel computing
- Multiple-instruction-multiple-data architecture
- Use asynchronous control between host and device

Why use a GPU?

- GPUs designed for massively parallel computing
- Multiple-instruction-multiple-data architecture
- Use asynchronous control between host and device
- Effective for some CPU-infeasible problems

Why use a GPU?

- GPUs designed for massively parallel computing
- Multiple-instruction-multiple-data architecture
- Use asynchronous control between host and device
- Effective for some CPU-infeasible problems
- Far cheaper per GFLOP than CPUs

OpenCL vs Alternatives

OpenCL vs Alternatives

- OpenCL is a cross-platform open standard

OpenCL vs Alternatives

- OpenCL is a cross-platform open standard
- CUDA has closed source components

OpenCL vs Alternatives

- OpenCL is a cross-platform open standard
- CUDA has closed source components
- So far, CUDA only works on nVidia

OpenCL vs Alternatives

- OpenCL is a cross-platform open standard
- CUDA has closed source components
- So far, CUDA only works on nVidia
- MP is CPU-only

OpenCL vs Alternatives

- OpenCL is a cross-platform open standard
- CUDA has closed source components
- So far, CUDA only works on nVidia
- MP is CPU-only
- OpenCL ecosystem is less developed

What do you need?

What do you need?

- Any computer with a CPU or Cell

What do you need?

- Any computer with a CPU or Cell
- Can also have a GPU (more parallelism!)

What do you need?

- Any computer with a CPU or Cell
- Can also have a GPU (more parallelism!)
- Developer driver and OpenCL compiler

How is it different?

How is it different?

- GPUs have more restrictions than CPU

How is it different?

- GPUs have more restrictions than CPU
- Designed for one task, not many

How is it different?

- GPUs have more restrictions than CPU
- Designed for one task, not many
- Performance greatly affected by two factors:

How is it different?

- GPUs have more restrictions than CPU
- Designed for one task, not many
- Performance greatly affected by two factors:
  - Memory access pattern

How is it different?

- GPUs have more restrictions than CPU
- Designed for one task, not many
- Performance greatly affected by two factors:
  - Memory access pattern
  - Instruction configuration

How is it different?

- GPUs have more restrictions than CPU
- Designed for one task, not many
- Performance greatly affected by two factors:
  - Memory access pattern
  - Instruction configuration
- Must keep track of memory spaces!

Talking to the GPU

Talking to the GPU

- All functions are "controlled" from CPU

Talking to the GPU

- All functions are "controlled" from CPU
- CPU launches a GPU fuction (kernel)

Talking to the GPU

- All functions are "controlled" from CPU
- CPU launches a GPU fuction (kernel)
- CPU regains control before function finishes

Talking to the GPU

- All functions are "controlled" from CPU
- CPU launches a GPU fuction (kernel)
- CPU regains control before function finishes
- Memory transfers can occur alongside computation

Kernels

Kernels

- Called from host - execute on device

Kernels

- Called from host - execute on device
- Function instances execute concurrently on threads

Kernels

- Called from host - execute on device
- Function instances execute concurrently on threads
- Must tell device how many threads to use

More Kernels

```
const char *particles =
"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
"__kernel␣void␣update_state(__global␣float4␣*p
"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

More Kernels

- Device performs identical operations on data

```
1  const char *particles =
2  "
3  "__kernel void update_state(__global float4 *p
4  "
5  "
6  "
7  "
```

More Kernels

- Device performs identical operations on data
- Launch kernels using task queue

```
1  const char *particles =
2  "
3  "__kernel void update_state(__global float4 *p
4  "
5  "
6  "
7  "
```

More Kernels

- Device performs identical operations on data
- Launch kernels using task queue
- Information about kernel given to device

```
1 const char *particles =
2 "
3 "__kernel void update_state(__global float4 *p
4 "
5 "
6 "
7 "
```

More Kernels

- Device performs identical operations on data
- Launch kernels using task queue
- Information about kernel given to device
- How many work-groups and work-items needed?

```
1 const char *particles =
2 "
3 "__kernel void update_state(__global float4 *p
4 "
5 "
6 "
7 "
```

More Kernels

- Device performs identical operations on data
- Launch kernels using task queue
- Information about kernel given to device
- How many work-groups and work-items needed?
- Which arguments does kernel take?

```
1 const char *particles =
2 "
3 "__kernel void update_state(__global float4 *p
4 "
5 "
6 "
7 "
```

More Kernels

- Device performs identical operations on data
- Launch kernels using task queue
- Information about kernel given to device
- How many work-groups and work-items needed?
- Which arguments does kernel take?
- Function definition passed as a string

```
1 const char *particles =
2 "
3 "__kernel void update_state(__global float4 *p
4 "
5 "
6 "
7 "
```

Work-groups and Work-items

Work-groups and Work-items

- Logical structures used to group processing

Work-groups and Work-items

- Logical structures used to group processing
- Workgroups processed independently on device cores

Work-groups and Work-items

- Logical structures used to group processing
- Workgroups processed independently on device cores
- Each workgroup contains $INTEGER wavefronts

Work-groups and Work-items

- Logical structures used to group processing
- Workgroups processed independently on device cores
- Each workgroup contains $INTEGER wavefronts
- Scheduling of workgroups handled by GPU

Work-groups and Work-items

- Logical structures used to group processing
- Workgroups processed independently on device cores
- Each workgroup contains $INTEGER wavefronts
- Scheduling of workgroups handled by GPU
- Can create more workgroups than cores

Wavefronts

Wavefronts

- Work-items per wavefront is device dependent

Wavefronts

- Work-items per wavefront is device dependent
- nVidia and some AMD cards - 32

Wavefronts

- Work-items per wavefront is device dependent
- nVidia and some AMD cards - 32
- Newer AMD cards - 64

Wavefronts

- Work-items per wavefront is device dependent
- nVidia and some AMD cards - 32
- Newer AMD cards - 64
- Different instructions within wavefront causes serialization

Wavefronts

- Work-items per wavefront is device dependent
- nVidia and some AMD cards - 32
- Newer AMD cards - 64
- Different instructions within wavefront causes serialization
- Different instructions between wavefronts is fine

Synchronization

Synchronization

- Two types of synchronization

Synchronization

- Two types of synchronization
- Work-group

Synchronization

- Two types of synchronization
- Work-group
  - Work-items in wavefront execute same instruction
- Command

Synchronization

- Two types of synchronization
- Work-group
  - Work-items in wavefront execute same instruction
  - No work-item proceeds until all finished
- Command

Synchronization

- Two types of synchronization
- Work-group
  - Work-items in wavefront execute same instruction
  - No work-item proceeds until all finished
- Command
  - Orders commands in instruction queue

Synchronization

- Two types of synchronization
- Work-group
  - Work-items in wavefront execute same instruction
  - No work-item proceeds until all finished
- Command
  - Orders commands in instruction queue
  - Change memory value $\Rightarrow$ subsequent commands notice

Host and Device Memory Spaces

Host and Device Memory Spaces

- Generally, GPU cannot access CPU memory

Host and Device Memory Spaces

- Generally, GPU cannot access CPU memory
- CPU indirectly accesses GPU through API

Host and Device Memory Spaces

- Generally, GPU cannot access CPU memory
- CPU indirectly accesses GPU through API
- Can map CPU pointers to GPU

Host and Device Memory Spaces

- Generally, GPU cannot access CPU memory
- CPU indirectly accesses GPU through API
- Can map CPU pointers to GPU
- Kernels on separate devices

Within The Kernel

Within The Kernel

- Kernel can discover information about itself

Within The Kernel

- Kernel can discover information about itself
- Location within a work-group

Within The Kernel

- Kernel can discover information about itself
- Location within a work-group
- Which workgroup contains the kernel instance

Within The Kernel

- Kernel can discover information about itself
- Location within a work-group
- Which workgroup contains the kernel instance
- Kernel can access three types of memory

Within The Kernel

- Kernel can discover information about itself
- Location within a work-group
- Which workgroup contains the kernel instance
- Kernel can access three types of memory
- Will use this later during example

Kernel Memory

Kernel Memory

- Global memory - RAM on GPU

Kernel Memory

- Global memory - RAM on GPU
  - Far from computing chip - slow access

Kernel Memory

- Global memory - RAM on GPU
  - Far from computing chip - slow access
  - More space than any other type

Kernel Memory

- Global memory - RAM on GPU
  - Far from computing chip - slow access
  - More space than any other type
- Local memory - shared within wavefront

Kernel Memory

- Global memory - RAM on GPU
  - Far from computing chip - slow access
  - More space than any other type
- Local memory - shared within wavefront
  - Physically close to chip - fast access

Kernel Memory

- Global memory - RAM on GPU
  - Far from computing chip - slow access
  - More space than any other type
- Local memory - shared within wavefront
  - Physically close to chip - fast access
  - Small amount of space available

Kernel Memory

Kernel Memory

- Private Memory - unique to work-item

Kernel Memory

- Private Memory - unique to work-item
  - Most non-local variables declared within kernel

Kernel Memory

- Private Memory - unique to work-item
  - Most non-local variables declared within kernel
  - Stored in global memory - slow

Kernel Memory

- Private Memory - unique to work-item
  - Most non-local variables declared within kernel
  - Stored in global memory - slow
- Registers - unique to work-item

Kernel Memory

- Private Memory - unique to work-item
  - Most non-local variables declared within kernel
  - Stored in global memory - slow
- Registers - unique to work-item
  - Similar to CPU registers

Kernel Memory

- Private Memory - unique to work-item
  - Most non-local variables declared within kernel
  - Stored in global memory - slow
- Registers - unique to work-item
  - Similar to CPU registers
  - Physically close to chip - fast access

What does CPU control mean?

What does CPU control mean?

- GPU memory managed from CPU code

What does CPU control mean?

- GPU memory managed from CPU code
- All kernels launched from CPU

What does CPU control mean?

- GPU memory managed from CPU code
- All kernels launched from CPU

Physical Problem

Physical Problem

- n point charges affected by potential

Physical Problem

- n point charges affected by potential
- Source located at $(0, 0)$

Physical Problem

- n point charges affected by potential
- Source located at $(0, 0)$
- Potential has $\frac{\hat{r}}{r}$ form

Physical Problem

- n point charges affected by potential
- Source located at $(0, 0)$
- Potential has $\frac{\hat{r}}{r}$ form
- For now, particles don't interact

Physical Problem

- n point charges affected by potential
- Source located at $(0, 0)$
- Potential has $\frac{\hat{r}}{r}$ form
- For now, particles don't interact
- Write OpenCL to model system

Getting Started

Getting Started

- Need to include relevant libraries

Getting Started

- Need to include relevant libraries
- Initialize OpenCL API

Getting Started

- Need to include relevant libraries
- Initialize OpenCL API
- Must detect and select usable devices

Getting Started

- Need to include relevant libraries
- Initialize OpenCL API
- Must detect and select usable devices
- Set up command queue and context

Getting Started

- Need to include relevant libraries
- Initialize OpenCL API
- Must detect and select usable devices
- Set up command queue and context
- Specify runtime compilation of kernels

Preprocessor

```
1  #include<CL/cl.h> // include the OpenCL library
2  #include<stdio.h>
```

Starting OpenCL and finding a GPU

```
1        cl_platform_id platform; //finding an appropriate platform
2        clGetPlatformIds(1, &platform, NULL); // only look for one
3
4        cl_device_id device; //finding an appropriate GPU
5        clGetDeviceIds(platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL); // only
```

Command queue and the context

```
1 cl_context context = clCreateContext(NULL, 1, &device, NULL, NULL, N
2 cl_command_queue queue = clCreateCommandQueue(context, device, 0, NULL);
```

Building a kernel

```
1    cl_program program = clCreateProgramWithSource(context, 1, &particles, N
2    clBuildProgram( program, 1, &device, NULL, NULL, NULL );
3
4    cl_kernel kernel   = clCreateKernel(program, "particles", NULL);
```

Setting Up CPU Storage

Setting Up CPU Storage

- Create initial state first on CPU

Setting Up CPU Storage

- Create initial state first on CPU
- Must copy state to GPU

Setting Up CPU Storage

- Create initial state first on CPU
- Must copy state to GPU
- Use same data structure for arrays

Setting Up CPU Storage

- Create initial state first on CPU
- Must copy state to GPU
- Use same data structure for arrays
- Choose one efficient for device architecture

Allocating and filling host arrays

```
1   cl_float4* h_pos = ( cl_float4* ) malloc( num_particles * sizeof(cl_floa
2   cl_float4* h_vel = ( cl_float4* ) malloc( num_particles * sizeof(cl_floa
3   for( int ii = 0; ii < num_particles; ++ii )
4   {
5           //Storing the charge of particle in pos
6           h_pos[ii].w = (float)( ii % 2 ) * ( -1 ) * ( ii % 3 + 1 );
7
8           //Let's start all our charges in a ring around the source
9           h_pos[ii].x = 2 * cos( 2 * M_PI * (float) ii / num_particles );
10          h_pos[ii].y = 2 * sin( 2 * M_PI * (float) ii / num_particles );
11
12          //And keep them stationary
13          h_vel[ii].x = 0.f;
14          h_vel[ii].y = 0.f;
15  }
```

Creating GPU Storage

Creating GPU Storage

- Create typed buffers to store data

Creating GPU Storage

- Create typed buffers to store data
- Copy data from host to device

Creating GPU Storage

- Create typed buffers to store data
- Copy data from host to device
- Can do both with `clCreateBuffer` call

Creating GPU Storage

- Create typed buffers to store data
- Copy data from host to device
- Can do both with `clCreateBuffer` call
- Want to pick appropriate data structure

Creating GPU Storage

- Create typed buffers to store data
- Copy data from host to device
- Can do both with `clCreateBuffer` call
- Want to pick appropriate data structure
- Vectors better than scalars on AMD

Creating GPU Storage

- Create typed buffers to store data
- Copy data from host to device
- Can do both with `clCreateBuffer` call
- Want to pick appropriate data structure
- Vectors better than scalars on AMD
- Store position and velocites in `float4`

GPU Arrays

```
1           cl_mem pos = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOS
2                                       num_particles * sizeof(cl_float4), h_pos, NULL)
3           cl_mem vel = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOS
4                                       num_particles * sizeof(cl_float4), h_vel, NULL)
```

Calling the Kernel

Calling the Kernel

- Set kernel arguments

Calling the Kernel

- Set kernel arguments
- Push kernel launch into task queue

Calling the Kernel

- Set kernel arguments
- Push kernel launch into task queue
- Launch kernel once for each iteration

Setting Arguments and Launching

```
1         clSetKernelArg(particles, 0, num_particles * sizeof( cl_float4 ), &pos);
2         clSetKernelArg(particles, 1, num_particles * sizeof( cl_float4 ), &vel);
3         clSetKernelArg(particles, 2, sizeof( float ), &strength);
4         clSetKernelArg(particles, 3, sizeof( float ), &delta_t);
5         for( int jj = 0; jj < num_iterations; ++jj )
6         {
7                 clEnqueueNDRangeKernel(queue, particles, 1, 0, &g_work_size,
8                                                                 NULL, 0, NULL,
9         }
```

Writing the Kernel

Writing the Kernel

- Designate function as kernel using `__kernel`

Writing the Kernel

- Designate function as kernel using `__kernel`
- Must designate where arguments reside

Writing the Kernel

- Designate function as kernel using `__kernel`
- Must designate where arguments reside
- Particles don't interact $\Rightarrow$ use one array

Writing the Kernel

- Designate function as kernel using `__kernel`
- Must designate where arguments reside
- Particles don't interact $\Rightarrow$ use one array
- One-to-one map between threads and elements

Writing the Kernel

- Designate function as kernel using `__kernel`
- Must designate where arguments reside
- Particles don't interact $\Rightarrow$ use one array
- One-to-one map between threads and elements
- Need to find thread number

Beginning Kernel

```
1  "__kernel void update_state(__global float4 *pos,                    "
2  "                           __global float4 *vel,                    "
3  "                           float strength,                          "
4  "                           float delta_t,                           "
5  "                           int num_particles)                       "
6  "{                                                                   "
7  "    // Figure out which particle we are handling                    "
8  "                                                                    "
9  "    uint current = get_global_id(0);                                "
```

Updating the State

Updating the State

- Need to pick integration scheme

Updating the State

- Need to pick integration scheme
- Euler is easy, but unstable

$$x(t + \Delta t) = x(t) + \Delta t \cdot v_x(t)$$
$$v_x(t + \Delta t) = v(t) + \Delta t \cdot a_x(t)$$

Updating the State

- Need to pick integration scheme
- Euler is easy, but unstable

$$x(t + \Delta t) = x(t) + \Delta t \cdot v_x(t)$$
$$v_x(t + \Delta t) = v(t) + \Delta t \cdot a_x(t)$$

- Find particle's position in polar coordinates

Updating the State

- Need to pick integration scheme
- Euler is easy, but unstable

$$x(t + \Delta t) = x(t) + \Delta t \cdot v_x(t)$$
$$v_x(t + \Delta t) = v(t) + \Delta t \cdot a_x(t)$$

- Find particle's position in polar coordinates
- Update position, then velocity

Updating the State

- Need to pick integration scheme
- Euler is easy, but unstable

$$x(t + \Delta t) = x(t) + \Delta t \cdot v_x(t)$$
$$v_x(t + \Delta t) = v(t) + \Delta t \cdot a_x(t)$$

- Find particle's position in polar coordinates
- Update position, then velocity
- Avoid array overruns

Kernel Body

```
1  "       if( current < num_particles )                                                    "
2  "       {                                                                                "
3  "           // Calculate new acceleration                                                "
4  "           float4 accel;                                                                "
5  "           accel.w         = pos[current].w * strength /                                "
6  "                                                   ( pos[current].x * pos[current].x +pc
7  "           float theta = atan2( pos[current].y / pos[current].x );                      "
8  "           accel.x         = accel.w * cos( theta ) * delta_t;                          "
9  "           accel.y         = accel.w * sin( theta ) * delta_t;                          "
10 "                                                                                        "
11 "           // Find new positions and velocities                                         "
12 "           pos[current].x += delta_t * vel[current].x;                                  "
13 "           pos[current].y += delta_t * vel[current].y;                                  "
14 "           vel[current].x += delta_t * accel.x;                                         "
15 "           vel[current].y += delta_t * accel.y;                                         "
16 "                                                                                        "
17 "       }                                                                                "
```

tableofcontents[currentsection]

Moving and Compiling

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`
- `ssh` in to this machine

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`
- `ssh` in to this machine
- Developer drivers and compiler already installed

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`
- `ssh` in to this machine
- Developer drivers and compiler already installed
- Two steps necessary:

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`
- `ssh` in to this machine
- Developer drivers and compiler already installed
- Two steps necessary:
  - Compile code with g++

Moving and Compiling

- `scp` code to `high-fructose-corn-syrup.csclub.uwaterloo.ca:`
- `ssh` in to this machine
- Developer drivers and compiler already installed
- Two steps necessary:
  - Compile code with g++
  - Link against OpenCL library

Compilation Steps

```
g++ -o mycode.o -DATI_OS_LINUX -c mycode.cl
-I$ATISTREAMSDKROOT/include
g++ -o mycode mycode.o -lOpenCL
-L$ATISTREAMSDKROOT/lib/x86_64
```

Testing

Testing

- Kernels compile JIT $\Rightarrow$ pass options then

Testing

- Kernels compile JIT $\Rightarrow$ pass options then
- Can use gdb to test program

Testing

- Kernels compile JIT $\Rightarrow$ pass options then
- Can use gdb to test program
- Can also set breakpoints in kernel

Testing

- Kernels compile JIT $\Rightarrow$ pass options then
- Can use gdb to test program
- Can also set breakpoints in kernel
- Let's see if program works

Using gdb

Before launching gdb, use:

```
AMD_OCL_BUILD_OPTIONS="-g -O0"
```

Then use:

```
gdb mycode.out
```

Use `r` to run the code

Improvements

Improvements

- Incorporate OpenGL - graph particle positions

Improvements

- Incorporate OpenGL - graph particle positions
- More accurate simulation - make particles interact

Improvements

- Incorporate OpenGL - graph particle positions
- More accurate simulation - make particles interact
- Use local memory to speed up kernel

Improvements

- Incorporate OpenGL - graph particle positions
- More accurate simulation - make particles interact
- Use local memory to speed up kernel
- Do time iteration within kernel

Improvements

- Incorporate OpenGL - graph particle positions
- More accurate simulation - make particles interact
- Use local memory to speed up kernel
- Do time iteration within kernel
- Use AMD Profiler to analyze code

Good Practices

Good Practices

- Keep work-items within wavefront instruction coherent

Good Practices

- Keep work-items within wavefront instruction coherent
- Use local and register memory

Good Practices

- Keep work-items within wavefront instruction coherent
- Use local and register memory
- Use appropriate data structure for architecture

Good Practices

- Keep work-items within wavefront instruction coherent
- Use local and register memory
- Use appropriate data structure for architecture
- Minimize control flow instructions within kernel

Learning More

Learning More

- Kronos group's OpenCL spec: http://www.khronos.org/opencl/

Learning More

- Kronos group's OpenCL spec: http://www.khronos.org/opencl/
- AMD's OpenCL tutorials and documentation: http://developer.amd.com/

Learning More

- Kronos group's OpenCL spec: http://www.khronos.org/opencl/
- AMD's OpenCL tutorials and documentation: http://developer.amd.com/
- nVidia's OpenCL sample code: http://developer.nvidia.com/opencl

Learning More

- Kronos group's OpenCL spec: http://www.khronos.org/opencl/
- AMD's OpenCL tutorials and documentation: http://developer.amd.com/
- nVidia's OpenCL sample code: http://developer.nvidia.com/opencl
- *Heterogenous Computing with OpenCL* - CSC has copies

Questions?

OpenCL contest

OpenCL contest

- Two categories:

OpenCL contest

- Two categories:
  - Open submission - make something awesome!

OpenCL contest

- Two categories:
  - Open submission - make something awesome!
  - Problem - ...

OpenCL contest

- Two categories:
  - Open submission - make something awesome!
  - Problem - ...
- Contest code party - March 02 2012

OpenCL contest

- Two categories:
  - Open submission - make something awesome!
  - Problem - ...
- Contest code party - March 02 2012
- Win a laptop or graphics card!