OpenCL BOF

SIGGRAPH
July 2010
Welcome to the OpenCL BOF!

- **Neil Trevett, NVIDIA, OpenCL Chair**
  - Overview of Khronos and OpenCL Ecosystem

- **Affie Munshi, OpenCL Spec Editor**
  - OpenCL Overview and OpenCL 1.1 Highlights

- **Ben Gaster, AMD and Yaki Tebeka, Graphic Remedy**
  - Rendering the Breeze

- **Ofer Rosenberg, Intel**
  - Optimizing OpenCL on CPUs

- **Cliff Woolley, NVIDIA**
  - OpenCL Optical Flow
We Have Prizes!

- gDEBugger CL one-year subscription license!
- Cypress-based ATI Radeon™ HD 5870!
- Fermi-based NVIDIA Quadro™ FX 5000 with 2.5GB GDDR5!

- We will ask trivia questions to determine winners – so pay attention! 😊
Over 100 companies creating visual computing standards

Board of Promoters

© Copyright Khronos Group, 2010 - Page 4
Processor Parallelism

CPUs
Multiple cores driving performance increases

GPUs
Increasingly general purpose data-parallel computing

Emerging Intersection

Multi-processor programming – e.g. OpenMP

Heterogeneous Computing
OpenCL

Graphics APIs and Shading Languages

OpenCL is a programming framework for heterogeneous compute resources
OpenCL Timeline

- **Six months from proposal to released OpenCL 1.0 specification**
  - Due to a strong initial proposal and a shared commercial incentive

- **Multiple conformant implementations shipping**
  - For CPUs and GPUs on multiple OS

- **18 month cadence between OpenCL 1.0 and OpenCL 1.1**
  - Backwards compatibility protect software investment

---

**Apple proposes OpenCL working group and contributes draft specification to Khronos**

**Jun08**  
Khronos publicly releases OpenCL 1.0 as royalty-free specification

**Dec08**  
Kronos releases OpenCL 1.0 conformance tests to ensure high-quality implementations

**May09**  
Multiple conformant implementations ship across diverse OS and platforms

**2H09**  
OpenCL 1.1 Specification released and first implementations ship

**Jun10**
OpenCL Working Group

- Diverse industry participation – many industry experts
  - Processor vendors, system OEMs, middleware vendors, application developers
- Apple made initial proposal and is very active in the working group
  - Serving as specification editor
Visual Computing Ecosystem

Mobile Visual Computing
Compute, graphics and AV APIs interoperate through EGL

Desktop Visual Computing
OpenGL and OpenCL have direct interoperability

OpenCL Specification
includes embedded profile to enable sooner deployment on mobile devices
OpenCL Overview
It’s a Heterogeneous World

• A modern platform Includes:
  - One or more CPUs
  - One or more GPUs
  - Accelerators such as DSPs

GMCH = graphics memory control hub
ICH = Input/output control hub
Design Goals of OpenCL

• Use all computational resources in the system
  - CPUs, GPUs and other processors as peers
  - Data- and task- parallel computational model

• Efficient parallel programming model
  - Based on C99
  - Abstract the specifics of underlying hardware

• Specify accuracy of floating-point computations

• Desktop and Handheld Profiles
OpenCL Platform Model

- **One Host + one or more Compute Devices**
  - Each Compute Device is composed of one or more Compute Units
  - Each Compute Unit is further divided into one or more Processing Elements
OpenCL Execution Model

- An OpenCL application runs on a host which submits work to the compute devices
  - **Work item**: the basic unit of work on an OpenCL device
  - **Kernel**: the code for a work item. Basically a C function
  - **Program**: Collection of kernels and other functions (Analogous to a dynamic library)
  - **Context**: The environment within which work-items executes ... includes devices and their memories and command queues

- Applications queue kernel execution instances
  - Queued in-order ... one queue to a device
  - Executed in-order or out-of-order
The BIG Idea behind OpenCL

• **OpenCL execution model** ...
  - Define N-dimensional computation domain
  - Execute a kernel at each point in computation domain

**Traditional loops**

```c
void trad_mul(int n,
    const float *a,
    const float *b,
    float *c)
{
    int i;
    for (i=0; i<n; i++)
        c[i] = a[i] * b[i];
}
```

**Data Parallel OpenCL**

```c
kernel void dp_mul(global const float *a,
    global const float *b,
    global float *c)
{
    int id = get_global_id(0);
    c[id] = a[id] * b[id];
} // execute over "n" work-items
```
An N-dimension domain of work-items

• Kernels executed across a global domain of work-items
• Work-items grouped into local workgroups
• Define the “best” N-dimensioned index space for your algorithm
  - Global Dimensions: 1024 x 1024 (whole problem space)
  - Local Dimensions: 128 x 128 (work group ... executes together)

Synchronization between work-items possible only within workgroups: barriers and memory fences

Cannot synchronize outside of a workgroup
OpenCL Memory Model

• **Private Memory**
  – Per work-item

• **Local Memory**
  – Shared within a workgroup

• **Global/Constant Memory**
  – Visible to all workgroups

• **Host Memory**
  – On the CPU

---

Memory management is Explicit
You must move data from host -> global -> local … and back
Using OpenCL

```
__kernel void dp_mul(global const float *a,
                     global const float *b,
                     global float *c)
{
    int id = get_global_id(0);
    c[id] = a[id] * b[id];
}
```

CPU

GPU

Context

Programs

Kernels

Memory Objects

Command Queues

Compile code

Create data & arguments

Send to execution

In Order Queue

Out of Order Queue

Compute Device

© Copyright Khronos Group, 2010 - Page 17
Building Program Objects

• The program object encapsulates:
  - A context
  - The program source/binary
  - List of target devices and build options

• The Build process … to create a program object
  - `clCreateProgramWithSource`
  - `clCreateProgramWithBinary`

Kernel Code

```c
kernel void horizontal_reflect(
    read_only image2d_t src,
    write_only image2d_t dst)
{
    int x = get_global_id(0);  // x-coord
    int y = get_global_id(1);  // y-coord
    int width = get_image_width(src);
    float4 src_val = read_imagef(src, sampler,
        (int2)(width-1-x, y));
    write_imagef(dst, (int2)(x, y), src_val);
}
```

Compile for GPU

Compile for CPU

Program
Synchronization: Queues & Events

- Each individual queue can execute in-order or out-of-order
  - For in-order queue, all commands execute in order
- You must explicitly synchronize between queues
  - Multiple devices each have their own queue
  - Multiple queues per device
  - Use events to synchronize
- Events
  - Commands return events and obey waitlists
  - `clEnqueue*(. . . , num_events_in_waitlist, *event_waitlist, *event);`
Synchronization: Queues & Events

- Events can be used to synchronize kernel executions between queues
- Example: 2 queues with 2 devices
Programming Kernels: OpenCL C

• Derived from ISO C99
  - But without some C99 features such as standard C99 headers, function pointers, recursion, variable length arrays, and bit fields

• Language Features Added
  - Work-items and workgroups
  - Vector types
  - Synchronization
  - Address space qualifiers

• Also includes a large set of built-in functions
  - Image manipulation
  - Work-item manipulation,
  - Math functions, etc.
Programming Kernels: What is a kernel

- A data-parallel function executed by each work-item

```c
kernel void square(global float* input, global float* output)
{
    int i = get_global_id(0);
    output[i] = input[i] * input[i];
}
```

<table>
<thead>
<tr>
<th>Input</th>
<th>6 1 1 0 9 2 4 1 1 9 7 6 8 2 2 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>36 1 1 0 81 4 16 1 1 81 49 36 64 4 4 25</td>
</tr>
</tbody>
</table>

get_global_id(0) = 7
Programming Kernels: WorkItems & Groups

get_work_dim = 1

get_global_size = 16

get_num_groups = 2

get_group_id = 0

get_local_size = 8

get_local_id = 3

get_global_id = 11
Programming Kernels: Data Types

- **Scalar data types**
  - char, uchar, short, ushort, int, uint, long, ulong, float
  - bool, intptr_t, ptrdiff_t, size_t, uintptr_t, void, half (storage)

- **Image types**
  - image2d_t, image3d_t, sampler_t

- **Vector data types**
  - Vector lengths 2, 4, 8, & 16 (char2, ushort4, int8, float16, double2, ...)
  - Endian safe
  - Aligned at vector length
  - Vector operations

Double is an optional type in OpenCL 1.0
Programming Kernels: Vector Operations

• Vector Literal

```
int4 vi0 = (int4) -7;
int4 vi1 = (int4)(0, 1, 2, 3);
```

• Vector Components

```
vi0.lo = vi1.hi;
int8 v8 = (int8)(vi0, vi1.s01, vi1.odd);
```

• Vector Operations

```
vi0 += vi1;
```
Introducing OpenCL 1.1

New version of the OpenCL specification adds significant new functionality:

• API and Language Features
• Developer feedback, Ease of use and Performance
• Improved OpenGL Interoperability
OpenCL 1.1 – Thread-safety & Buffers

• Thread-safety
  - All API calls except `clSetKernelArg` are now thread-safe

• Sub-Buffer Objects
  - Easy and efficient way to distribute regions of a buffer across multiple devices
  - Modifications to sub-buffer objects reflected in appropriate regions of parent buffer object.

• Reading, writing & copying rectangular regions in a buffer object
  - Specify the following
    - Region type – 2D or 3D
    - Row-pitch for a 2D & 3D region and Slice-pitch for a 3D region
  - `clEnqueue{Read|Write|Copy}BufferRect`
OpenCL 1.1 – Events

• User Events
  - All `clEnqueue*` commands take a list of events to wait on
  - In OpenCL 1.0, events can only refer to OpenCL commands
  - User events allow developers to enqueue commands that wait on an external event

• Event Callbacks
  - Allow applications to enqueue CL commands based on event state changes in a non-blocking manner
  - `clSetEventCallback` to register a user callback function
    - Called when command identified by event has completed
    - Recommend **not calling** expensive system APIs, OpenCL APIs that create objects or enqueue blocking commands in the callback function.
OpenCL 1.1 – Memory Object Callbacks

- Memory Object Destructor Callback
  - For **cl_mem** objects created with **CL_MEM_USE_HOST_PTR** need a way to determine when it is safe to free or reuse the **host_ptr**
  - Lazy deallocation of **cl_mem** objects make this a little difficult
  - **clSetMemObjectDestructorCallback**
    - Registers a destructor callback function
    - Called when the memory object is ready to be deleted
  - Recommend **not calling** expensive system APIs, OpenCL APIs that create objects or enqueue blocking commands in the callback function.
OpenCL 1.1 – Queries

• Kernel Queries
  - `CL_KERNEL_PREFERRED_WORKGROUP_SIZE_MULTIPLE`
    - A performance hint

• Device Queries
  - `CL_DEVICE_LOCAL_MEM_SIZE`
    - Increased from 16 KB to 32 KB
  - `CL_DEVICE_MAX_PARAMETER_SIZE`
    - Increased from 256 to 1024 bytes
  - `CL_DEVICE_OPENCL_C_VERSION`
    - Version of OpenCL C supported by device.
  - `CL_DEVICE_HOST_UNIFIED_MEMORY`
    - Whether device & host have a unified memory subsystem
OpenCL 1.1 – Additional API Features

- **global_work_offset**
  - Argument to `clEnqueueNDRangeKernel`
  - No longer required to be a NULL value
  - Enqueue kernels that operate on different regions of the N-D range

- **C++ API bindings**
  - A wrapper API
  - Built on top of the OpenCL 1.1 API specification (not a replacement)
Implicit Conversions

- OpenCL 1.0 requires widening for arithmetic operators

```c
float4 a, b;
float c;

b = a + c; // c is widened to a float4 first
// and then the + is performed.
```

- OpenCL 1.1 extends this feature to all operators
  - relational, equality, bitwise, logical and ternary
OpenCL 1.1 – Language Features

• 3-component vector data types
  - Useful data type for a number of applications such as game physics
  - Aligned to the corresponding 4-component data type
  - vload3 and vstore3 can be used to view a buffer of scalar elements as a packed buffer of 3-component vectors

• cl_khr_byte_addressable is a core feature
  - Writes to a pointer or array of type char, char2, uchar, uchar2, short, ushort and half are now supported

• 32-bit atomic operations to local and global memory is a core feature
OpenCL 1.1 – Built-in Functions

- **get_global_offset**
  - Global offset values specified to `clEnqueueNDRangeKernel`

- **clamp for integer data types**
  - Only floating-point types were supported in OpenCL 1.0

- **async_work_group_strided_copy**
  - Gather from global to local memory
  - Scatter from local to global memory

- **shuffle**
  - Construct a runtime permutation of elements from 1 or 2 vectors and a mask

```c
uint4 mask = (uint4)(3, 2, 1, 0);
float4 a;
float4 r = shuffle(a, mask)
// r.s0123 = a.wzyx
```
OpenCL 1.1 – Images

- **Addressing mode** – `CL_ADDRESS_MIRRORED_REPEAT`
  - Flip the image coordinate at every integer junction
  - Can only be used with normalized coordinates i.e. `CL_NORMALIZED_COORDS_TRUE` must be set in the sampler

- **Optional image formats** – `CL_Rx`, `CL_RGx` and `CL_RGBx`
  - Similar to `CL_R`, `CL_RG` and `CL_RGB` except `alpha = 0` at edges
  - For image processing, `alpha` must always be 0 at edges.
OpenCL 1.1 – OpenCL / OpenGL Sharing

• Improve performance of CL / GL interoperability
  - In OpenCL 1.0, portable CL / GL sharing requires
    - a `glFinish` before `clEnqueueAcquireGLObjects`
    - a `clFinish` after `clEnqueueReleaseGLObjects`
    - `glFinish` and `clFinish` are heavyweight APIs

• New Extensions
  - `cl_khr_gl_event`
    - Create a CL event from a GL sync object
  - `GL_ARB_cl_event`
    - Create a GL sync object from a CL event

• Use CL events and GL sync objects for a faster & finer-grained synchronization
OpenCL 1.1 – OpenCL / OpenGL Sharing

// queue pGL commands
gl_sync = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);


gl_event = clCreateEventFromGLSyncKHR(context, gl_sync, NULL);
clEnqueueAcquireGLObjects(queue, num_objects, mem_objects,
                          1, &gl_event, NULL);

// queue CL commands

clEnqueueReleaseGLObjects(queue, num_objects, mem_objects,
                          0, NULL, &release_event);

cl_sync = glCreateSyncFromCLEventARB(context, release_event, 0);
glWaitSync(cl_sync, 0, TIMEOUT_IGNORERED);

// queue GL commands
Summary

• **OpenCL is a portable and high performance framework**
  - Ideal for computationally intensive algorithms
  - Efficient parallel programming language
  - Access to all compute resources
  - Defines hardware and numerical precision requirements

• **OpenCL 1.1 includes new API and language features**
  - Requested by developers
  - For better ease of use and performance

• **Open Standard for heterogeneous parallel computing**
Trivia Questions!

- **Neil Trevett, OpenCL Chair**
  - Overview of Khronos and OpenCL Ecosystem

- **Affie Munshi, OpenCL Spec Editor**
  - OpenCL Overview and OpenCL 1.1 Highlights

- **Ben Gaster, AMD and Yaki Tebeka, Graphic Remedy**
  - Rendering the Breeze

- **Ofer Rosenberg, Intel**
  - Optimizing OpenCL on CPUs

- **Cliff Woolley, NVIDIA**
  - OpenCL Optical Flow
OpenCL Books

- **The OpenCL Programming Book**
  - Available now
  - Search for OpenCL on Amazon

- **OpenCL Programming Guide - The “Red Book” of OpenCL**
  - Coming in May 2011
  - Rough cut available on Safaribooks
  - [http://my.safaribooksonline.com/9780132488006](http://my.safaribooksonline.com/9780132488006)
OzViz 2010

- 1-3 December 2010, Brisbane Queensland
- OpenCL Workshop
- Call for presentations now open!
Khronos OpenCL Resources

• OpenCL is 100% free for developers
  - Download drivers from your silicon vendor

• OpenCL Registry – OpenCL 1.0 and OpenCL 1.1
  - www.khronos.org/registry/cl/

• OpenCL 1.1 Reference Card
  - PDF version

• Online Man pages
  - http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

• OpenCL Developer Forums
  - Give us your feedback!
  - www.khronos.org/message_boards/