Introduction to OpenCL™ Programming
Agenda

- GPGPU Overview
- Introduction to OpenCL™
- Getting Started with OpenCL™
- OpenCL™ Programming in Detail
- The OpenCL™ C Language
- Application Optimization and Porting
GPGPU Overview
GPGPU Overview

• What is GPU Compute?
• Brief History of GPU Compute
• Heterogeneous Computing

Introduction to OpenCL™

Getting Started with OpenCL™

OpenCL™ Programming in Detail

The OpenCL™ C Language

Application Optimization and Porting
What is GPGPU?

• General Purpose computation on Graphics Processing Units

• High performance multi-core processors
  • excels at parallel computing

• Programmable coprocessors for other than just for graphics
Brief History of GPGPU

• November 2006
  • Birth of GPU compute with release of Close to Metal (CTM) API
  • Low level API to access GPU resources
  • New GPU accelerated applications
    • Folding@Home released with 20-30x speed increased
Brief History of GPGPU

- December 2007
  - ATI Stream SDK v1 released
Brief History of GPGPU

• June 2008
  • OpenCL™ working group formed under Khronos™
  • OpenCL™ 1.0 Spec released in Dec 2008
  • AMD announced adoption of OpenCL™ immediately

• December 2009
  • ATI Stream SDK v2 released
  • OpenCL™ 1.0 support
Heterogeneous Computing

- Using various types of computational units
  - CPU, GPU, DSP, etc...

- Modern applications interact with various systems (audio/video, network, etc...)
  - CPU scaling unable to keep up
  - Require specialized hardware to achieve performance
Heterogeneous Computing

• Ability to select most suitable hardware in heterogeneous system

Serial and Task Parallel Workloads

Data Parallel Workloads

Graphics Workloads

Software Applications
Introduction to OpenCL™
GPGPU Overview

Introduction to OpenCL™

- What is OpenCL™?
- Benefits of OpenCL™
- Anatomy of OpenCL™
- OpenCL™ Architecture
  - Platform Model
  - Execution Model
  - Memory Model

Getting Started with OpenCL™

OpenCL™ Programming in Detail

The OpenCL™ C Language

Application Optimization and Porting
What is OpenCL™?

- **Open Computing Language**
- Open and royalty free API
  - Enables GPU, DSP, co-processors to work in tandem with CPU
  - Released December 2008 by Khronos™ Group
Benefits of OpenCL™

• Acceleration in parallel processing

• Allows us to manage computational resources
  • View multi-core CPUs, GPUs, etc as computational units
  • Allocate different levels of memory

• Cross-vendor software portability
  • Separates low-level and high-level software
Anatomy of OpenCL™

- **Language Specification**
  - Based on ISO C99 with added extension and restrictions

- **Platform API**
  - Application routines to query system and setup OpenCL™ resources

- **Runtime API**
  - Manage kernels objects, memory objects, and executing kernels on OpenCL™ devices
OpenCL™ Architecture – Platform Model

- Host
- Compute Device
- Compute Unit
- Processing Element
- OpenCL™ Architecture – Platform Model
OpenCL™ Device Example

• ATI Radeon™ HD 5870 GPU
OpenCL™ Device Example

- ATI Radeon™ HD 5870 GPU

1 Compute Unit
Contains 16 Stream Cores

1 Stream Core = 5 Processing Elements
OpenCL™ Architecture – Execution Model

• **Kernel:**
  ▪ Basic unit of executable code that runs on OpenCL™ devices
  ▪ Data-parallel or task-parallel

• **Host program:**
  ▪ Executes on the host system
  ▪ Sends kernels to execute on OpenCL™ devices using command queue
Kernels – Expressing Data-Parallelism

• Define N-dimensional computation domain
  ▪ N = 1, 2, or 3
  ▪ Each element in the domain is called a work-item
  ▪ N-D domain (global dimensions) defines the total work-items that execute in parallel
  ▪ Each work-item executes the same kernel

Process 1024x1024 image:
Global problem dimension: 1024x1024
1 kernel execution per pixel: 1,048,576 total executions
Kernels: Work-item and Work-group

- Work-items are grouped into **work-groups**
  - **Local dimensions** define the size of the workgroups
  - Execute together on same compute unit
  - Share local memory and synchronization

Synchronization between work-items possible only within work-groups
Cannot synchronize between workgroups
Kernels: Work-item and Work-group Example

dimension: 2
global size: 32x32=1024
num of groups: 16

local id: (4,2)
global id: (28,10)

workgroup id: (3,1)
local size: 8x8=64
## Kernels Example

<table>
<thead>
<tr>
<th><strong>Scalar</strong></th>
<th><strong>Data-Parallel</strong></th>
</tr>
</thead>
</table>
| void square(int n, const float *a, float *result)  
{  
    int i;  
    for (i=0; i<n; i++)  
        result[i] = a[i] * a[i];  
} | kernel dp_square (const float *a, float *result)  
{  
    int id = get_global_id(0);  
    result[id] = a[id] * a[id];  
} |

// dp_square executes over “n” work-items
Execution Model – Host Program

• Create “Context” to manage OpenCL™ resources
  ▪ **Devices** – OpenCL™ device to execute kernels
  ▪ **Program Objects**: source or binary that implements kernel functions
  ▪ **Kernels** – the specific function to execute on the OpenCL™ device
  ▪ **Memory Objects** – memory buffers common to the host and OpenCL™ devices
Execution Model – Command Queue

- Manage execution of kernels
- Accepts:
  - Kernel execution commands
  - Memory commands
  - Synchronization commands
- Queued in-order
- Execute in-order or out-of-order
Memory Model

- Host Memory
- Global/Constant Memory
- Local Memory
- Work-item
- Private Memory
- Workgroup
- Compute Device
- Host
Memory Model

- **Global** – read and write by all work-items and work-groups
- **Constant** – read-only by work-items; read and write by host
- **Local** – used for data sharing; read/write by work-items in same work-group
- **Private** – only accessible to one work-item

Memory management is explicit
Must move data from host to global to local and back
Getting Started with OpenCL™
GPGPU Overview

Getting Started with OpenCL™

- Software Development Environment
  - Requirements
  - Installation on Windows®
  - Installation on Linux®
- First OpenCL™ Program
- Compiling OpenCL™ Source

OpenCL™ Programming in Detail

The OpenCL™ C Language

Application Optimization and Porting
# Software Development Kit

## ATI Stream SDK v2
Download free at http://developer.amd.com/stream

<table>
<thead>
<tr>
<th>File Name</th>
<th>Launch Date</th>
<th>Bitness</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ati-stream-sdk-v2.01-lnx32.tgz (34.2MB)</td>
<td>03/29/2010</td>
<td>32-bit</td>
<td>ATI Stream SDK built for 32-bit Linux®</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-lnx64.tgz (59.2MB)</td>
<td>03/29/2010</td>
<td>64-bit</td>
<td>ATI Stream SDK built for 64-bit Linux®</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-rhel32.tgz (35.3MB)</td>
<td>02/10/2010</td>
<td>32-bit</td>
<td>ATI Stream SDK built for 32-bit Red Hat® Enterprise Linux®</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-rhel64.tgz (61.0MB)</td>
<td>02/10/2010</td>
<td>64-bit</td>
<td>ATI Stream SDK built for 64-bit Red Hat® Enterprise Linux®</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-vista-win7-32.exe (49.2MB)</td>
<td>03/29/2010</td>
<td>32-bit</td>
<td>ATI Stream SDK built for 32-bit Microsoft® Windows Vista® and Microsoft® Windows® 7</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-vista-win7-64.exe (91.9MB)</td>
<td>03/29/2010</td>
<td>64-bit</td>
<td>ATI Stream SDK built for 64-bit Microsoft® Windows Vista® and Microsoft® Windows® 7</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-xp32.exe (49.1MB)</td>
<td>03/29/2010</td>
<td>32-bit</td>
<td>ATI Stream SDK built for 32-bit Microsoft® Windows® XP</td>
</tr>
<tr>
<td>ati-stream-sdk-v2.01-xp64.exe (91.7MB)</td>
<td>03/29/2010</td>
<td>64-bit</td>
<td>ATI Stream SDK built for 64-bit Microsoft® Windows® XP</td>
</tr>
</tbody>
</table>
**SDK Requirements**

**Supported Operating Systems:**

| Windows®: | • Windows® XP SP3 (32-bit), SP2 (64-bit)  
|           | • Windows® Vista® SP1 (32/64-bit)  
|           | • Windows® 7 (32/64-bit)  |
| Linux®:   | • openSUSE™ 11.1 (32/64-bit)  
|           | • Ubuntu® 9.10 (32/64-bit)  
|           | • Red Hat® Enterprise Linux® 5.3 (32/64-bit) |

**Supported Compilers:**

| Windows®: | • Microsoft® Visual Studio® 2008 Professional Ed. |
| Linux®:   | • GNU Compiler Collection (GCC) 4.3 or later  
|           | • Intel® C Compiler (ICC) 11.x |
## SDK Requirements

### Supported GPUs:

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATI Radeon™ HD</td>
<td>5970, 5870, 5850, 5770, 5670, 5570, 5450, 4890, 4870 X2, 4870, 4850, 4830, 4770, 4670, 4650, 4550, 4350</td>
</tr>
<tr>
<td>ATI FirePro™</td>
<td>V8800, V8750, V8700, V7800, V7750, V5800, V5700, V4800, V3800, V3750</td>
</tr>
<tr>
<td>AMD FireStream™</td>
<td>9270, 9250</td>
</tr>
<tr>
<td>ATI Mobility Radeon™ HD</td>
<td>5870, 5850, 5830, 5770, 5730, 5650, 5470, 5450, 5430, 4870, 4860, 4850, 4830, 4670, 4650, 4500 series, 4300 series</td>
</tr>
<tr>
<td>ATI Mobility FirePro™</td>
<td>M7820, M7740, M5800</td>
</tr>
<tr>
<td>ATI Radeon™ Embedded</td>
<td>E4690 Discrete GPU</td>
</tr>
</tbody>
</table>
SDK Requirements

Supported GPU Drivers:

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>Driver Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATI Radeon™ HD</td>
<td>ATI Catalyst™ 10.4</td>
</tr>
<tr>
<td>ATI FirePro™</td>
<td>ATI FirePro™ Unified Driver 8.723</td>
</tr>
<tr>
<td>AMD FireStream™</td>
<td>ATI Catalyst™ 10.4</td>
</tr>
<tr>
<td>ATI Mobility Radeon™ HD</td>
<td>ATI Catalyst™ Mobility 10.4</td>
</tr>
<tr>
<td>ATI Mobility FirePro™</td>
<td>Contact the laptop manufacturer for the appropriate driver</td>
</tr>
<tr>
<td>ATI Radeon™ Embedded</td>
<td>Contact the laptop manufacturer for the appropriate driver</td>
</tr>
</tbody>
</table>

Supported Processors:
- Any X86 CPU with SSE 3.x or later
Installing SDK on Windows®
Installing SDK on Windows®

The display driver installed on this system does not have OpenCL™ support for AMD GPUs. If you continue with the current installation, you will still be able to run OpenCL™ on compatible x86 CPUs. Do you wish to continue?

[Yes] [No]
Installing SDK on Linux®

1. Untar the SDK to a location of your choice:
   - `tar -zxvf ati-stream-sdk-v2.1-lnx32.tgz`

2. Add `ATISTREAMSDKROOT` to environment variables:
   - `export ATISTREAMSDKROOT=<your_install_location>`

3. If the sample code was installed, add `ATISTREAMSDKSAMPLESROOT` to your environment variables:
   - `export ATISTREAMSDKSAMPLESROOT=<your_install_location>`
Installing SDK on Linux®

4. Add the appropriate path to the `LD_LIBRARY_PATH`:

On 32-bit systems:

- `export
  LD_LIBRARY_PATH=$ATISTREAMSDKROOT/lib/x86:$LD_LIBRARY_PATH`

On 64-bit systems:

- `export
  LD_LIBRARY_PATH=$ATISTREAMSDKROOT/lib/x86_64:$LD_LIBRARY_PATH`
Installing SDK on Linux®

5. Register the OpenCL™ ICD to allow applications to run by:
   - `sudo -s`
   - `mkdir -p /etc/OpenCL/vendors`

On all systems:
   - `echo libatiocl32.so > /etc/OpenCL/vendors/atiocl32.icd`

On 64-bit systems also perform:
   - `echo libatiocl64.so > /etc/OpenCL/vendors/atiocl64.icd`
First OpenCL™ Application
see “hello_world.c”
Compiling on Linux®

• To compile on Linux®:
  ▪ gcc -o hello_world -I$ATISTREAMSDKROOT/include -L$ATISTREAMSDKROOT/lib/x86 hello_world.c -lOpenCL

• To execute the program:
  ▪ Ensure LD_LIBRARY_PATH environment variable is set to find libOpenCL.so, then:
  ▪ ./hello_world
Compiling on Windows® Visual Studio®

• Set include path:
Compiling on Windows® Visual Studio®

• Set library path:
Compiling on Windows® Visual Studio®

• Set additional library to link:
OpenCL™ Programming in Detail
GPGPU Overview

OpenCL™ Programming in Detail
- OpenCL™ Application Execution
- Resource Setup
- Kernel Programming and Compiling
- Program Execution
- Memory Objects
- Synchronization

The OpenCL™ C Language

Application Optimization and Porting
OpenCL™ Program Flow

Context

Programs

Kernels

Memory Objects

Command Queue

__kernel void
sqr(__global float *input,
__global float *output)
{
size_t id = get_global_id(0);
output[id] = input[id] * input[id];
}

__kernel void
sqr(__global float *input,
__global float *output)
{
sqr = arg[0];
arg[0] value
arg[1] value
images

buffers

In Order Queue

Out of Order Queue

Compile
Create data & arguments
Send to execution

In
Order
Queue

Out of
Order
Queue
Query for Platform IDs

• First Step in any OpenCL™ application

```
cl_platform_id platforms;
cl_uint num_platforms;

cl_int err = clGetPlatformIDs(
    1, // the number of entries that can added to platforms
    &platforms, // list of OpenCL found
    &num_platforms // the number of OpenCL platforms available
);
```

Returns:

- **CL_INVALID_VALUE** — Platforms and `num_platforms` is NULL or the number of entries is 0.
- **CL_SUCCESS** — The function executed successfully.
Query for Platform Information

• Get specific info. about the OpenCL™ Platform
• Use
  ▪ clGetPlatformInfo()
    – platform_profile
    – platform_version
    – platform_name
    – platform_vendor
    – platform_extensions
Query for OpenCL™ Device

• Search for OpenCL™ compute devices in system

```c
cl_device_id device_id;
cl_uint num_of_devices;
err = clGetDeviceIDs(
    platform_id,             // the platform_id retrieved from clGetPlatformIDs
    CL_DEVICE_TYPE_GPU,      // the device type to search for
    1,                       // the number of ids to add to device_id list
    &device_id,              // the list of device ids
    &num_of_devices)         // the number of compute devices found
);
```
Query for OpenCL™ Device

Supported device types:

- CL_DEVICE_TYPE_CPU
- CLDEVICE_TYPE_GPU
- CL_DEVICE_TYPE_ACCELERATOR
- CL_DEVICE_TYPE_DEFAULT
- CL_DEVICE_TYPE_ALL

clGetDeviceIDs() Returns:

- CL_SUCCESS — The function executed successfully.
- CL_INVALID_PLATFORM — Platform is not valid.
- CL_INVALID_DEVICE_TYPE — The device is not a valid value.
- CL_INVALID_VALUE — num_of_devices and devices are NULL.
- CL_DEVICE_NOT_FOUND — No matching OpenCL of device_type was found.
Query for Device Information

- Get specific info. about the OpenCL™ Device
  - Use
    - `clGetDeviceInfo()`
      - `device_type`
      - `max_compute_units`
      - `max_workgroup_size`
      - ...
Creating Context

- Manage command queues, program objects, kernel objects, memory object

```c
cl_context context;
// context properties list - must be terminated with 0
properties[0]= CL_CONTEXT_PLATFORM; // specifies the platform to use
properties[1]= (cl_context_properties) platform_id;
properties[2]= 0;

context = clCreateContext(
    properties,       // list of context properties
    1,                // num of devices in the device_id list
    &device_id,       // the device id list
    NULL,             // pointer to the error callback function (if required)
    NULL,             // the argument data to pass to the callback function
    &err);            // the return code
```
Creating Context

clGreateContext() Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_PLATFORM** — Property list is NULL or the platform value is not valid.
- **CL_INVALID_VALUE** — Either:
  - The property name in the properties list is not valid.
  - The number of devices is 0.
  - The device_id list is null.
  - The device in the device_id list is invalid or not associated with the platform.
- **CL_DEVICE_NOT_AVAILABLE** — The device in the device_id list is currently unavailable.
Creating Command Queue

- Allows kernel commands to be sent to compute devices

```c
cl_command_queue command_queue;
command_queue = clCreateCommandQueue(
    context,       // a valid context
    device_id,     // a valid device associated with the context
    0,             // properties for the queue (not used here)
    &err);         // the return code
```
Create Command Queue

Supported Command Queue Properties:

- `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE`
- `CL_QUEUE_PROFILING_ENABLE`

`clCreateCommandQueue()` Returns:

- `CL_SUCCESS` — The function executed successfully.
- `CL_INVALID_CONTEXT` — The context is not valid.
- `CL_INVALID_DEVICE` — Either the device is not valid or it is not associated with the context.
- `CL_INVALID_VALUE` — The properties list is not valid.
- `CL_INVALID_QUEUE_PROPERTIES` — The device does not support the properties specified in the properties list.
Program Object

- **Program** – collection of kernel and helper functions
- **Function** – written in OpenCL™ C Language
- **Kernel Function** – identified by `__kernel`
- **Program Object** - Encapsulates
  - Program sources or binary file
  - Latest successful-built program executable
  - List of devices for which exec is built
  - Build options and build log
- Created **online** or **offline**
Create Program Object Online

• Use \texttt{clCreateProgramWithSource()}

```c
const char *ProgramSource = 
"__kernel void hello(__global float *input, __global float *output)\n{
 size_t id = get_global_id(0);
 output[id] = input[id] \times input[id];
}"
;

cl_program program;
program = clCreateProgramWithSource(
    context, // a valid context
    1, // the number strings in the next parameter
    (const char **) &ProgramSource, // the array of strings
    NULL, // the length of each string or can be NULL terminated
    &err ); // the error return code
```
Create Program Object

clCreateProgramWithSource() Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_CONTEXT** — The context is not valid.
- **CL_INVALID_VALUE** — The string count is 0 (zero) or the string array contains a NULL string.

- Creating program object offline
  - Use **clGetProgramInfo()** to retrieve program binary for already created program object
  - Create program object from existing program binary with **clCreateProgramWithBinary()**
Building Program Executables

• Compile and link program object created from `clCreateProgramWithSource()` or `clCreateProgramWithBinary()`
• Create using `clBuildProgram()`

```c
err = clBuildProgram(
    program,   // a valid program object
    0,         // number of devices in the device list
    NULL,      // device list – NULL means for all devices
    NULL,      // a string of build options
    NULL,      // callback function when executable has been built
    NULL);     // data arguments for the callback function
```
Building Program Executables

Program Build Options – passing additional options to compiler such as preprocessor options or optimization options

Example:

```c
char * buildoptions = "-DFLAG1_ENABLED -cl-opt-disable "
```

clBuildProgram() Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_VALUE** — The number of devices is greater than zero, but the device list is empty.
- **CL_INVALID_VALUE** — The callback function is NULL, but the data argument list is not NULL.
- **CL_INVALID_DEVICE** — The device list does not match the devices associated in the program object.
- **CL_INVALID_BUILD_OPTIONS** — The build options string contains invalid options.
Retrieving Build Log

- Access build log with `clGetProgramBuildInfo()`

```c
if (clBuildProgram(program, 0, NULL, buildoptions, NULL, NULL) != CL_SUCCESS)
{
    printf("Error building program\n");
    char buffer[4096];
    size_t length;
    clGetProgramBuildInfo(
        program,             // valid program object
        device_id,           // valid device_id that executable was built
        CL_PROGRAM_BUILD_LOG, // indicate to retrieve build log
        sizeof(buffer),      // size of the buffer to write log to
        buffer,              // the actual buffer to write log to
        &length);            // the actual size in bytes of data copied to buffer

    printf("%s\n",buffer);
    exit(1);
}
```
Sample Build Log

```c
platform 1
platform name: ATI Stream
__kernel void square(const __global float *input0,
                     const __global float *input1,
                     __global float *out)
{
    const int Width = get_global_size(0);
    const size_t xid = get_global_id(0);
    const size_t yid = get_global_id(1);

    const int idx = id*Width + xid;
    out[idx] = input0[idx] + input1[idx];
}
```

```
-- /tmp/OCLZenMa8.cl(9): error: identifier "id" is undefined
   const int idx = id*Width + xid;
   ^

1 error detected in the compilation of "/tmp/OCLZenMa8.cl".
```
Creating Kernel Objects

- **Kernel function** identified with qualifier __kernel
- **Kernel object** encapsulates specified __kernel function along with the arguments
- Kernel object is what get sent to command queue for execution
- Create Kernel Object with `clCreateKernel()`

```c
cl_kernel kernel;
kernel = clCreateKernel(
    program, // a valid program object that has been successfully built
    "hello", // the name of the kernel declared with __kernel
    &err     // error return code
);
```
Creating Kernel Object

clCreateKernel() Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_PROGRAM** — The program is not a valid program object.
- **CL_INVALID_PROGRAM_EXECUTABLE** — The program does not contain a successfully built executable.
- **CL_INVALID_KERNEL_NAME** — The kernel name is not found in the program object.
- **CL_INVALID_VALUE** — The kernel name is NULL.
Setting Kernel Arguments

- Specify arguments that are associated with the __kernel function
- Use clSetKernelArg()

```c
err = clSetKernelArg(
    kernel, // valid kernel object
    0, // the specific argument index of a kernel
    sizeof(cl_mem), // the size of the argument data
    &input_data // a pointer of data used as the argument
);
```

- Example Kernel function declaration

```c
__kernel void hello(__global float *input, __global float *output)
```
Setting Kernel Arguments

- Must use **memory object** for arguments with __global or __constant
- Must use **image object** for arguments with image2d_t or image3d_t

`clSetKernelArg()` Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_PROGRAM** — The program is not a valid program object.
- **CL_INVALID_PROGRAM_EXECUTABLE** — The program does not contain a successfully built executable.
- **CL_INVALID_KERNEL_NAME** — The kernel name is not found in the program object.
- **CL_INVALID_VALUE** — The kernel name is NULL.
Executing Kernel

- Determine the problem space
- Determine global work size (total work-items)
- Determine local size (work-group size – work-items share memory in work-group)
- Use `clGetKernelWorkGroupInfo` to determine max work-group size
Enqueuing Kernel Commands

• Place kernel commands into command queue by using `clEnqueueNDRangeKernel()`

```c
err = clEnqueueNDRangeKernel(
    command_queue, // valid command queue
    kernel, // valid kernel object
    1, // the work problem dimensions
    NULL, // reserved for future revision - must be NULL
    &global, // work-items for each dimension
    NULL, // work-group size for each dimension
    0, // number of event in the event list
    NULL, // list of events that needs to complete before this executes
    NULL // event object to return on completion
);
```

```c
size_t local[2]={8,8}; // clGetKernelWorkGoupInfo()
```

```c
size_t global[2]={512,512};
```
Creating Kernel Object

Common `clEnqueueNDRangeKernel()` Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_PROGRAM_EXECUTABLE** — No executable has been built in the program object for the device associated with the command queue.
- **CL_INVALID_COMMAND_QUEUE** — The command queue is not valid.
- **CL_INVALID_KERNEL** — The kernel object is not valid.
- **CL_INVALID_CONTEXT** — The command queue and kernel are not associated with the same context.
- **CL_INVALID_KERNEL_ARGS** — Kernel arguments have not been set.
- **CL_INVALID_WORK_DIMENSION** — The dimension is not between 1 and 3.
- **CL_INVALID_GLOBAL_WORK_SIZE** — The global work size is NULL or exceeds the range supported by the compute device.
- **CL_INVALID_WORK_GROUP_SIZE** — The local work size is not evenly divisible with the global work size or the value specified exceeds the range supported by the compute device.
- **CL_INVALID_EVENT_WAIT_LIST** — The events list is empty (NULL) but the number of events arguments is greater than 0; or number of events is 0 but the event list is not NULL; or the events list contains invalid event objects.
Cleaning Up

• Release resources when execution is complete

```c
clReleaseMemObject(input);
clReleaseMemObject(output);
clReleaseProgram(program);
clReleaseKernel(kernel);
clReleaseCommandQueue(command_queue);
clReleaseContext(context);
```

• **clRelease** functions decrement reference count
• Object is deleted when reference count reaches zero
Memory Objects

• Allows packaging data and easy transfer to compute device memory

• Minimizes memory transfers from host and device

• Two types of memory objects:
  ▪ Buffer object
  ▪ Image object
Creating Buffer Object

```c
cl_mem input;
input = clCreateBuffer(
    context, // a valid context
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, // bit-field flag to specify
    // the usage of memory
    sizeof(float) * DATA_SIZE, // size in bytes of the buffer to allocated
    inputsrc, // pointer to buffer data to be copied from host
    &err // returned error code
);
```

### Memory usage flag

<table>
<thead>
<tr>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_MEM_READ_WRITE</td>
</tr>
<tr>
<td>CL_MEM_WRITE_ONLY</td>
</tr>
<tr>
<td>CL_MEM_READ_ONLY</td>
</tr>
<tr>
<td>CL_MEM_USE_HOST_PTR</td>
</tr>
<tr>
<td>CL_MEM_COPY_HOST_PTR</td>
</tr>
<tr>
<td>CL_MEM_ALLOC_HOST_PTR</td>
</tr>
</tbody>
</table>
Reading/Writing Buffer Objects

```c
err = clEnqueueReadBuffer(
    command_queue, // valid command queue
    output, // memory buffer to read from
    CL_TRUE, // indicate blocking read
    0,
    sizeof(float) * DATA_SIZE,
    results, // pointer to buffer in host mem to store read data
    0,
    NULL, // list of events that needs to complete before this executes
    NULL // event object to return on completion
);
```

```c
err = clEnqueueWriteBuffer(
    command_queue, // valid command queue
    input, // memory buffer to write to
    CL_TRUE, // indicate blocking write
    0,
    sizeof(float) * DATA_SIZE, // size in bytes of data being read
    host_ptr, // pointer to buffer in host mem to read data from
    0, // number of event in the event list
    NULL, // list of events that needs to complete before this executes
    NULL // event object to return on completion
);
```
Read/Writing Buffer Objects

clEnqueueReadBuffer and clEnqueueWriteBuffer () Returns:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_COMMAND_QUEUE** — The command queue is not valid
- **CL_INVALID_CONTEXT** — The command queue buffer object is not associated with the same context.
- **CL_INVALID_VALUE** — The region being read/write specified by the offset is out of bounds or the host pointer is NULL.
- **CL_INVALID_EVENT_WAIT_LIST** — Either:
  - The events list is empty (NULL), but the number of events argument is greater than 0
  - The number of events is 0, but the event list is not NULL
  - The events list contains invalid event objects.
Creating Image Object

• Built in support for representing image data

```c
image2d = clCreateImage2D( 
    context, // valid context
    flags, // bit-field flag to specify usage of memory
    image_format, // ptr to struct that specifies image format properties
    width, // width of the image in pixels
    height, // height of the image in pixels
    row_pitch, // scan line row pitch in bytes
    host_ptr, // pointer to image data to be copied from host
    &err // error return code
);```

• For 3D image object use `clCreateImage3D()`
  ▪ Specify depth, and slice pitch
Channel Order and Channel Data Type

• Built in support for representing image data

```
// Example:
cl_image_format image_format;
image_format.image_channel_data_type = CL_FLOAT;
image_format.image_channel_order = CL_RGBA;
```

• Channel Ordering:
  - CL_RGB, CL_ARGB, CL_RGBA, CL_R, etc...

• Channel Data Types:
  - CL_SNORM_INT8, CL_UNORM_INT16, CL_FLOAT, CL_UNSIGNED_INT32
Reading/Writing Image Objects

```c
err = clEnqueueReadImage (  
    command_queue, // valid command queue  
    image,         // valid image object to read from  
    blocking_read, // blocking flag, CL_TRUE or CL_FALSE  
    origin_offset, // (x,y,z) offset in pixels to read from z=0 for 2D image  
    region,       // (width,height,depth) in pixels to read from, depth=1 for 2D image  
    row_pitch,    // length of each row in bytes  
    slice_pitch,  // size of each 2D slice in the 3D image in bytes, 0 for 2D image  
    host_ptr,     // host memory pointer to store write image object data to  
    num_events,   // number of events in events list  
    event_list,   // list of events that needs to complete before this executes  
    &event        // event object to return on completion
);
```

```c
err = clEnqueueWriteImage (  
    command_queue, // valid command queue  
    image,         // valid image object to write to  
    blocking_read, // blocking flag, CL_TRUE or CL_FALSE  
    origin_offset, // (x,y,z) offset in pixels to write to z=0 for 2D image  
    region,       // (width,height,depth) in pixels to write to, depth=1 for 2D image  
    row_pitch,    // length of each row in bytes  
    slice_pitch,  // size of each 2D slice in the 3D image in bytes, 0 for 2D image  
    host_ptr,     // host memory pointer to store read data from  
    num_events,   // number of events in events list  
    event_list,   // list of events that needs to complete before this executes  
    &event        // event object to return on completion
);
```
Reading/Writing Image Objects

Common `clEnqueueReadImage()` and `clEnqueueWriteImage()` Return Codes:

- **CL_SUCCESS** — The function executed successfully.
- **CL_INVALID_COMMAND_QUEUE** — The command queue is not valid.
- **CL_INVALID_CONTEXT** — The command queue and image object are not associated with the same context.
- **CL_INVALID_MEM_OBJECT** — The image object is not valid
- **CL_INVALID_VALUE** — The region being read/write specified by the origin_offset and region is out of bounds or the host pointer is NULL.
- **CL_INVALID_VALUE** — The image object is 2D and origin_offset[2] (y component) is not set to 0, or region[2] (depth component) is not set to 1.
- **CL_INVALID_EVENT_WAIT_LIST** — Either: The events list is empty (NULL), but the number of events argument is greater than 0; or number of events is 0, but the event list is not NULL; or the events list contains invalid event objects.
Retaining and Releasing Memory Objects

• On creation reference counter set to “1”
• Counter used to track the number of references to the particular memory object
• Object retain reference by using:
  ▪ `clRetainMemObject()`
• Object decrement reference by using:
  ▪ `clReleaseMemObject()`
• Memory Object freed when reference counter = 0
Synchronization

• Kernel queued may not execute immediately
• Force kernel execution by using blocking call
  ▪ Set \textbf{CL\_TRUE} flag for clEnqueueRead*/Write*
• Use event to track execution status of kernels without blocking host application
• Queue can execute commands
  ▪ \textit{in-order}
  ▪ \textit{out-of-order}
• clEnqueue*(...,num_events, events_wait_list, event_return)
  ▪ Number of events to wait on
  ▪ A list of events to wait on
  ▪ Event to return
Synchronization Example 1: In-order Queue

Kernel 2 waits until Kernel 1 is finished

Command Queue

GPU

Kernel 1

Kernel 2
Two Command Queues Unsynchronized

Kernel 2 starts before the results from Kernel 1 is ready

[Diagram showing two command queues and two kernels, one for CPU and one for GPU, with arrows indicating the order of execution and time progression.]
Two Command Queues Synchronized

Kernel 2 waits for an event from Kernel 1 indicating it is finished

Command Queue

Enqueue Kernel 1

Enqueue Kernel 2

Kernel 2

Event

CPU

Kernel 1

GPU
Additional Event Functions

• Host block until all events in wait list are complete
  ▪ `clWaitForEvents(num_events, event_list)`

• OpenCL block until all events in wait list are complete
  ▪ `clEnqueueWaitForEvents(queue,num_events, event_list)`

• Tracking events by using event marker
  ▪ `clEnqueueMarker(queue, *event_return)`
Query Event Information

- Get status of command associated with event
  - `clEventInfo(event, param_name, param_size, ...)`

<table>
<thead>
<tr>
<th>CL_EVENT_COMMAND_QUEUE</th>
<th>Command queue associated with event</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_EVENT_COMMAND_TYPE</td>
<td>CL_COMMAND_NDRANGE_KERNEL,</td>
</tr>
<tr>
<td></td>
<td>CL_COMMAND_READ_BUFFER</td>
</tr>
<tr>
<td></td>
<td>CL_COMMAND_WRITE_BUFFER</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>CL_EVENT_COMMAND_EXECUTION_STATUS</td>
<td>CL_QUEUED, CL_SUBMITTED,</td>
</tr>
<tr>
<td></td>
<td>CL_RUNNING, CL_COMPLETE</td>
</tr>
<tr>
<td>CL_EVENT_REFERENCE_COUNT</td>
<td>Reference counter of the event object</td>
</tr>
</tbody>
</table>
Exercise 1

Complete code to swap 2 arrays. See “e1/exercise1.c”
OpenCL™ C Language
GPGPU Overview

Introduction to OpenCL™

Getting Started with OpenCL™

OpenCL™ Programming in Detail

The OpenCL™ C Language

- Restrictions
- Data Types
- Type Casting and Conversions
- Qualifiers
- Built-in Functions

Application Optimization and Porting
OpenCL™ C Language

• Language based on ISO C99
  ▪ Some restrictions
• Additions to language for parallelism
  ▪ Vector types
  ▪ Work-items/group functions
  ▪ Synchronization
• Address Space Qualifiers
• Built-in Functions
OpenCL™ C Language Restrictions

• Key restriction in the OpenCL™ language are:
  - No function pointers
  - No bit-fields
  - No variable length arrays
  - No recursion
  - No standard headers
## Data Types

<table>
<thead>
<tr>
<th>Scalar Type</th>
<th>Vector Type (n = 2, 4, 8, 16)</th>
<th>API Type for host app</th>
</tr>
</thead>
<tbody>
<tr>
<td>char, uchar</td>
<td>charn, ucharn</td>
<td>cl_char&lt;n&gt;, cl_uchar&lt;n&gt;</td>
</tr>
<tr>
<td>short, ushort</td>
<td>shortn, ushortn</td>
<td>cl_short&lt;n&gt;, cl_ushort&lt;n&gt;</td>
</tr>
<tr>
<td>int, uint</td>
<td>intn, uintn</td>
<td>cl_int&lt;n&gt;, cl_uint&lt;n&gt;</td>
</tr>
<tr>
<td>long, ulong</td>
<td>longn, ulongn</td>
<td>cl_long&lt;n&gt;, cl_ulong&lt;n&gt;</td>
</tr>
<tr>
<td>float</td>
<td>floatn</td>
<td>cl_float&lt;n&gt;</td>
</tr>
</tbody>
</table>
Using Vector Types

- Creating vector from a set of scalar set

```c
float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);

uint4 u = (uint4)(1);  // u will be (1, 1, 1, 1)

float4 f = (float4)((float2)(1.0f, 2.0f), (float2)(3.0f, 4.0f));

float4 f = (float4)(1.0f, 2.0f);  // error
```
Accessing Vector Components

- Accessing components for vector types with 2 or 4 components
  - `<vector2>.xy, <vector4>.xyzw`

```c
float2 pos;
pos.x = 1.0f;
pos.y = 1.0f;
pos.z = 1.0f; // illegal since vector only has 2 components

float4 c;
c.x = 1.0f;
c.y = 1.0f;
c.z = 1.0f;
c.w = 1.0f;
```
## Accessing Vector with Numeric Index

<table>
<thead>
<tr>
<th>Vector components</th>
<th>Numeric indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 components</td>
<td>0, 1</td>
</tr>
<tr>
<td>4 components</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>8 components</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>16 components</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, e, E, f, F</td>
</tr>
</tbody>
</table>

```c
float8 f;
f.s0 = 1.0f; // the 1st component in the vector
f.s7 = 1.0f; // the 8th component in the vector

float16 x;
f.sa = 1.0f; // or f.sA is the 10th component in the vector
f.sF = 1.0f; // or f.sF is the 16th component in the vector
```
### Handy addressing of Vector Components

<table>
<thead>
<tr>
<th>Vector access suffix</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>.lo</td>
<td>Returns the lower half of a vector</td>
</tr>
<tr>
<td>.hi</td>
<td>Returns the upper half of a vector</td>
</tr>
<tr>
<td>.odd</td>
<td>Returns the odd components of a vector</td>
</tr>
<tr>
<td>.even</td>
<td>Returns the even components of a vector</td>
</tr>
</tbody>
</table>

```c
float4 f = (float4) (1.0f, 2.0f, 3.0f, 4.0f);
float2 low, high;
float2 o, e;

low = f.lo;    // returns f.xy (1.0f, 2.0f)
high = f.hi;   // returns f.zw (3.0f, 4.0f)
o = f.odd;     // returns f.yw (2.0f, 4.0f)
e = f.even;    // returns f.xz (1.0f, 3.0f)
```
Vector Operations

- Support all typical C operator +, -, *, /, &, | etc.
  - Vector operations performed on each component in vector independently

// example 1:

```c
int4 vi0, vi1;
int v;
vi1 = vi0 + v;
```

//is equivalent to:
```
vi1.x = vi0.x + v;
vi1.y = vi0.y + v;
vi1.z = vi0.z + v;
vi1.w = vi0.w + v;
```

// example 2:
```
float4 u, v, w;
w = u + v
w.odd = v.odd + u.odd;
```

// is equivalent to:
```
w.x = u.x + v.x;
w.y = u.y + v.y;
w.z = u.z + v.z;
w.w = u.w + v.w;
```
```
w.y = v.y + u.y;
w.w = v.w + u.w;
```
Type Casting and Conversions

- Implicit conversion of scalar and pointer types
- **Explicit** conversion required for vector types

```c
// implicit conversion
int i;
float f = i;

int4 i4;
float4 = i4;   // not allowed

// explicit conversion through casting
float x;
int i = (int)x;

int4 i4;
float4 f = (float4) i4;   // not allowed
```
Explicit Conversions

- Use built-in conversion functions for explicit conversion (support scalar & vector data types)
  - `convert_<destination_type>(source_type)`

```c
int4 i;
float4 f = convert_float4(i); // converts an int4 vector to float4

float f;
int i = convert_int(f); // converts a float scalar to an integer scalar

int8 i;
float4 f = convert_float4(i); // illegal – components in each vectors must be the same
```
Rounding Mode and Out of Range Conversions

convert_<destination_type><_sat><_roundingMode>(source_type)

- _sat clamps out of range value to nearest representable value
  - Support only integer type
  - Floating point type following IEEE754 rules
- <_roundingMode> specifies the rounding mode

<table>
<thead>
<tr>
<th>_rte</th>
<th>round to nearest even</th>
</tr>
</thead>
<tbody>
<tr>
<td>_rtz</td>
<td>round to nearest zero</td>
</tr>
<tr>
<td>_rtp</td>
<td>round towards positive infinity</td>
</tr>
<tr>
<td>_rtn</td>
<td>round towards negative infinity</td>
</tr>
</tbody>
</table>
| no modifier | default to _rtz for integer  
defaults to _rte for float point |
Rounding Examples

```c
float4 f = (float4)(-1.0f, 252.5f, 254.6f, 1.2E9f);
uchar4 c = convert_uchar4_sat(f);
// c = (0, 253, 255, 255)
// negative value clamped to 0, value > TYPE MAX is set to the type MAX
// -1.0 clamped to 0, 1.2E9f clamped to 255

float4 f = (float4)(-1.0f, 252.5f, 254.6f, 1.2E9f);
uchar4 c = convert_uchar4_sat_rte(f);
// c = (0, 252, 255, 255)
// 252.5f round down to near even becomes 252

int4 i;
float4 = convert_float4(i);
// convert to floating point using the default rounding mode

int4 i;
float4 = convert_float4_rtp(i);
// convert to floating point. Integers values not representable as float
// is round up to the next representable float
```
Reinterpret Data

- Scalar and Vector data can be reinterpreted as another data type
  - `as_<typen>(value)`
- Reinterpret bit pattern in the source to another without modification

```c
uint x = as_uint(1.0f);
// x will have value 0x3f800000

uchar4 c;
int4 d = as_int4(c); // error. result and operand have different size
```
Address Space Qualifiers

- **__global**
  - memory objects allocated in global memory pool

- **__local**
  - fast local memory pool
  - sharing between work-items

- **__constant**
  - read-only allocation in global memory pool

- **__private**
  - accessible by work-item
  - kernel arguments are private
Address Space Qualifiers

- All functions including the \texttt{__kernel} function and their arguments variable are \texttt{__private}
- Arguments to \texttt{__kernel} function declared as a pointer must use \texttt{__global, __local,} or \texttt{__constant}
- Assigning pointer address from on space to another is not allowed;
- Casting from one space to another can cause unexpected behavior.

\begin{tabular}{|l|}
\hline
\texttt{__global float *ptr} & // the pointer ptr is declared in the \texttt{__private} address space and \texttt{__global} address space and points to a float that is in the \texttt{__global} address space \\
\hline
\texttt{int4 x} & // declares an \texttt{int4} vector in the \texttt{__private} address space \\
\hline
\end{tabular}
Image Qualifiers

• Access qualifier for image memory object passed to `__kernel` can be:
  • `__read_only` (default)
  • `__write_only`

• Kernel cannot read and write to same image memory object

```c
__kernel void myfunc(__read_only image2d_t inputImage,
                      __write_only image2d_t outputImage)
```
Work-item Functions

// returns the number of dimensions of the data problem space
uint get_work_dim()

// returns the number total work-items for the specified dimension
size_t get_global_size(dimidx)

// returns the number of local work-items in the work-group specified by dimension
size_t get_local_size(dimidx)

// returns the unique global work-item ID for the specified dimension
size_t get_global_id(dimidx)

// returns the unique local work-item ID in the work-group for the specified dimension
size_t get_local_id(dimidx)

// returns the number of work-groups for the specified dimension
size_t get_num_groups(dimidx)

// returns the unique ID of the work-group being processed by the kernel
size_t get_group_id(dimidx)
Example Work-item Functions

```cpp
__kernel void square(__global int *input, __global int *output) {
    size_t id = get_global_id(0);
    output[id] = input[id] * input[id];
}
```

**get_global_id(0) = 6**

<table>
<thead>
<tr>
<th>input</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>9</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>7</th>
<th>8</th>
<th>5</th>
<th>6</th>
<th>1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>16</td>
<td>25</td>
<td>4</td>
<td>49</td>
<td>1</td>
<td>0</td>
<td>81</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>49</td>
<td>64</td>
<td>25</td>
<td>36</td>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>
### Example Work-item Functions

- `get_global_size(0) → 16`
- `get_local_size(0) → 8`
- `get_num_groups(0) → 2`
- `get_group_id(0) → 0`
- `get_group_id(0) → 1`
- `get_local_id(0) → 5`
- `get_global_id(0) → 13`
Synchronization Functions

• Used to synchronize between work-items
• Synchronization occur only within work-group
• OpenCL uses **barrier** and **fence**
• **Barrier** – blocks current work-item until all work-item in the work-group hits the barrier
  
  ```c
  void barrier(mem_fence_flag)
  ```

• **Fence** – ensures all reads or writes before the memory fence have committed to memory
  
  ```c
  void mem_fence(mem_fence_flag) // orders read and writes operations before the fence
  void read_mem_fence(mem_fence_flag) // orders only reads before the fence
  void write_mem_fence(mem_fence_flag) // orders only writes before the fence
  ```
Exercise 2

Complete kernel function perform matrix transpose. See “e2/transposeMatrix_kernel.cl”
Application Optimization and Porting
GPGPU Overview

Introduction to OpenCL™

Getting Started with OpenCL™

OpenCL™ Programming in Detail

The OpenCL™ C Language

Application Optimization and Porting

- Debugging OpenCL™
- Performance Measurement
- General Optimization Tips
- Porting CUDA to OpenCL™
Debugging OpenCL™

- Debugging OpenCL™ kernels in Linux® using GDB

- Setup:
  - Enable debugging when building program object
    ```c
    err = clBuildProgram(program, 1, devices, "-g", NULL, NULL);
    ```
  - Without modifying source, set environment var
    ```bash
    export CPU_COMPILER_OPTIONS=-g
    ```
  - Set kernel to execute on CPU device ensure kernel is executed deterministically
    ```bash
    export CPU_MAX_COMPUTE_UNITS=1
    ```
Using GDB

• Setting Breakpoints:
  
  
  
  b linenumber
  b function_name | kernel_function_name

• Setting Breakpoint for a kernel function
  
  ▪ Use construct __OpenCL_function_kernel

  __kernel void square(__global int *input, __global int * output)
  b __OpenCL_square_kernel

• Conditional breakpoint
  
  b __OpenCL_square_kernel  if  get_global_id(0) == 5
Performance Measurement

• Built-in mechanism for timing kernel execution
• Enable profiling when creating queue with queue properties `CL_QUEUE_PROFILING_ENABLE`
• Use `clGetEventProfilingInfo()` to retrieve timing information

```c
err = clGetEventProfilingInfo(
    event,  // the event object to get info for
    param_name  // the profiling data to query - see list below
    param_value_size  // the size of memory pointed by param_value
    param_value  // pointer to memory in which the query result is returned
    param_actual_size  // actual number of bytes copied to param_value
);
```

• ATI Stream Profiler plug-in for Visual Studio®
Get Profiling Data with Built-in functions

<table>
<thead>
<tr>
<th>Profiling Data</th>
<th>ulong counter (nanoseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_PROFILING_COMMAND_QUEUE</td>
<td>When command is enqueued</td>
</tr>
<tr>
<td>CL_PROFILING_COMMAND_SUBMIT</td>
<td>When the command has been submitted to device for execution</td>
</tr>
<tr>
<td>CL_PROFILING_COMMAND_START</td>
<td>When command started execution</td>
</tr>
<tr>
<td>CL_PROFILING_COMMAND_END</td>
<td>When command finished execution</td>
</tr>
</tbody>
</table>

```c
cl_event myEvent;
cl_ulong startTime, endTime;

clCreateCommandQueue (…, CL_QUEUE_PROFILING_ENABLE, NULL);
clEnqueueNDRangeKernel (…, &myEvent);
clFinish (myCommandQ); // wait for all events to finish

clGetEventProfilingInfo (myEvent, CL_PROFILING_COMMAND_COMMAND_START, sizeof(cl_ulong), &startTime, NULL);
clGetEventProfilingInfo (myEvent, CL_PROFILING_COMMAND_COMMAND_END, sizeof(cl_ulong), &endTime, NULL);
cl_ulong elapsedTime = endTime - startTime;
```
General Optimization Tips

• Use local memory
• Specific work-group size
• Loop Unrolling
• Reduce Data and Instructions
• Use built-in vector types
General Optimization Tips

• Use local memory
  ▪ Local memory order of magnitude faster
  ▪ Work-items in the same work-group share fast local memory
  ▪ Efficient memory access using collaborative read/write to local memory
General Optimization Tips

- Work-group division
  - Implicit
  - Explicit – recommended
  - AMD GPUs optimized for work-group size multiple of 64.
  - Use `clGetDeviceInfo()` or `clGetKernelWorkGroupInfo()` to determine max group size
General Optimization Tips

- Loop unrolling
  - Overhead to evaluate control-flow and execute branch instructions
  - ATI Stream SDK OpenCL™ compiler performs simple loop unroll
  - Complex loop benefit from manual unroll
General Optimization Tips

• Use built-in vector types
  ▪ Generate efficiently-packed SSE instructions
  ▪ AMD CPUs and GPUs benefit from vectorization

• Reduce Data and Instructions
  ▪ Use smaller version of data set for easy debugging and optimization
  ▪ Performance optimization for smaller data set benefits full-size data set
  ▪ Use profiler data to time data set
Exercise 3

Complete kernel function perform matrix multiplication using local memory. See “e3/multMatrix_kernel.cl”
Matrix Multiplication

\[
\begin{array}{ccc}
A & X & B \\
\text{A(0,1)} & \text{A(1,1)} & \text{A(2,1)} & \text{A(3,1)} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{B(1,0)} & \text{B(1,1)} & \text{B(1,2)} & \text{B(1,3)} \\
\end{array}
\]

\[
C(1,1) = A(0,1) * B(1,0) + A(1,1) * B(1,1) + A(2,1) * B(1,2) + A(3,1) * B(1,3)
\]

// simple matrix multiplication
__kernel void multMatrixSimple(__global float *mO, __global float *mA, __global float *mB,
    uint widthA, uint widthB)
{
    int globalIdx = get_global_id(0);
    int globalIdy = get_global_id(1);
    float sum = 0;
    for (int i = 0; i < widthA; i++)
    {
        float tempA = mA[globalIdy * widthA + i];
        float tempB = mB[i * widthB + globalIdx];
        sum += tempA * tempB;
    }
    mO[globalIdy * widthA + globalIdx] = sum;
}
Optimizing Matrix Multiplication

Matrix Multiplication using local memory

\[ \text{grpC} = \text{grpA0} \times \text{grpB0} + \text{grpA1} \times \text{grpB1} \]
Porting CUDA to OpenCL™

• General terminology

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Work-item</td>
</tr>
<tr>
<td>Thread block</td>
<td>Work-group</td>
</tr>
<tr>
<td>Global memory</td>
<td>Global memory</td>
</tr>
<tr>
<td>Constant memory</td>
<td>Constant memory</td>
</tr>
<tr>
<td>Shared memory</td>
<td>Local memory</td>
</tr>
<tr>
<td>Local memory</td>
<td>Private memory</td>
</tr>
</tbody>
</table>
Porting CUDA to OpenCL™

• Qualifiers

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>global</strong> function</td>
<td>__kernel function</td>
</tr>
<tr>
<td>__device__function</td>
<td>function (no qualifier required)</td>
</tr>
<tr>
<td><strong>constant</strong> variable declaration</td>
<td>__constant variable declaration</td>
</tr>
<tr>
<td><strong>device</strong> variable declaration</td>
<td>__global variable declaration</td>
</tr>
<tr>
<td><strong>shared</strong> variable declaration</td>
<td>__local variable declaration</td>
</tr>
</tbody>
</table>
## Porting CUDA to OpenCL™

### Kernel Indexing

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gridDim</code></td>
<td><code>get_num_groups()</code></td>
</tr>
<tr>
<td><code>blockDim</code></td>
<td><code>get_local_size()</code></td>
</tr>
<tr>
<td><code>blockIdx</code></td>
<td><code>get_group_id()</code></td>
</tr>
<tr>
<td><code>threadIdx</code></td>
<td><code>get_local_id()</code></td>
</tr>
<tr>
<td>No direct global index – needs to be calculated</td>
<td><code>get_global_id()</code></td>
</tr>
<tr>
<td>No direct global size – needs to be calculated</td>
<td><code>get_global_size()</code></td>
</tr>
</tbody>
</table>
### Porting CUDA to OpenCL™

- **Kernel Synchronization**

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__syncthreads()</code></td>
<td><code>barrier()</code></td>
</tr>
<tr>
<td><code>__threadfence()</code></td>
<td><code>no direct equivalent</code></td>
</tr>
<tr>
<td><code>__threadfence_block()</code></td>
<td><code>mem_fence()</code></td>
</tr>
<tr>
<td>No direct equivalent</td>
<td><code>read_mem_fence()</code></td>
</tr>
<tr>
<td>No direct equivalent</td>
<td><code>write_mem_fence()</code></td>
</tr>
</tbody>
</table>
Porting CUDA to OpenCL™

• General API Terminology

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUdevice</td>
<td>cl_device_id</td>
</tr>
<tr>
<td>CUcontext</td>
<td>cl_context</td>
</tr>
<tr>
<td>CUmodule</td>
<td>cl_program</td>
</tr>
<tr>
<td>CUfunction</td>
<td>cl_kernel</td>
</tr>
<tr>
<td>CUdeviceptr</td>
<td>cl_mem</td>
</tr>
<tr>
<td>No direct equivalent</td>
<td>cl_command_queue</td>
</tr>
</tbody>
</table>
# Porting CUDA to OpenCL™

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuInit()</td>
<td>No OpenCL™ initialization required</td>
</tr>
<tr>
<td>cuDeviceGet()</td>
<td>clGetContextInfo()</td>
</tr>
<tr>
<td>cuCtxCreate()</td>
<td>clCreateContextFromType()</td>
</tr>
<tr>
<td>No direct equivalent</td>
<td>clCreateCommandQueue()</td>
</tr>
<tr>
<td>cuModuleLoad()</td>
<td>clCreateProgramWithSource() or clCreateProgramWithBinary()</td>
</tr>
<tr>
<td>cuModuleGetFunction()</td>
<td>clBuildProgram()</td>
</tr>
<tr>
<td>No direct equivalent. CUDA programs are compiled off-line</td>
<td></td>
</tr>
<tr>
<td>cuMemAlloc()</td>
<td>clCreateBuffer()</td>
</tr>
</tbody>
</table>
### Porting CUDA to OpenCL™

<table>
<thead>
<tr>
<th>C for CUDA Terminology</th>
<th>OpenCL™ Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuMemcpyHtoD()</td>
<td>clEnqueueWriteBuffer()</td>
</tr>
<tr>
<td>cuMemcpyDtoH()</td>
<td>clEnqueueReadBuffer()</td>
</tr>
<tr>
<td>cuFuncSetBlockShape()</td>
<td>No direct equivalent; functionality is part of clEnqueueNDRangeKernel()</td>
</tr>
<tr>
<td>cuParamSeti()</td>
<td>clSetKernelArg()</td>
</tr>
<tr>
<td>cuParamSetSize()</td>
<td>No direct equivalent; functionality is part of clSetKernelArg()</td>
</tr>
<tr>
<td>cuLaunchGrid()</td>
<td>clEnqueueNDRangeKernel()</td>
</tr>
<tr>
<td>cuMemFree()</td>
<td>clReleaseMemObj()</td>
</tr>
</tbody>
</table>
Please forward all feedback or information requests regarding this training course to streamcomputing@amd.com
Disclaimer and Attribution

DISCLAIMER
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARK ATTRIBUTION
© 2010 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ATI, the ATI logo, Catalyst, Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Microsoft, Windows, Vista and Visual Studio are registered trademarks, of Microsoft Corporation in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permissions by Khronos.