Introduction to OpenCL programming

Nasos Iliopoulos
George Mason University, resident at Computational Multiphysics Systems Lab.
Center of Computational Material Science
Naval Research Laboratory
Washington, DC, USA

athanasios.iliopoulos.ctr.gr@nrl.navy.mil

August 12-15, 2012, Chicago, Illinois, USA
OpenCL overview

- Industry accepted standard.
  - Vendors provide implementations
- Take advantage of massively parallel execution to accelerate computations.
- Cross-platform in a wide sense:
  - Multiple OSes (Linux, Windows, OS X).
  - Multiple Devices (GPUs, CPUs, ...).
  - Multiple Vendors (AMD, nVidia, Intel, Apple, ...).
- C – like syntax.
OpenCL main differences with cuda
OpenCL main differences with cuda

• (+) Cuda is supported only by nVidia.
OpenCL main differences with cuda

• (+) Cuda is supported only by nVidia.
• (+) OpenCL has a diverse ecosystem.
OpenCL main differences with cuda

- (+) Cuda is supported only by nVidia.
- (+) OpenCL has a diverse ecosystem.
- (+) OpenCL runs on GPUs and CPUs.
OpenCL main differences with cuda

• (+) Cuda is supported only by nVidia.
• (+) OpenCL has a diverse ecosystem.
• (+) OpenCL runs on GPUs and CPUs.
• (+) OpenCL runs on AMD and nVidia GPUs.
OpenCL main differences with cuda

• (+) Cuda is supported only by nVidia.
• (+) OpenCL has a diverse ecosystem.
• (+) OpenCL runs on GPUs and CPUs.
• (+) OpenCL runs on AMD and nVidia GPUs.
• (+) OpenCL uses the native compiler.
OpenCL main differences with cuda

• (+) Cuda is supported only by nVidia.
• (+) OpenCL has a diverse ecosystem.
• (+) OpenCL runs on GPUs and CPUs.
• (+) OpenCL runs on AMD and nVidia GPUs.
• (+) OpenCL uses the native compiler.
• (−) OpenCL is slightly slower than Cuda on nVidia GPUs. (≈5%)
• Platform model (Host + OpenCL devices)
• Execution model (kernels-functions + programs)
• Memory model (storage of arrays – buffers)
• Programming model (data parallel or task parallel)
OpenCL Platform model
OpenCL Platform model

Host (i.e. PC)
OpenCL Platform model

Host (i.e. PC)

Compute device
(i.e. GPU, CPU, ...)

(i.e. GPU, CPU, ...
OpenCL Platform model

- **Host (i.e. PC)**
- **Compute device** (i.e. GPU, CPU, ...)
- **Compute unit**
  - Executes work-groups that are collections of work-items
OpenCL Platform model

- **Host (i.e. PC)**
- **Compute device** (i.e. GPU, CPU, ...)
- **Compute unit**
  - Executes work-groups that are collections of work-items
- **Processing Element**
  - Virtual processor executing work items
OpenCL Execution Model

- **Kernel**: Managed at the Device level
  - Analogous to a function
- **Program**: Collection of kernels
  - Analogous to a library of functions
- **Application queue**: Managed at the Host level
  - Kernels queued in order
  - Kernels executed in-order or out-of-order
OpenCL memory model
OpenCL memory model

PROCESSING ELEMENT:
- Virtual Processor
- Maps to a physical processor at some point in time
OpenCL memory model

PROCESSING ELEMENT:
- Virtual Processor
- Maps to a physical processor at some point in time
OpenCL memory model

Compute unit 1

PE 1
Private Memory 1

PE N
Private Memory N

Compute unit is usually referred to as a “Work Group”
OpenCL memory model

Compute unit 1

PE 1
Private Memory 1

... PE N
Private Memory N

Local Memory 1

Compute unit is usually referred to as a “Work Group”
OpenCL memory model

Compute unit 1

Compute unit N

PE 1
Private Memory 1
Local Memory 1

PE N
Private Memory N

PE 1
Private Memory 1

PE N
Private Memory N

Local Memory N
OpenCL memory model

Compute Device

Compute unit 1

**PE 1**
- Private Memory 1
- Local Memory 1

...**PE N**
- Private Memory N

Compute unit N

**PE 1**
- Private Memory 1
- Local Memory N

...**PE N**
- Private Memory N
OpenCL memory model

Compute Device

Compute unit 1

- PE 1
  - Private Memory 1
  - Local Memory 1

... etc...

- PE N
  - Private Memory N
  - Local Memory N

Compute unit N

- PE 1
  - Private Memory 1
  - Local Memory N

... etc...

- PE N
  - Private Memory N
  - Local Memory N

Global / Constant Memory Data Cache
OpenCL memory model

Compute Device

Compute unit 1

PE 1
Private Memory 1
Local Memory 1

PE N
Private Memory N

Compute unit N

PE 1
Private Memory 1
Local Memory N

PE N
Private Memory N

Global / Constant Memory Data Cache

Constant Memory

Global Memory
Programming Model
• Supports two programming models: data parallel and task parallel
• Supports two programming models: data parallel and task parallel

• Data parallel: Processing Elements execute the same task on different pieces of distributed data.
• Supports two programming models: data parallel and task parallel

• **Data parallel**: Processing Elements execute the **same task** on different pieces of distributed data. Example: array increment

```
5 5 3 2 6 ...
```

```
6 6 4 3 7 ...
```
• Supports two programming models: data parallel and task parallel

• **Data parallel**: Processing Elements execute the **same task** on different pieces of distributed data. Example: array increment

```
5 5 3 2 6 ...
```

```
6 6 4 3 7 ...
```

Element increment is processed in parallel
• Supports two programming models: data parallel and task parallel

• **Data parallel**: Processing Elements execute the same task on different pieces of distributed data. Example: array increment

  | 5 | 5 | 3 | 2 | 6 | ... | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ... |

  Element increment is processed in parallel

• **Task parallel**: Each processing element executes a different task on the same or different data.
Programming Model

• Supports two programming models: data parallel and task parallel

• **Data parallel**: Processing Elements execute the **same task** on different pieces of distributed data. Example: array increment

  ![Data parallel example](image)

  Element increment is processed in parallel

• **Task parallel**: Each processing element executes a **different task** on the same or different data.

  ![Task parallel example](image)
• Supports two programming models: data parallel and task parallel

• **Data parallel**: Processing Elements execute the **same task** on different pieces of distributed data. Example: array increment

```
5 5 3 2 6 ...
6 6 4 3 7 ...
```

Element increment is processed in parallel

• **Task parallel**: Each processing element executes a **different task** on the same or different data.

```
5 5 3 2 6 ...
6 6 4 3 7 ...
```

Task A and B executed in parallel
OpenCL execution process

• Create an OpenCL **context** bound to a Device type.
• Create a **command queue** on one of the devices of the context.
• Allocate and create **memory buffer** objects.
• Create and **build** the OpenCL program.
• **Create a kernel object** from the kernels in the program.
• **Execute** the kernel.
• **Read** results if needed.
• **Clean up.**
OpenCL example – array increment

- Array increment

```
5 5 3 2 6 ...
```

```
6 6 4 3 7 ...
```
Array increment

**C++ - SERIAL VERSION**

```cpp
void aInc( const unsigned int n, 
           float *a) {

    for (std::size_t i=0; i!=n; i++)
        a[i]=a[i]+1.0;

}
```
OpenCL example – array increment

Array increment

C++ - SERIAL VERSION

```cpp
void aInc(const unsigned int n, float *a) {
    for (std::size_t i=0; i!=n; i++)
        a[i]=a[i]+1.0;
}
```

OpenCL VERSION

```c
__kernel void aInc(__global const unsigned int n, __global float *a) {
    unsigned int i=get_global_id(0);
    if (i<n)
        a[i] = a[i]+1.0;
}
```
OpenCL example – array increment

Compute Device

Compute unit 1

- PE 1
  - Private Memory 1
  - Local Memory 1

PE N
- Private Memory N

Compute unit N

- PE 1
  - Private Memory 1
  - Local Memory N

PE N
- Private Memory N

Global / Constant Memory Data Cache

Constant Memory

Global Memory
Array increment

C++ - SERIAL VERSION

```c++
void aInc( const unsigned int n,
           float *a) {
    for (std::size_t i=0; i!=n; i++)
        a[i]=a[i]+1.0;
}
```

OpenCL VERSION

```opencl
__kernel void aInc( __global const unsigned int n,
                   __global float *a) {
    unsigned int i=get_global_id(0);
    if (i<n)
        a[i] = a[i]+1.0;
}
```

- A kernel can be thought as the body of a for-loop
- Note how **indexing** is happening in the OpenCL version
OpenCL example – array increment

Typical compilation setup

• Include the OpenCL header:

    #include <CL/opencl.h>

• Compiler include paths (i.e. nVidia SDK):

    -I$SDK_PATH/OpenCL/common/inc -I$SDK_PATH/shared/inc

• Link libraries:

    -lOpenCL
OpenCL example – array increment

Initialization

• Get an OpenCL platform:
  
  error = clGetPlatformIDs(1, &cpPlatform, NULL);
  If (error != CL_SUCCESS) { // Error handling}

• Get the devices
  
  error= clGetDeviceIDs(cpPlatform, CL_DEVICE_TYPE_GPU, 1, &cdDevice, NULL);

• Create the context
  
  GPUContext = clCreateContext(0, 1, &cdDevice, NULL, NULL, &error);

• Create a command - queue
  
  cqCommandQueue = clCreateCommandQueue(cxGPUContext, cdDevice, 0, &error);
OpenCL example – array increment

Compile the kernel

- Create the program object

```c
cpProgram = clCreateProgramWithSource(cxGPUContext, 1, (const char**) &cSourceCL, &szKernelLength, &error);
```
OpenCL example – array increment

Compile the kernel

- Create the program object

```c
cpProgram = clCreateProgramWithSource(cxGPUContext, 1, (const char **)&cSourceCL, &szKernelLength, &error);

__kernel void aInc(__global const unsigned int n, __global float *a) {
    unsigned int i = get_global_id(0);
    if (i < n)
        a[i] = a[i]+1.0;
}
```

```c
// Compile the kernel
```
OpenCL example – array increment

Compile the kernel

• Create the program object

    cpProgram = clCreateProgramWithSource(cxGPUContext, 1, (const char **) &cSourceCL, &szKernelLength, &error);

• Compile the program

    error = clBuildProgram(cpProgram, 0, NULL, NULL, NULL, NULL, NULL);

• Create the kernel

    ckKernel = clCreateKernel(cpProgram, "aInc", &error);
OpenCL example – array increment

Load some data to the GPU

• Create and fill an array on the host

```cpp
std::vector<float> a_host(szGlobalWorkSize);
(for std::size_t i=0; i!=numElements; i++)
    a_host[i]=i;
```

• Create a buffer on the GPU

```cpp
cmDevSrcA = clCreateBuffer(cxGPUContext, CL_MEM_READ_ONLY, sizeof(cl_float) * szGlobalWorkSize, NULL, &error);
```

• Asynchronously Copy the data to the GPU

```cpp
error = clEnqueueWriteBuffer(cqCommandQueue, cmDevSrcA, CL_FALSE, 0, sizeof(cl_float) * szGlobalWorkSize, &a_host[0], 0, NULL, NULL);
```
OpenCL example – array increment

Set kernel arguments and execute it

• Set the kernel arguments

```c
error = clSetKernelArg(ckKernel, 0, sizeof(cl_uint), (void*)&numElements);
error |= clSetKernelArg(ckKernel, 1, sizeof(cl_mem), (void*)&cmDevSrcA);
```

• Execute the kernel

```c
Error = clEnqueueNDRangeKernel(cqCommandQueue, ckKernel, 1, NULL, &szGlobalWorkSize, &szLocalWorkSize, 0, NULL, NULL);
```
OpenCL example – array increment

Post-processing

• Get the result from the GPU

```c
error = clEnqueueReadBuffer(cqCommandQueue, cmDevSrcA, CL_TRUE, 0, sizeof(cl_float) * szGlobalWorkSize, dst, 0, NULL, NULL);
```

•

•

•
OpenCL example – array increment

Array increment performance
C serial version vs OpenCL

Execution Time (sec)

C Serial version - i7 @3.9GHz

OpenCL - Tesla c1060

About 9x speedup.
OpenCL example – array reversal

```
5 5 3 2 6 ...

... 6 2 3 5 5
```
OpenCL example – array reversal

A simple kernel
A simple kernel

__kernel void
ArrayRev( __global const float* in,
          __global float *out,
          int iNumElements)
A simple kernel

__kernel void
ArrayRev(__global const float* in,
         __global float *out,
         int iNumElements)
{
    // get index into global data array
    const int iGID = get_global_id(0);

    // bound check
    if (iGID >= iNumElements) return;
A simple kernel

__kernel void
ArrayRev(__global const float* in,
         __global float *out,
         int iNumElements)
{
    // get index into global data array
    const int iGID = get_global_id(0);

    // bound check
    if (iGID >= iNumElements) return;

    // Run “out” reversely
    const int oGID = iNumElements - iGID - 1;
    out[oGID] = in[iGID];
}
OpenCL example – array reversal

A simple kernel
Modifications on the HOST code

• Create buffers on the GPU

\[
\text{cmDevSrcA} = \text{clCreateBuffer} (\text{cxGPUContext}, \text{CL\_MEM\_READ\_ONLY}, \text{sizeof(cl\_float) * szGlobalWorkSize}, \text{NULL, &error});
\]

\[
\text{cmDevDstB} = \text{clCreateBuffer} (\text{cxGPUContext}, \text{CL\_MEM\_READ\_ONLY}, \text{sizeof(cl\_float) * szGlobalWorkSize}, \text{NULL, &error});
\]

• Set the kernel arguments

\[
\text{error} = \text{clSetKernelArg} (\text{ckKernel}, 0, \text{sizeof(cl\_mem)}, (\text{void*})&\text{cmDevSrcA});
\]

\[
\text{error} |\!|= \text{clSetKernelArg} (\text{ckKernel}, 1, \text{sizeof(cl\_mem)}, (\text{void*})&\text{cmDevDstB});
\]

\[
\text{error} |\!|= \text{clSetKernelArg} (\text{ckKernel}, 2, \text{sizeof(cl\_uint)}, (\text{void*})&\text{numElements});
\]
Array reversal performance
C serial version vs OpenCL
Array reversal performance
C serial version vs OpenCL

About 2.3x speedup.
OpenCL example – array reversal

Why?
SIMPLE ARRAY INCREMENT CASE
OpenCL example – array reversal

Why?
SIMPLE ARRAY INCREMENT CASE
OpenCL example – array reversal

Why?
SIMPLE ARRAY INCREMENT CASE

Global memory

16 word packet

16 word packet

16 word packet

Local memory

16 word packet
OpenCL example – array reversal

Why?
SIMPLE ARRAY INCREMENT CASE

Global memory

16 word packet

16 word packet

16 word packet

16 word packet

Local memory

Thread 1

Thread 2

Simple array increment case.
OpenCL example – array reversal

Why?
SIMPLE ARRAY INCREMENT CASE

Global memory

16 word packet

16 word packet

16 word packet

Local memory

Thread 1

Thread 2

Back to Global memory

16 word packet
OpenCL example – array reversal

Why?
ARRAY REVERSAL CASE

Global memory 16 word packet Local memory Thread 10

Global memory

Local memory Thread 538

Global memory

Local memory

Back to Global memory ...

Waste of memory operations
OpenCL example – array reversal

Solution: Bring data in local memory in order to achieve coalescence
Solution: Bring data in local memory in order to achieve coalescence

Input array
Solution: Bring data in local memory in order to achieve coalescence
OpenCL example – array reversal

Solution: Bring data in local memory in order to achieve coalescence

Input array                Local memory

One workgroup
OpenCL example – array reversal

Solution: Bring data in local memory in order to achieve coalescence

Input array  Local memory  Output array

One workgroup
OpenCL example – array reversal

Solution: Bring data in local memory in order to achieve coalescence

Input array  Local memory  Output array

One workgroup
OpenCL example – array reversal

Solution: Bring data in local memory in order to achieve coalescence

Input array → Local memory → Output array
OpenCL example – array reversal

An improved kernel

__kernel void
ArrayRev(__global const float* in,
        __global float *out,
        __local float *shared,
        int iNumElements)
An improved kernel

__kernel void
ArrayRev(__global const float* in,
          __global float *out,
          __local float *shared,
          int iNumElements)
{
    const int lid = get_local_id(0);
    const int lsize = get_local_size(0);
An improved kernel

```c
__kernel void ArrayRev(__global const float* in,
                      __global float *out,
                      __local float *shared,
                      int iNumElements)
{
    const int lid = get_local_id(0);
    const int lsize = get_local_size(0);

    shared[lsize-lid-1]=in[get_global_id(0)];
```
An improved kernel

__kernel void
ArrayRev(__global const float* in,
    __global float* out,
    __local float* shared,
    int iNumElements)
{
    const int lid = get_local_id(0);
    const int lsize = get_local_size(0);

    shared[lsize-lid-1]=in[get_global_id(0)];
    barrier(CLK_LOCAL_MEM_FENCE);
OpenCL example – array reversal

An improved kernel

Input array  Local memory  Output array

Wait until all threads have finished fetching data to local memory
OpenCL example – array reversal

An improved kernel

Input array   Local memory   Output array

Wait untill ALL threads have finished fetching data to local memory
OpenCL example – array reversal

An improved kernel

Input array  Local memory  Output array
An improved kernel

__kernel void
ArrayRev(__global const float* in,
__global float *out,
__local float *shared,
int iNumElements)
{
    const int lid = get_local_id(0);
    const int lsize = get_local_size(0);

    shared[lsize-lid-1]=in[get_global_id(0)];
    barrier(CLK_LOCAL_MEM_FENCE);
An improved kernel

__kernel void
ArrayRev(__global const float* in,
         __global float *out,
         __local float *shared,
         int iNumElements)
{
    const int lid = get_local_id(0);
    const int lsize = get_local_size(0);

    shared[lsize-lid-1]=in[get_global_id(0)];
    barrier(CLK_LOCAL_MEM_FENCE);

    int oGID = iNumElements - (get_group_id(0)+1)*lsize+lid;
    if (oGID<0) return;
    out[oGID] = shared[lid];
}
An improved kernel

Modifications on the HOST code

• Define the shared array size (local to each workgroup):

```c
size_t shared_size = szLocalWorkSize * sizeof(cl_float);
```

Number of work items in each work group
An improved kernel

Modifications on the HOST code

• Define the shared array size (local to each workgroup):

  size_t shared_size = szLocalWorkSize * sizeof(cl_float);

• Set the kernel arguments

  error = clSetKernelArg(ckKernel, 0, sizeof(cl_mem), (void*) &cmDevSrcA);
  error |= clSetKernelArg(ckKernel, 1, sizeof(cl_mem), (void*) &cmDevDstB);
  error |= clSetKernelArg(ckKernel, 2, shared_size, NULL);
  error |= clSetKernelArg(ckKernel, 3, sizeof(cl_uint), (void*) &numElements);
Array reversal performance – improved kernel
C serial version vs OpenCL

About 7.4x speedup.
Suggested internet resources
Suggested internet resources

OpenCL official specification:
http://www.khronos.org/opencl/

SDKs / Drivers / Tutorials /Tools

AMD:

Intel:

nVidia:
http://developer.nvidia.com/opencl

Apple:

IBM Power architecture:
http://www.alphaworks.ibm.com/tech/opencl
Questions?