Heterogeneous Computing using openCL
lecture 3

F21DP Distributed and Parallel Technology

Sven-Bodo Scholz
Recap: Initialise Device

- Declare context
- Choose a device from context
- Using device and context create a command queue

```c
cl_context myctx = clCreateContextFromType( 
    0, CL_DEVICE_TYPE_GPU, 
    NULL, NULL, &ciErrNum);
```

```c
ciErrNum = clGetDeviceIDs(0, 
    CL_DEVICE_TYPE_GPU, 
    1, &device, cl_uint *num_devices)
```

```c
cl_commandqueue myqueue;
myqueue = clCreateCommandQueue( 
    myctx, device, 0, &ciErrNum);
```
Recap: Create Buffers

- Create buffers on device
  - Input data is read-only
  - Output data is write-only

```
cl_mem d_ip = clCreateBuffer(  
    myctx, CL_MEM_READ_WRITE,  
    mem_size,  
    NULL, &ciErrNum);
```

```
cl_mem d_op = clCreateBuffer(  
    myctx, CL_MEM_READ_WRITE,  
    mem_size,  
    NULL, &ciErrNum);
```

- Transfer input data to the device

```
ciErrNum = clEnqueueWriteBuffer (  
    myqueue, d_ip, CL_TRUE,  
    0, mem_size, (void *)src_vector,  
    0, NULL, NULL)
```
Recap: Build Program, Select Kernel

// create the program
c_l_program myprog = clCreateProgramWithSource
        ( myctx, 1, (const char **)&source, 
        &program_length, &ciErrNum);

// build the program
cliErrNum = clBuildProgram( myprog, 0, 
        NULL, NULL, NULL, NULL);

// Use the “relax” function as the kernel
cl_kernel mykernel = clCreateKernel ( 
        myprog, “relax”, 
        error_code)
Recap: Set Arguments, Enqueue Kernel

```c
// Set Arguments
clSetKernelArg(mykernel, 0, sizeof(cl_mem), (void *)&d_ip);
clSetKernelArg(mykernel, 1, sizeof(cl_mem), (void *)&d_op);
clSetKernelArg(mykernel, 2, sizeof(cl_int), (void *)&len);
...
```

```c
// Set local and global workgroup sizes
size_t localws[2] = {16};
size_t globalws[2] = {LEN}; // Assume divisible by 16
```

```c
// execute kernel
clEnqueueNDRangeKernel(
    myqueue, myKernel,
    2, 0, globalws, localws,
    0, NULL, NULL);
```
Recap: Read Back Result

- Only necessary for data required on the host
- Data output from one kernel can be reused for another kernel
  - Avoid redundant host-device IO

```c
// copy results from device back to host
clEnqueueReadBuffer(
    myctx, d_op,
    CL_TRUE,     //Blocking Read Back
    0, mem_size, (void *) op_data,
    NULL, NULL, NULL);
```
The Big Picture

• Introduction to Heterogeneous Systems
• OpenCL Basics
• Memory Issues
• Scheduling
Aspects

• Introduction to GPU bus addressing
• Coalescing memory accesses (Global Memory)
• Synchronisation issues
• Local and Private Memory
Example

– Array \( X \) is a pointer to an array of integers (4-bytes each) located at address 0x00001232

\[
\begin{array}{cccccccc}
0 & 0x00001232 \\
1 & 0x00001236 \\
2 & \\
3 & \\
4 & \\
5 & \\
6 & \\
7 & 0x00001248 \\
\end{array}
\]

– A thread wants to access the data at \( X[0] \):

\[
\text{int tmp = X[0];}
\]
Bus Addressing

- Assume that the memory bus is 32-bytes (256-bits) wide
  - This is the width on a Radeon 5870 GPU
- The byte-addressable bus must make accesses that are aligned to the bus width, so the bottom 5 bits are masked off

Desired address: \(0x00001232\)
Bus mask: \(0xFFFFFFFFE0\)
Bus access: \(0x00001220\)

- Any access in the range \(0x00001220\) to \(0x0000123F\) will produce the address \(0x00001220\)
Bus Addressing

• All data in the range 0x00001220 to 0x0000123F is returned on the bus
• In this case, 4 bytes are useful and 28 bytes are wasted
Coalescing Memory Accesses

• To fully utilize the bus, GPUs combine the accesses of multiple threads into fewer requests when possible
• Consider the following OpenCL kernel code:

  ```c
  int tmp = X[get_global_id(0)];
  ```

• Assuming that array X is the same array from the example, the first 16 threads will access addresses 0x00001232 through 0x00001272
• If each request was sent out individually, there would be 16 accesses total, with 64 useful bytes of data and 448 wasted bytes
  – Notice that each access in the same 32-byte range would return exactly the same data
Coalescing Memory Accesses

• When GPU threads access data in the same 32-byte range, multiple accesses are combined so that each range is only accessed once
  – Combining accesses is called *coalescing*

• For this example, only 3 accesses are required
  – If the start of the array was 256-bit aligned, only two accesses would be required
Coalescing Memory Accesses

• Recall that for AMD hardware, 64 threads are part of a wavefront and must execute the same instruction in a SIMD manner

• For the AMD 5870 GPU, memory accesses of 16 consecutive threads are evaluated together and can be coalesced to fully utilize the bus
  – This unit is called a quarter-wavefront and is the important hardware scheduling unit for memory accesses
Coalescing Memory Accesses

- Global memory performance for a simple data copying kernel of entirely coalesced and entirely non-coalesced accesses on an ATI Radeon 5870

![Graph showing bandwidth (in GB/s) against data size (in MB) for coalesced and uncoalesced access patterns. The graph illustrates that coalesced access consistently outperforms uncoalesced access across all data sizes.]
Coalescing Memory Accesses

- Global memory performance for a simple data copying kernel of entirely coalesced and entirely non-coalesced accesses on an NVIDIA GTX 285
Working on Global Memory

<table>
<thead>
<tr>
<th>global_id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>_global x[16]</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>_global x2[16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

kernel:

- `x[global_id] = 42;`
- `x[3] = global_id;`
- `x[local_id] = global_id;`
To Sync or Not to Sync....

<table>
<thead>
<tr>
<th>global_id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>__global x[16]</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
<td>26</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>__global x2[16]</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
<td>26</td>
<td>28</td>
<td>30</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

kernel:  

```
x[global_id] = foo( global_id);
Sync?
X2[global_id] = x[(global_id+2)%16];
```
But this can happen too!

```plaintext
 Assumed a warp size of 2
```

```plaintext
global_id          0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
local_id           0 1 2 3 0 1 2 3
__global x[16]     0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
__global x2[16]    4 6 0 0 12 14 16 18 20 22 24 26 28 30 0 2

kernel:
  x[global_id] = foo( global_id);
  Sync?
  X2[global_id] = x[(global_id+2)%16];
```
Enforcing Synchronisation

```c
kernel:
x[global_id] = foo( global_id);
barrier(CLK_GLOBAL_MEM_FENCE);
X2[global_id] = x[(global_id+2)%16];
```
To Sync or Not to Sync II

Assuming a warp size of 2

Kernel:
\[
x[\text{global}_i] = \text{foo}(\text{global}_i);
\]
\[
\text{barrier(\text{CLK\_GLOBAL\_MEM\_FENCE})}, \text{?????}
\]
\[
\text{x2}[\text{((global}_i+14) \mod 16)] = x[\text{global}_i];
\]
To Sync or Not to Sync III

Assuming a warp size of 2

kernel:

```
x[global_id] = foo( global_id);
barrier(CLK_GLOBAL_MEM_FENCE);
X[(global_id+14)%16] = x[global_id];
```

Now lets schedule this one!

Does not help!!!
Memories

- Like AMD, a subset of hardware memory exposed in OpenCL
- Configurable shared memory is usable as local memory
  - Local memory used to share data between items of a work group at lower latency than global memory
- Private memory utilizes registers per work item
A solution

Assuming a warp size of 2

Now let's schedule this one!

<table>
<thead>
<tr>
<th>global_id</th>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_id</td>
<td>0 1 2 3 0 1 2 3 0 1 2 3 2 3</td>
</tr>
<tr>
<td>__global x[16]</td>
<td>0 2 4 6 8 10 12 14 16 18 20 22 28 30 0 2</td>
</tr>
<tr>
<td>__private p</td>
<td>0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30</td>
</tr>
</tbody>
</table>

kernel:

```c
x[global_id] = foo( global_id);
p = x[global_id];
barrier(CLK_GLOBAL_MEM_FENCE);
X[(global_id+14)%16] = p;
```
Work Groups, Threads, Local and Global Memory

```
global_id  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
local_id   0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
__global x[16]
__local y[4]
__local z
__private p
```
Work Groups, Threads, Local and Global Memory

<table>
<thead>
<tr>
<th>global_id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>__global x[16]</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>__local y[4]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>?</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__local z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__private p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

kernel:

\[ y[\text{local_id}] = \text{global_id} + 3; \]
\[ y[3] = \text{local_id}; \]
\[ y[\text{global_id}] = \text{local_id}; \]

Out of bounds access!!
To Sync or Not to Sync?

| global_id | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| local_id  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  | 0  | 1  | 2  | 3  |
| __global x[16] | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 |
| __local y[4] | 3  | 4  | 5  | 6  | ... | ... | ... |
| __local z   | 21 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| __private p | 0  | 1  | 9  | 3  | ... | ... | ... |

kernel: y[local_id] = foo( local_id);

barrier(CLK_LOCAL_MEM_FENCE), ?????

x[global_id+2 % 4] = y[local_id];
To Sync or Not to Sync?

<table>
<thead>
<tr>
<th>global_id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>__global x[16]</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>__local y[4]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__local z</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__private p</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Kernel:

\[ y[(local_id+1) \% 4] = \text{foo}(\text{local_id}); \]

\[ \text{barrier}(\text{CLK\_LOCAL\_MEM\_FENCE}); ??? ?? \]

\[ X[\text{global_id+2} \% 4] = y[\text{local_id}]; \]