Administrivia

- HW2 out, due Nov. 2.
- HW3 will be out by Nov. 2.
- Run test.py from https://github.com/fasrc/cs205_Intro_to_Odyssey to check OpenCL.
- Final Projects: keep sending ideas.
Final Projects

• Read over cs205.org Projects page, look at projects from previous years.

• Proposals will be due Nov. 6, we’ll provide reviews back by the 9th.
  
  • You can propose an idea before then, even if not completely ready.

• Proposing early will make it easier for everyone.
More OpenCL intro.

https://github.com/fasrc/cs205_Intro_to_Odyssey
test.py
Review

• CPU manages memory and scheduling.
• Large number of threads (global_size).
• Broken up into work groups (local_size).
• Synchronization between threads only possible at group level.
Global and Local Sizes

• When we launch a kernel, we give the global size (total number of threads), and local size (size of each work group).

• local_size has to evenly divide global_size.

• Both can be 1D, 2D, …
N, local_size, global_size

N = (10, 10)
\( N, \text{local}_\text{size}, \text{global}_\text{size} \)

\[
N = (10, 10)
\]

\[
\text{local}_\text{size} = (4, 4)
\]
N, local_size, global_size

N = (10, 10)
local_size = (4, 4)
global_size = (12, 12)
N, local_size, global_size

N = (10, 10)
local_size = (4, 4)
global_size = (12, 12)

Need to mask off these threads.
__kernel void
add_images(__global float* out,
    __global const float *im1,
    __global const float *im2,
    const unsigned int w,
    const unsigned int h)
{
    unsigned int x = get_global_id(0);
    unsigned int y = get_global_id(1);

    int offset;

    // Make sure we stay in-bounds
    if ((x < w) && (y < h)) {
        offset = y * w + x;
        out[offset] = im1[offset] + im2[offset];
    }
}
__kernel void add_images(__global float* out,
    __global const float *im1,
    __global const float *im2,
    const unsigned int w,
    const unsigned int h)
{
    unsigned int x = get_global_id(0);
    unsigned int y = get_global_id(1);

    int offset;

    // Make sure we stay in-bounds
    if ((x < w) && (y < h)) {
        offset = y * w + x;
        out[offset] = im1[offset] + im2[offset];
    }
}
__kernel void
add_images(__global float* out,
   __global const float *im1,
   __global const float *im2,
   const unsigned int w,
   const unsigned int h)
{
    unsigned int x = get_global_id(0);
    unsigned int y = get_global_id(1);

    int offset;

    // Make sure we stay in-bounds
    if ((x < w) && (y < h)) {
      offset = y * w + x;
      out[offset] = im1[offset] + im2[offset];
    }
}
Image Coordinate Math
Image Coordinate Math

We have a memory address (aka pointer) to (0, 0).
Image Coordinate Math

We have a memory address (aka pointer) to \((0, 0)\).

Adjacent pixels in \(x\) are adjacent in memory.
Image Coordinate Math

We have a memory address (aka pointer) to (0, 0).

Adjacent pixels in x are adjacent in memory.

Adjacent rows are adjacent in memory, and w pixels apart.
Image Coordinate Math

We have a memory address (aka pointer) to $(0, 0)$.

Adjacent pixels in $x$ are adjacent in memory.

Adjacent rows are adjacent in memory, and $w$ pixels apart.

$(x, y)$ pixel is $(y \times w + x)$ away from $(0, 0)$.
Nothing is ever simple

• Note: Numpy array dimensions are flipped from images.

• Instead of \((x, y)\), \((y, x) == (rows, columns)\), and indexed in that order.

• (Like a matrix.)
A 1D filtering algorithm
Repeatedly add \(\frac{\text{neighbors}}{2}\) and divide by 4.
A 1D filtering algorithm

__kernel void smooth(__global float* out,
    __global const float *in,
    const unsigned int N)
{
    unsigned int x = get_global_id(0);

    // Make sure we stay in-bounds
    if (…) {
        out[x] = (in[x - 1] +
            2 * in[x] +
            in[x + 1]) / 4
    }
}
A 1D filtering algorithm

```c
__kernel void smooth(__global float* out,
    __global const float *in,
    const unsigned int N)
{
    unsigned int x = get_global_id(0);

    // Make sure we stay in-bounds
    if (...) {
        out[x] = (in[x - 1] +
                  2 * in[x] +
                  in[x + 1]) / 4
    }
}
```

How many memory reads?
Local memory

• Local memory is on-chip, like a managed L1 cache.
• We have to allocate it at kernel-launch.
• It is shared within a work-group.
• It can be used like cache, or as a shared scratchpad.
__kernel void smooth(__global float* out, __global const float *in, __local float *tmp, const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);

    // Make sure we stay in-bounds
    if (...) {
        tmp[lx] = in[x];
    }

    // Make sure we stay in-bounds
    if (...) {
        out[x] = (tmp[lx - 1] + 2 * tmp[lx] + tmp[lx + 1]) / 4
    }
}
__kernel void
smooth(__global float* out,  
    __global const float *in,  
    __local float *tmp,  
    const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);
    // Make sure we stay in-bounds
    if (...) {
        tmp[lx] = in[x];
    }
    // Make sure we stay in-bounds
    if (...) {
        tmp[lx] = in[x];
    }
    // Make sure we stay in-bounds
    if (...) {
        out[x] = (tmp[lx - 1] + 
                  2 * tmp[lx] + 
                  tmp[lx + 1]) / 4
    }
}

How many global memory reads?
Allocating Local Memory

```python
program.kernel(queue, global_size, local_size,
               gpu_out, gpu_in,
               cl.LocalMemory(local_size[0]),
               N)
```

Remember: local memory becomes part of the work-group's thread context.

Too large, and not many work-groups can be running at the same time.
__kernel void smooth(__global float* out,
    __global const float *in,
    __local  float *tmp,
    const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);

    // Make sure we stay in-bounds
    if (…) {
        tmp[lx] = in[x];
    }

    // Make sure we stay in-bounds
    if (…) {
        out[x] = (tmp[lx - 1] +
               2 * tmp[lx] +
               tmp[lx + 1]) / 4
    }
}
__kernel void smooth(__global float* out,
                    __global const float *in,
                    __local  float *tmp,
                    const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);

    // Make sure we stay in-bounds
    if (…) {
        tmp[lx] = in[x];
    }

    _barrier(CLK_LOCAL_MEM_FENCE);

    // Make sure we stay in-bounds
    if (…) {
        out[x] = (tmp[lx - 1] +
                   2 * tmp[lx] +
                   tmp[lx + 1]) / 4
    }
}
__kernel void 
smooth(__global float* out,
    __global const float *in,
    __local  float *tmp,
    const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);

    // Make sure we stay in-bounds
    if (...) {
        tmp[lx] = in[x];
    }

    _barrier(CLK_LOCAL_MEM_FENCE);

    // Make sure we stay in-bounds
    if (...) {
        out[x] = (tmp[lx - 1] +
               2 * tmp[lx] +
               tmp[lx + 1]) / 4
    }
}
Synchronization

• `__barrier()` wait for all threads in a work group to reach this point.

• `__mem_fence()` - make sure all read/writes have flushed.
Reading Global memory

- On GPUs, access to global memory will read between 16 and 32 adjacent floats at a time.

- Threads in work-group can share the result.

- Reading scattered data causes serialized reads, which is slow (could be 32x slower or worse).
Coalesced Reads

• Try to read global memory in contiguous blocks, using multiple threads.

• Good: thread N reads memory location data[N].

• Bad: thread N reads memory location data[K*N].

• When dealing with images, remember layout direction!
__kernel void smooth(__global float* out,
   __global const float *in,
   __local float *tmp,
   const unsigned int N)
{
    unsigned int x = get_global_id(0);
    unsigned int lx = get_local_id(0);

    // Make sure we stay in-bounds
    if (...) {
        tmp[lx] = in[x];
    }

    // Make sure we stay in-bounds
    if (...) {
        out[x] = (tmp[lx - 1] +
                  2 * tmp[lx] +
                  tmp[lx + 1]) / 4
    }
}
Next time

• Getting info about GPUs
• More about multidimensional Kernels
• Benchmarking

• Try to get OpenCL installed. Test it on your machine.
• FINAL PROJECTS - post on Piazza, talk to us.