OpenCL, double buffering using two command-queues for a single device

I'm creating an aplication with openCL 1.2 which is a test for a bigger application. This test sums 1 to each value of a 4x4 matrix with each kernel execution. The idea is to get double buffering to work. I created two kernels that do actually the same thing, they share the same READ_WRITE buffer so each kernel execution can continue where the last one left it, but they differ because they have a different output buffer, allowing to use one of the output buffer with a kernel while reading the data of the other, just like this:

Diagram

The pieces of code I think are relevant or could be problematic are the following, I include queues, buffers and events just in case but I tried changing everything regarding this:

Queues

```
compute_queue = clCreateCommandQueueWithProperties(context, device_id, 0, &err);
data_queue = clCreateCommandQueueWithProperties(context, device_id, 0, &err);
```

Buffer

```
input_Parametros = clCreateBuffer(context, CL_MEM_READ_WRITE |
CL_MEM_COPY_HOST_PTR, sizeof(double) * 5, Parametros, NULL);
input_matA = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,
sizeof(double) * 4, matA_1, NULL); // The 4x4 matrix
output_buffer = clCreateBuffer(context, CL_MEM_WRITE_ONLY , sizeof(double) * 4 *
iteraciones_por_kernel, NULL, NULL);
output_buffer_2 = clCreateBuffer(context, CL_MEM_WRITE_ONLY , sizeof(double) * 4 *
iteraciones_por_kernel, NULL, NULL);
```

Argument set for each kernel

```
    clSetKernelArg(kernel_1, 0, sizeof(cl_mem), &input_matA);
    clSetKernelArg(kernel_1, 1, sizeof(cl_mem), &input_Parametros);
    clSetKernelArg(kernel_1, 3, sizeof(cl_mem), &output_buffer);

    clSetKernelArg(kernel_2, 0, sizeof(cl_mem), &input_matA);
    clSetKernelArg(kernel_2, 1, sizeof(cl_mem), &input_Parametros);
    clSetKernelArg(kernel_2, 3, sizeof(cl_mem), &output_buffer_2);
```

Events

```
cl_event event_1, event_2, event_3, event_4;
```

Kernel and Read enqueue

```
    ////////////////////////////////////////////////////////////////
    // START
    ////////////////////////////////////////////////////////////////

clEnqueueNDRangeKernel(compute_queue, kernel_1, 1, NULL, global, local, 0, 0,
&event_1);

clEnqueueNDRangeKernel(compute_queue, kernel_2, 1, NULL, global, local, 0, 0,
&event_2);

clEnqueueReadBuffer(data_queue, output_buffer, CL_FALSE, 0,
sizeof(double)*4*iteraciones_por_kernel, datos_salida, 1 , &event_1, &event_3);
```

```
    ////////////////////////////////////////////////////////////////
    // ENQUEUE LOOP
    ////////////////////////////////////////////////////////////////

for (int i = 1; i <= (n_iteraciones_int - 2); i++){

        ////////////////////////////////////////////////////////////////
        // LOOP PART 1
        ////////////////////////////////////////////////////////////////

        if (i % 2 != 0){
            clEnqueueNDRangeKernel(compute_queue, kernel_1, 1, NULL, global, local,
1, &event_3, &event_1);

            clEnqueueReadBuffer(data_queue, output_buffer_2, CL_FALSE, 0,
sizeof(double) * 4 * iteraciones_por_kernel,
&datos_salida[i*iteraciones_por_kernel_int*4], 1, &event_2, &event_4);
         }

        ////////////////////////////////////////////////////////////////
        // LOOP PART 2
        ////////////////////////////////////////////////////////////////

        if (i % 2 == 0){

            clEnqueueNDRangeKernel(compute_queue, kernel_2, 1, NULL, global, local,
1, &event_4, &event_2);

            clEnqueueReadBuffer(data_queue, output_buffer, CL_FALSE, 0,
sizeof(double) * 4 * iteraciones_por_kernel,
&datos_salida[i*iteraciones_por_kernel_int * 4], 1, &event_1, &event_3);
        }

    }

    ////////////////////////////////////////////////////////////////
    // END
    ////////////////////////////////////////////////////////////////

clEnqueueReadBuffer(data_queue, output_buffer_2, CL_TRUE, 0, sizeof(double) * 4 *
iteraciones_por_kernel, &datos_salida[(n_iteraciones_int - 1) * 4], 1, &event_2,
0);
```

I just can't get this to work even with everything seeming perfectly fine. The first read gives the expected values, but from then on it's like the kernels don't execute anymore, since i get 0's from the output_buffer_2 and the same values as in the first read form the first output_buffer.

This works perfectly fine with the same kernels and just one queue that does it all with a single data transfer at the end, but I don't want that.

I revised everything and investigated as much as I could, tried every variation I could imagine. This should be easy and possible I think... where is the problem?

I'm using AMD HD7970 as device, Windows 10 and visual studio community 2013 if it is any help.

opencl

asked Mar 16 at 14:34

Pr0nboy
21 4

1 Both modulus checks are against zero. One should be 1 – huseyin tugrul buyukisik Mar 16 at 15:53

1 And latest blocking read expects an event. Be sure it has proper step at Last iteration that returns that event. The odd numbered Last iteration that you forgot to compare against 1 – huseyin tugrul buyukisik Mar 16 at 15:59

1 Lastly, you May need to clflush both queues After loop to initiate both queues while the Last read initiates only its own queue iirc – huseyin tugrul buyukisik Mar 16 at 16:14

1 And creating a New Pair of events for each iteration wouldnt hurt. Since completion of first iteration May make all following iterations start rightaway without waiting When same(already completed) event used – huseyin tugrul buyukisik Mar 16 at 16:24

Thank you very much! this made a lot of sense, now the porgram works as it should! I couldn't find a clear example of this anywhere, so I'll post the code in case it is any help for anyone. I think that creating events for each iteration was the key. – Pr0nboy Mar 17 at 12:49

1 Answer

Thanks to huseyin tugrul buyukisik help, the program worked with the following variations:

Events

```
cl_event event[20]; //adjust this to your needs
```

Kernel and read enqueue

```
//////////////////////////////////////////////////////////////////
// START
//////////////////////////////////////////////////////////////////

clEnqueueNDRangeKernel(compute_queue, kernel_1, 1, NULL, global, local, 0, 0,
&event[0]);


clEnqueueNDRangeKernel(compute_queue, kernel_2, 1, NULL, global, local, 0, 0,
&event[1]);


clEnqueueReadBuffer(data_queue, output_buffer, CL_FALSE, 0,
sizeof(double)*4*iteraciones_por_kernel, datos_salida, 1 , &event[0], &event[2]);

//////////////////////////////////////////////////////////////////
// LOOP
//////////////////////////////////////////////////////////////////

for (int i = 1; i <= (n_iteraciones_int - 2); i++){

        ////////////////////////////////////////////////////////////////
        // LOOP PART 1
        ////////////////////////////////////////////////////////////////

        if (i % 2 == 1){

            clEnqueueNDRangeKernel(compute_queue, kernel_1, 1, NULL, global, local,
1, &event[2+2*(i - 1)], &event[4 + 2 * (i - 1)]);

            clEnqueueReadBuffer(data_queue, output_buffer_2, CL_FALSE, 0,
sizeof(double) * 4 * iteraciones_por_kernel, &datos_salida[i*
(iteraciones_por_kernel_int) * 4], 1, &event[1+2*(i - 1)], &event[3 + 2 * (i -
1)]);

        }

        ////////////////////////////////////////////////////////////////
        // LOOP PART 2
        ////////////////////////////////////////////////////////////////

        if (i % 2 == 0){

            clEnqueueNDRangeKernel(compute_queue, kernel_2, 1, NULL, global, local,
1, &event[3 + 2 * (i - 2)], &event[5 + 2 * (i - 2)]);

            clEnqueueReadBuffer(data_queue, output_buffer, CL_FALSE, 0,
sizeof(double) * 4 * iteraciones_por_kernel, &datos_salida[i*
```

```
(iteraciones_por_kernel_int) * 4], 1, &event[4 + 2 * (i - 2)], &event[6 + 2 * (i -
2)]);
        }

    }

//////////////////////////////////////////////////////////////////
// END
//////////////////////////////////////////////////////////////////

clFlush(compute_queue);
clFlush(data_queue);
clEnqueueReadBuffer(data_queue, output_buffer_2, CL_TRUE, 0, sizeof(double) * 4 *
iteraciones_por_kernel, &datos_salida[(n_iteraciones_int-1)*
(iteraciones_por_kernel_int) * 4], 1, &event[5+2*(n_iteraciones_int-4)], 0);
```

answered Mar 17 at 13:12

Pr0nboy

21 4