OpenCL bevezetés II.

Párhuzamos primitívek

- Map
- Reduce
- Scan
- Histogram
- Compact

Map

```
const size_t dataSize = 1024;
cl_kernel mapKernel = cl.createKernel(clProgram, "map");
float* hData = new float[dataSize];
cl_mem gData = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(float) * dataSize,
                              NULL, NULL);
clEnqueueWriteBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                     sizeof(float) * dataSize, hData,
                     0, NULL, NULL);
clSetKernelArg(mapKernel, 0, sizeof(cl_mem), &gData);
size_t workSize = dataSize;
clEnqueueNDRangeKernel(cl.cqueue(), mapKernel,
                       1, NULL, &workSize, NULL,
                       0, NULL, NULL);
clEnqueueReadBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                    sizeof(float) * dataSize, hData,
                    0, NULL, NULL);
```

Map

```
__kernel
void map(__global float* data)
{
        int id = get_global_id(0);
        float square = data[id] * data[id];
        data[id] = square;
}
```

Reduce

```
const size_t dataSize = 1024;
cl_kernel reduceKernel = cl.createKernel(clProgram, "reduce_global");
float* hData = new float[dataSize];

cl_mem gData = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(float) * dataSize, NULL, NULL);

clEnqueueWriteBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                     sizeof(float) * dataSize, hData, 0, NULL, NULL);

clSetKernelArg(reduceKernel, 0, sizeof(cl_mem), &gData);
size_t workSize = dataSize;
clEnqueueNDRangeKernel(cl.cqueue(), reduceKernel,
                        1, NULL, &workSize, NULL,
                        0, NULL, NULL);

clEnqueueReadBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                    sizeof(float) * dataSize, hData, 0, NULL, NULL);
```

Reduce

```
__kernel
void reduce_global(__global float* data)
{
     int id = get_global_id(0);
     for(unsigned int s = get_global_size(0) / 2; s > 0; s >>= 1)
     {
             if(id < s)
             {
                    data[id] = max(data[id], data[id + s]);
             }
             barrier(CLK_GLOBAL_MEM_FENCE);
     }
}
```

Scan (exclusive)

```
const size_t dataSize = 1024;
cl_kernel reduceKernel = cl.createKernel(clProgram, "reduce_global");

float* hData = new float[dataSize];
cl_mem gData = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(float) * dataSize, NULL, NULL);
clEnqueueWriteBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                     sizeof(float) * dataSize, hData, 0, NULL, NULL);

clSetKernelArg(reduceKernel, 0, sizeof(cl_mem), &gData);
size_t workSize = dataSize;
clEnqueueNDRangeKernel(cl.cqueue(), reduceKernel,
                       1, NULL, &workSize, NULL,
                       0, NULL, NULL);

clEnqueueReadBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                    sizeof(float) * dataSize, hData, 0, NULL, NULL);
```

Scan (exclusive)

```
__kernel
void exscan_global(__global int* data) {
        int id = get_global_id(0);
        data[id] = (id > 0) ? data[id-1] : 0;
        barrier(CLK_GLOBAL_MEM_FENCE);

        for(int s = 1; s < get_global_size(0); s *= 2) {
                int tmp = data[id];
                if(id + s < get_global_size(0)) {
                        data[id + s] += tmp;
                }
                barrier(CLK_GLOBAL_MEM_FENCE);
        }

        if(id == 0) {
                data[0] = 0;
        }
}
```

Histogram

```
const size_t dataSize = 1024;
const size_t histogramSize = 32;
cl_kernel histogramGlobalKernel = cl.createKernel(clProgram, "hist");

int *hData = new int[dataSize];
int *hHist = new int[histogramSize];
memset(hHist, 0, sizeof(int)*histogramSize);
cl_mem gData = clCreateBuffer(cl.context(), CL_MEM_READ_ONLY,
                              sizeof(int) * dataSize, NULL, NULL);
clEnqueueWriteBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                     sizeof(int) * dataSize, hData,
                     0, NULL, NULL);
cl_mem gHist = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(int) * histogramSize,
                              NULL, NULL);
clEnqueueWriteBuffer(cl.cqueue(), gHist, CL_TRUE, 0,
                     sizeof(int) * histogramSize, hHist,
                     0, NULL, NULL);
```

Histogram

```
clSetKernelArg(histogramGlobalKernel, 0, sizeof(cl_mem), &gData);
clSetKernelArg(histogramGlobalKernel, 1, sizeof(cl_mem), &gHist);

const size_t workSize = dataSize;
clEnqueueNDRangeKernel(cl.cqueue(), histogramGlobalKernel,
                       1, NULL, &workSize, NULL,
                       0, NULL, NULL);

clEnqueueReadBuffer(cl.cqueue(), gHist, CL_TRUE, 0,
                    sizeof(int) * histogramSize, hHist,
                    0, NULL, NULL);
```

Histogram

```
__kernel
void histogram_global(__global int* data, __global int* histogram)
{
        int id = get_global_id(0);
        histogram[data[id]] += 1;
}
```

- Nincs szinkronizáció!
- Hibás!

Histogram

```
__kernel
void histogram_global(__global int* data, __global int* histogram)
{
        int id = get_global_id(0);
        atomic_add(&histogram[data[id]], 1);
}
```

- Az atomikus műveletek szinkronizáltak!

Histogram (lokális)

```
clSetKernelArg(histogramLocalKernel, 0, sizeof(cl_mem), &gData);
clSetKernelArg(histogramLocalKernel, 1, sizeof(cl_mem), &gHist);

clSetKernelArg(histogramLocalKernel, 2, sizeof(int) * histogramSize,
                                                              NULL);

clSetKernelArg(histogramLocalKernel, 3, sizeof(int), &histogramSize);
```

- Lokális memória allokálása host oldalról

Histogram

```
__kernel
void histogram_local(__global int* data, __global int* histogram,
                     __local int* lhistogram,
                     const int histogramSize) {
        int id = get_global_id(0);
        int lid = get_local_id(0);
        if(lid < histogramSize) {
                lhistogram[lid] = 0;
        }
        barrier(CLK_LOCAL_MEM_FENCE);

        atomic_add(&lhistogram[data[id]], 1);

        barrier(CLK_LOCAL_MEM_FENCE);
        if(lid < histogramSize){
                histogram[lid] = lhistogram[lid];
        }
}
```

Compact

```
const size_t dataSize = 1024;
cl_kernel predicateKernel = cl.createKernel(clProgram, "c_pred");
cl_kernel exscanKernel = cl.createKernel(clProgram, "c_exscan");
cl_kernel compactKernel = cl.createKernel(clProgram, "c_compact");

int* hData = new int[dataSize];
cl_mem gData = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(float) * dataSize,
                              NULL, NULL);
clEnqueueWriteBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                     sizeof(float) * dataSize, hData,
                     0, NULL, NULL);
cl_mem gPred = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                              sizeof(int) * dataSize,
                              NULL, NULL);
cl_mem gPrefSum = clCreateBuffer(cl.context(), CL_MEM_READ_WRITE,
                                 sizeof(int) * dataSize,
                                 NULL, NULL);
```

Compact

```
size_t workSize = dataSize;

// Predicate
clSetKernelArg(predicateKernel, 0, sizeof(cl_mem), &gData);
clSetKernelArg(predicateKernel, 1, sizeof(cl_mem), &gPred);
clEnqueueNDRangeKernel(cl.cqueue(), predicateKernel,
                           1, NULL, &workSize, NULL,
                           0, NULL, NULL);

// Exclusive scan on predicate buffer
clSetKernelArg(exscanKernel, 0, sizeof(cl_mem), &gPred);
clSetKernelArg(exscanKernel, 1, sizeof(cl_mem), &gPrefSum);
clEnqueueNDRangeKernel(cl.cqueue(), exscanKernel,
                       1, NULL, &workSize, NULL,
                       0, NULL, NULL);
```

Compact

```
// Compact
clSetKernelArg(compactKernel, 0, sizeof(cl_mem), &gData);
clSetKernelArg(compactKernel, 1, sizeof(cl_mem), &gPred);
clSetKernelArg(compactKernel, 2, sizeof(cl_mem), &gPrefSum);
clEnqueueNDRangeKernel(cl.cqueue(), compactKernel,
                       1, NULL, &workSize, NULL,
                       0, NULL, NULL);

clEnqueueReadBuffer(cl.cqueue(), gData, CL_TRUE, 0,
                    sizeof(float) * dataSize, hData,
                    0, NULL, NULL);
```

Compact

```
__kernel
void c_pred(__global int* data, __global int* pred)
{
      int id = get_global_id(0);
      int predVal = data[id] < 50 ? 1 : 0;
      pred[id] = predVal;
}
```

Compact

```
__kernel
void c_exscan(__global int* pred, __global int* prefSum) {
        int id = get_global_id(0);
        prefSum[id] = (id > 0) ? pred[id-1] : 0;
        barrier(CLK_GLOBAL_MEM_FENCE);

        for(int s = 1; s < get_global_size(0); s *= 2) {
                int tmp = prefSum[id];
                if(id + s < get_global_size(0)) {
                        prefSum[id + s] += tmp;
                }
                barrier(CLK_GLOBAL_MEM_FENCE);
        }

        if(id == 0) {
                prefSum[0] = 0;
        }
}
```

Compact

```
__kernel
void c_compact(__global int* data,
               __global int* pred,
               __global int* prefSum)
{
        int id = get_global_id(0);
        int val = data[id];
        barrier(CLK_GLOBAL_MEM_FENCE);
        if(pred[id] == 1)
        {
                data[prefSum[id]]=val;
        }
}
```