Performing Reductions in OpenCL

Recall the OpenCL Memory Model

Here's the Problem We're Trying to Solve

Reduction Takes Place in a Single Work-Group

Like the first.cpp demo program, we are piecewise multiplying two arrays. Unlike the first demo program, we want to then add up all the products and return the sum.

\[ A \times B \rightarrow \text{prods} \]
\[ \sum \text{prods} \rightarrow C \]

After the array multiplication, we want each work-group to sum the products within that work-group, then return them to the host in an array for final summing.

To do this, we will not put the products into a large global device array, but into a `prods[]` array that is shared within its work-group.

If we had 8 work-items in a work-group, we would like the threads in each work-group to execute the following instructions . . .

... but in a more general way than writing them all out by hand.
Here’s What You Would Change in your Host Program

```c
size_t numWorkGroups = NUM_ELEMENTS / LOCAL_SIZE;

float * hA = new float [ NUM_ELEMENTS ];
float * hB = new float [ NUM_ELEMENTS ];
float * hC = new float [ numWorkGroups ];
size_t abSize = NUM_ELEMENTS * sizeof(float);
size_t cSize = numWorkGroups * sizeof(float);
cl_mem dA = clCreateBuffer( context, CL_MEM_READ_ONLY, abSize, NULL, &status );
cl_mem dB = clCreateBuffer( context, CL_MEM_READ_ONLY, abSize, NULL, &status );
cl_mem dC = clCreateBuffer( context, CL_MEM_WRITE_ONLY, cSize, NULL, &status );
status = clEnqueueWriteBuffer( cmdQueue, dA, CL_FALSE, 0, abSize, hA, 0, NULL, NULL );
status = clEnqueueWriteBuffer( cmdQueue, dB, CL_FALSE, 0, abSize, hB, 0, NULL, NULL );
cl_kernel kernel = clCreateKernel( program, "ArrayMultReduce", &status );
status = clSetKernelArg( kernel, 0, sizeof(cl_mem), &dA );
status = clSetKernelArg( kernel, 1, sizeof(cl_mem), &dB );
status = clSetKernelArg( kernel, 2, LOCAL_SIZE * sizeof(float), NULL );
// local "prods" array is dimensioned the size of each work-group
status = clSetKernelArg( kernel, 3, sizeof(cl_mem), &dC );
```

This NULL is how you tell OpenCL that this is a local (shared) array, not a global array.

"cl_mem" is a GPU buffer memory address.

The Arguments to the Kernel

```c
kernel void ArrayMultReduce( global const float *dA, global const float *dB, local float *prods, global float *dC ) {
    int gid = get_global_id( 0 );  // 0 .. total_array_size-1
    int numItems = get_local_size( 0 );  // # work-items per work-group
    int tnum = get_local_id( 0 );      // thread (i.e., work-item) number in this work-group
    // 0 .. numItems-1
    int wgNum = get_group_id( 0 );     // which work-group number this is in
    prods[ tnum ] =  dA[ gid ] * dB[ gid ];   // multiply the two arrays together
    // now add them up – come up with one sum per work-group
    // it is a big performance benefit to do it here while "prods" is still available – and is local
    // it would be a performance hit to pass "prods" back to the host then bring it back to the device for reduction
    status = clSetKernelArg( kernel, 0, sizeof(cl_mem), &dA );
    status = clSetKernelArg( kernel, 1, sizeof(cl_mem), &dB );
    status = clSetKernelArg( kernel, 2, LOCAL_SIZE * sizeof(float), NULL );
    status = clSetKernelArg( kernel, 3, sizeof(cl_mem), &dC );
}
```

Reduction Takes Place Within a Single Work-Group

Each work-item is run by a single thread

<table>
<thead>
<tr>
<th>Thread #0:</th>
<th>Thread #1:</th>
<th>Thread #2:</th>
<th>Thread #3:</th>
</tr>
</thead>
</table>

Offset = 4; mask = 7;

<table>
<thead>
<tr>
<th>Thread #4:</th>
<th>Thread #5:</th>
<th>Thread #6:</th>
<th>Thread #7:</th>
</tr>
</thead>
</table>

Offset = 2; mask = 3;

A work-group consisting of numItems work-items can be reduced to a sum in log2(numItems) steps. In this example, numItems=8.

The reduction begins with the individual products in prods[0] .. prods[7].

The final sum will end up in prods[0], which will then be copied into dC[wgNum].

A Review of Bitmasks

Remember Truth Tables?

<table>
<thead>
<tr>
<th>F</th>
<th>F</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Or, with Bits:

0 & 0 = 0
1 & 1 = 1
0 & 1 = 0
1 & 0 = 1

Or, with Multiple Bits:

000 & 001 = 000
010 & 011 = 010
011 & 011 = 011
100 & 101 = 100
101 & 101 = 101
000 & 001 = 000
011 & 011 = 011
100 & 101 = 100
101 & 101 = 101

If it’s been a long time since you have looked at bitmask operators (or never!), here is a good review reference: https://en.wikipedia.org/wiki/Bitwise_operations_in_C
Reduction Takes Place in a Single Work-Group
Each work-item is run by a single thread

Thread #0:
prod[0] += prod[1];
Thread #2:
prod[2] += prod[3];
Thread #4:
prod[4] += prod[5];
Thread #6:
prod[6] += prod[7];

offset = 1;
mask = 1;
numItems = 8;

Anding bits

Σ prod → C

Reduction Performance
Work-Group Size = 32

And, Finally, in your Host Program

Wait( cmdQueue );
double time0 = omp_get_wtime( );
status = clEnqueueNDRangeKernel(cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL);
PrintCLError(status, "clEnqueueNDRangeKernel failed: ");
Wait( cmdQueue );
double time1 = omp_get_wtime( );
status = clEnqueueReadBuffer(cmdQueue, dC, CL_TRUE, 0, numWorkGroups*sizeof(float), hC, 0, NULL, NULL);
PrintCLError(status, "clEnqueueReadBuffer failed: ");
Wait(cmdQueue);

float sum = 0.;
for ( int i = 0; i < numWorkGroups; i++ )
{
    sum += hC[i];
}

GigaNumbers Multiplied and Reduced Per Second

Array Size (MegaNumbers)