Performing Reductions in OpenCL

Here's the Problem We are Trying to Solve

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 \text{prods[]} array that is local to each work-group.

Reduction Takes Place in a Single 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 . . .

\begin{align*}
\text{Thread #0:} & \quad \text{prods}[0] += \text{prods}[1]; \\
\text{Thread #2:} & \quad \text{prods}[2] += \text{prods}[3]; \\
\text{Thread #4:} & \quad \text{prods}[4] += \text{prods}[5]; \\
\text{Thread #6:} & \quad \text{prods}[6] += \text{prods}[7]; \\
\end{align*}

. . . 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);

kernel void ArrayMultReduce( global const float *dA, global const float *dB, local float *prods, local float *dC )
{
    int gid = get_global_id(0);     // 0 .. total_array_size-1
    int numItems = get_local_id(0); // # work-items per work-group

    // all threads execute this code simultaneously:
    // for( int offset = 0; offset < numItems; offset *= 2 )
    //   mask = 2*offset - 1;
    barrier(CLK_LOCAL_MEM_FENCE); // wait for completion
    if( (tNum & mask) == 0 )
    {
        int tNum = get_local_id(0);   // thread number
        int wgNum = get_group_id(0);  // work-group number
        int offset = 2*wgNum;         // thread offset
        prods[tNum] += dA[gid] * dB[gid]; // multiply the two arrays together
    } // now add them up – come up with one sum per work-group
}
```

Reduction Takes Place Within a Single Work-Group

Each work-item is run by a single thread.

Thread #0:
prods[0] += prods[1];

Thread #2:
prods[2] += prods[3];

Thread #4:
prods[4] += prods[5];

Thread #6:
prods[6] += prods[7];

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].

Reduction Takes Place in a Single Work-Group

Each work-item is run by a single thread.

Thread #0:
prods[0] += prods[1];

Thread #2:
prods[2] += prods[3];

Thread #4:
prods[4] += prods[5];

Thread #6:
prods[6] += prods[7];

kernel void ArrayMultReduce( )
{
    int gid = get_global_id(0); // 0 .. total_array_size-1
    int numItems = get_local_id(0); // # work-items per work-group
    int tNum = get_local_id(0); // thread number
    int wgNum = get_group_id(0); // work-group number
    int offset = 2*wgNum; // thread offset
    prods[tNum] += dA[gid] * dB[gid]; // multiply the two arrays together
}
```
And, Finally, in your Host Program

```c
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];
}
```

Reduction Performance
Work-Group Size = 32