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

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.

Here’s What You Would Change in your Host Program

Here are the following instructions...

\[ \text{Thread #0: } \text{prods}[0] += \text{prods}[1]; \]
\[ \text{Thread #2: } \text{prods}[2] += \text{prods}[3]; \]
\[ \text{Thread #4: } \text{prods}[2] += \text{prods}[4]; \]
\[ \text{Thread #6: } \text{prods}[4] += \text{prods}[5]; \]

Recall the OpenCL Memory Model

Reduction Takes Place in a Single Work-Group

The Arguments to the Kernel

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

Thread #0:
prods[0] += prods[2];
Thread #4:
prods[4] += prods[6];
Thread #0:
prods[0] += prods[4];

A work-group consisting of numItems work-items can be reduced to a sum in log(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?
F & F = F
T & F = F
T & T = T
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
Or, with Bits:
000 & 011 = 000
001 & 011 = 001
010 & 011 = 010
011 & 011 = 011
100 & 011 = 000
101 & 011 = 001

Or, with Multiple Bits:
00000000 & 01101010 = 00000000
00000000 & 01111111 = 00000000

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];

Thread #0:
prods[0] += prods[2];
Thread #4:
prods[4] += prods[6];
Thread #0:
prods[0] += prods[4];

A work-group consisting of numItems work-items can be reduced to a sum in log(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].

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];
}

Reduction Performance
Work-Group Size = 32

Array Size (MegaNumbers) vs. Operations Multiplied and Reduced Per Second

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];

Thread #0:
prods[0] += prods[2];
Thread #4:
prods[4] += prods[6];
Thread #0:
prods[0] += prods[4];