

Performing Reductions in OpenCL



Oregon State University
Mike Bailey

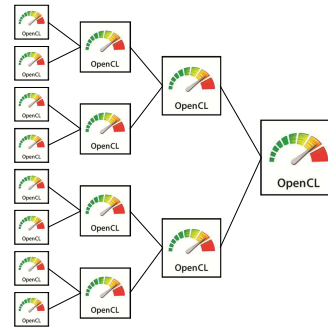
mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State University
Computer Graphics

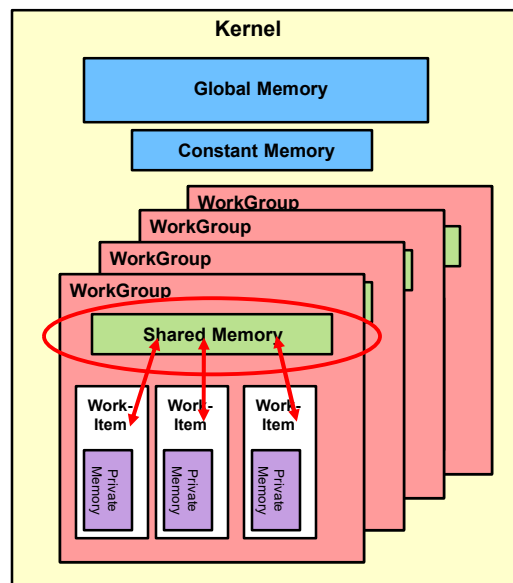


opengl.reduction.pptx

mjb - June 14, 2021

1

Recall the OpenCL Memory Model



Oregon State University
Computer Graphics

mjb - June 14, 2021

2

Here's the Problem We are Trying to Solve

3

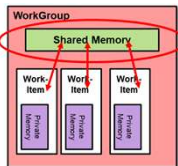
Like the *first.cpp* demo program, we are piecewise multiplying two arrays. Unlike the first demo program, we want to then add up all the products and return the sum.

$$A * B \rightarrow \text{prods}$$

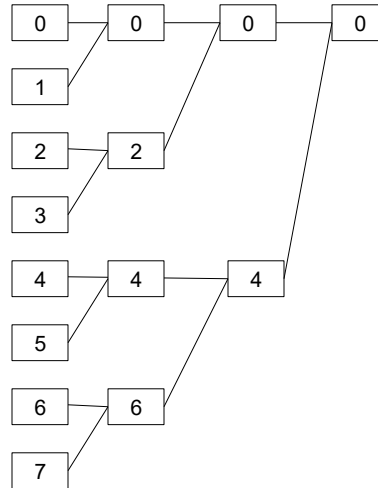
$$\sum \text{prods} \rightarrow C$$

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.



numItems = 8;



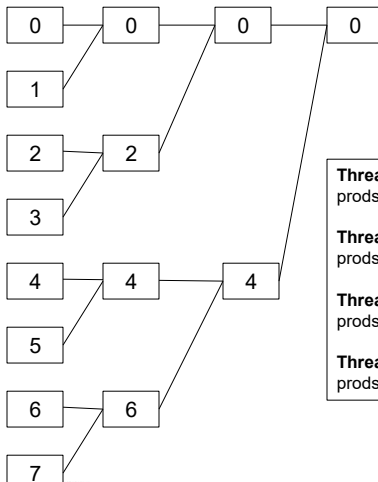
mjb - June 14, 2021

3

Reduction Takes Place in a Single Work-Group

4

numItems = 8;



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

Thread #0: <code>prods[0] += prods[1];</code>	Thread #0: <code>prods[0] += prods[2];</code>	Thread #0: <code>prods[0] += prods[4];</code>
Thread #2: <code>prods[2] += prods[3];</code>	Thread #4: <code>prods[4] += prods[6];</code>	
Thread #4: <code>prods[4] += prods[5];</code>		
Thread #6: <code>prods[6] += prods[7];</code>		

. . . but in a more general way than writing them all out by hand.



mjb - June 14, 2021

4

Here's What You Would Change in your Host Program

5

```
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

Computer Graphics

mjb - June 14, 2021

5

The Arguments to the Kernel

6

```
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 - one per work-item
status = clSetKernelArg( kernel, 3, sizeof(cl_mem), &dC );
```

```
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
}
```

$A * B \rightarrow \text{prods}$

 Oregon State
University
Computer Graphics

mjb - June 14, 2021

6

Reduction Takes Place Within a Single Work-Group Each work-item is run by a single thread

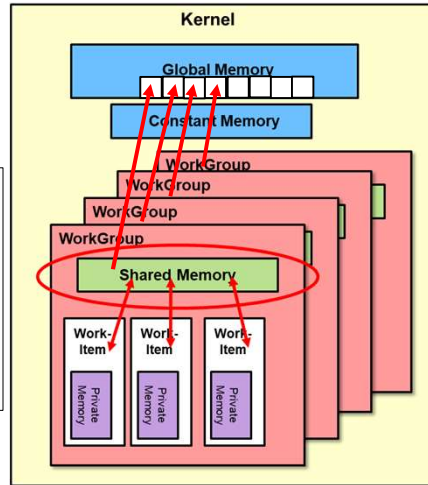
7

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

A work-group consisting of *numItems* work-items can be reduced to a sum in $\log_2(\text{numItems})$ 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].



7

A Review of Bitmasks

8

Remember *Truth Tables*?

F	F	T	T
& F	& T	& F	& T
= F	= F	= F	= T

Or, with Bits:

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

Or, with Multiple Bits:

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

8

9


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]; offset = 1; mask = 1;	Thread #0: prods[0] += prods[2]; Thread #4: prods[4] += prods[6]; offset = 2; mask = 3;	Thread #0: prods[0] += prods[4]; offset = 4; mask = 7;
--	--	--

kernel void
ArrayMultReduce(...)
 int gid = get_global_id(0);
 int numItems = get_local_size(0);
 int tnum = get_local_id(0); // thread number
 int wgNum = get_group_id(0); // work-group number

 prods[tnum] = dA[gid] * dB[gid];

numItems = 8;

$\sum \text{prods} \rightarrow C$


Oregon State
University
Computer Graphics

// all threads execute this code simultaneously:
 for(int offset = 1; offset < numItems; offset *= 2)
 {
 int mask = 2*offset - 1;
 barrier(CLK_LOCAL_MEM_FENCE); // wait for all threads to get here
 if((tnum & mask) == 0) // bit-by-bit and'ing tells us which
 { // threads need to do work now
 prods[tnum] += prods[tnum + offset];
 }
 }

 barrier(CLK_LOCAL_MEM_FENCE);
 if(tnum == 0)
 dC[wgNum] = prods[0];

Aanding bits

9

10

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 ];
}
  
```



Oregon State
University
Computer Graphics

mjb - June 14, 2021

10

