Slide 1

Performing Reductions in OpenCL

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics
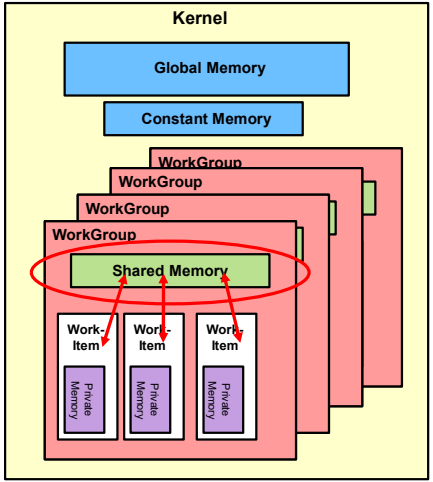
opencl.reduction.pptx

mjb – June 14, 2021

Slide 2

Recall the OpenCL Memory Model

Oregon State University
Computer Graphics

mjb – June 14, 2021

Slide 3

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.

$$A * B \rightarrow prods$$
$$\Sigma \, prods \rightarrow C$$

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;

Oregon State University
Computer Graphics

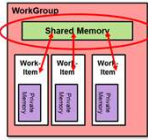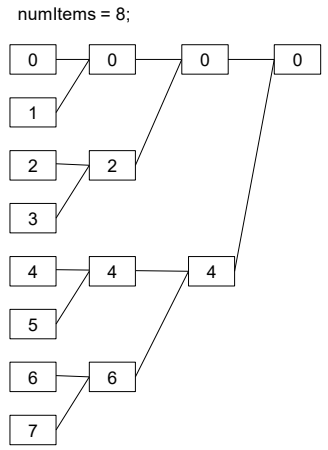mjb – June 14, 2021

Slide 4
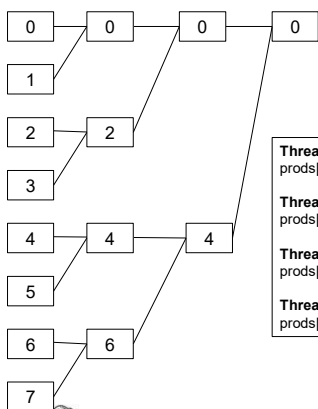
Reduction Takes Place in a Single Work-Group

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

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

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

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

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

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

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

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

Oregon State University
Computer Graphics

mjb – June 14, 2021

Slide 5

Here's What You Would Change in your Host Program

```
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 );
```

A * B → prods
Σ prods → C

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

Computer Graphics

mjb – June 14, 2021

Slide 6

The Arguments to the Kernel

```
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 → prods

Oregon State University
Computer Graphics

mjb – June 14, 2021

Slide 7

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];

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;

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

Kernel
Global Memory
Constant Memory
WorkGroup
WorkGroup
WorkGroup
WorkGroup
Shared Memory
Work-Item   Work-Item   Work-Item
Private Memory   Private Memory   Private Memory

Oregon State University
Computer Graphics

mjb – June 14, 2021

Slide 8

A Review of Bitmasks

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

Oregon State University
Computer Graphics

mjb – June 14, 2021

Slide 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;

Anding bits →

$\sum$ prods → C

```
// 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 ];
```

Oregon State
University
Computer Graphics

Slide 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, "clEnqueueReadBufferl failed: " );
Wait( cmdQueue );


float sum = 0.;
for( int i = 0; i < numWorkgroups; i++ )
{
        sum += hC[ i ];
}
```
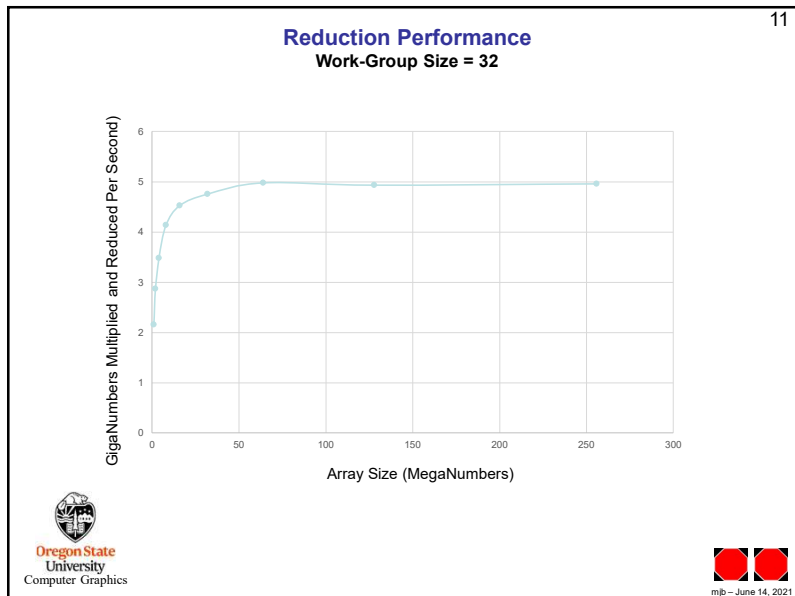
Oregon State
University
Computer Graphics

mjb – June 14, 2021

Slide 11

Reduction Performance
Work-Group Size = 32

(y-axis: GigaNumbers Multiplied and Reduced Per Second, 0–6)
(x-axis: Array Size (MegaNumbers), 0–300)

Oregon State
University
Computer Graphics

mjb – June 14, 2021