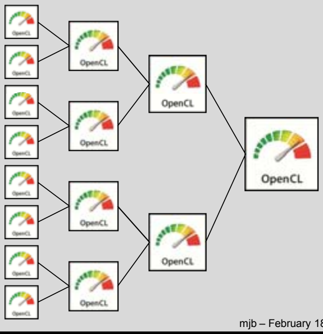
Performing Reductions in OpenCL

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University

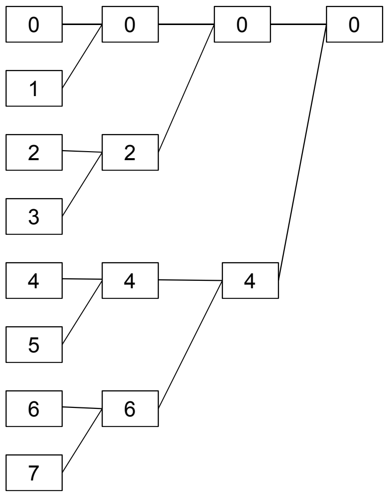


Here's the Problem We are Trying to Solve

Like the first 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.

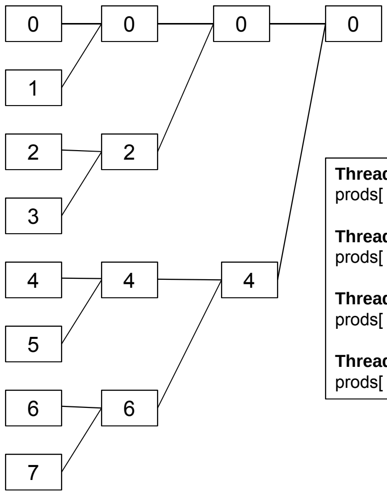
$A * B \rightarrow \text{prods}$
$\Sigma\ \text{prods} \rightarrow C$

numItems = 8;



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 . . .

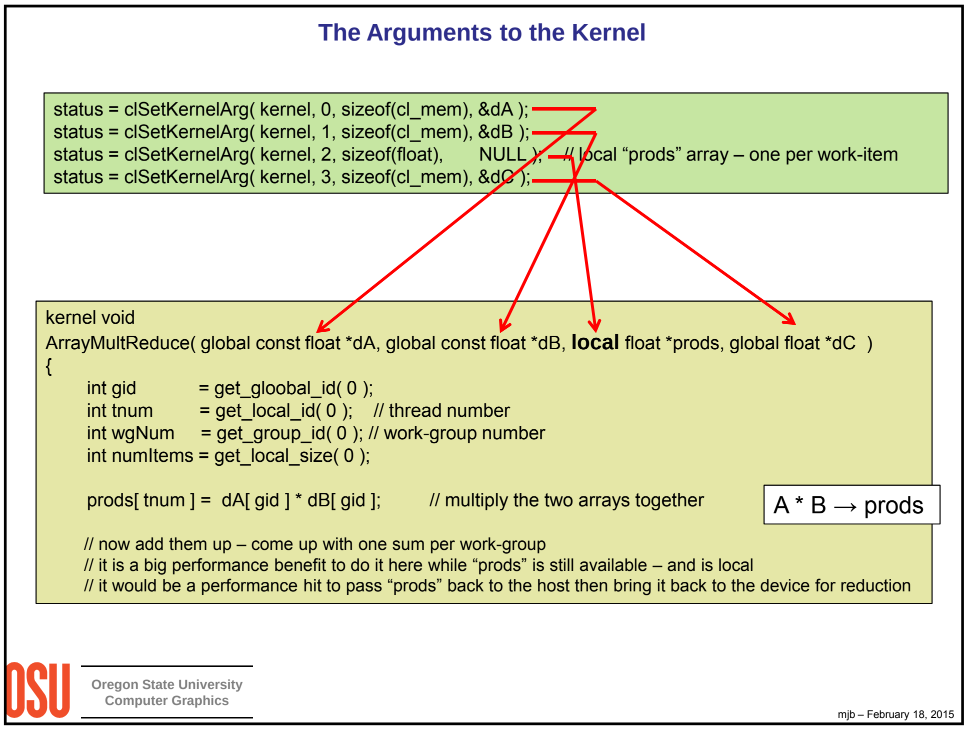
Thread #0: prods[0] += prods[1];	**Thread #0:** prods[0] += prods[2];	**Thread #0:** prods[0] += prods[4];
Thread #2: prods[2] += prods[3];		
Thread #4: prods[4] += prods[5];	**Thread #4:** prods[4] += prods[6];	
Thread #6: prods[6] += prods[7];		

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

Here's What You Would Change in your Host Program

A * B → prods
Σ prods → 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, sizeof(float),       NULL );    // local "prods" array – one per work-item
status = clSetKernelArg( kernel, 3, sizeof(cl_mem), &dC );
```



The Arguments to the Kernel

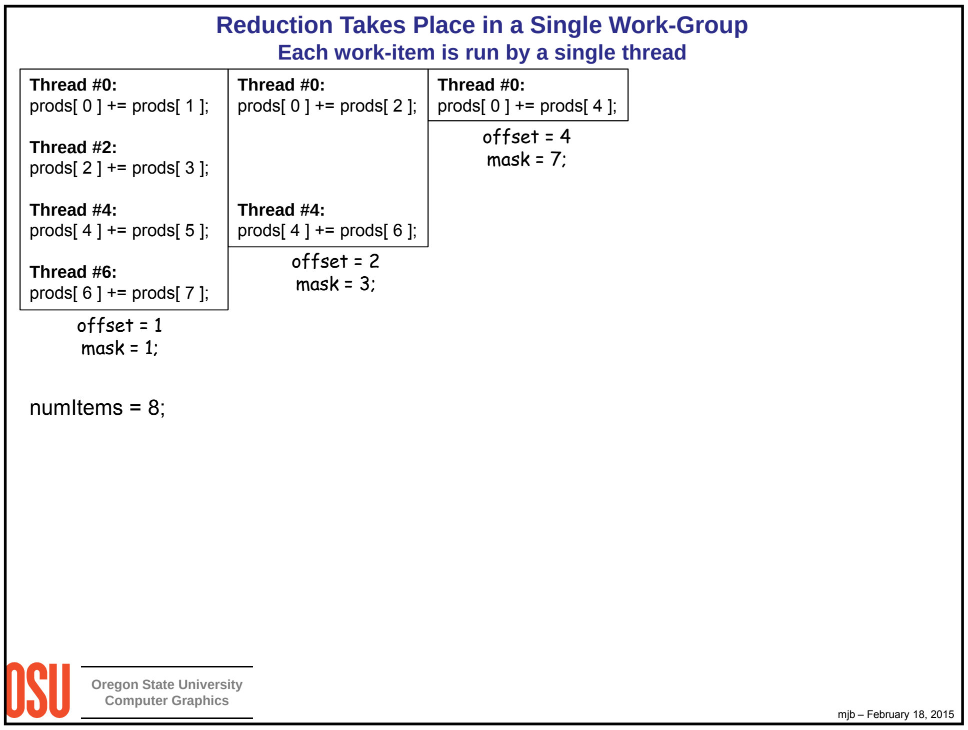
```
status = clSetKernelArg( kernel, 0, sizeof(cl_mem), &dA );
status = clSetKernelArg( kernel, 1, sizeof(cl_mem), &dB );
status = clSetKernelArg( kernel, 2, sizeof(float),       NULL );     // local "prods" array – one per work-item
status = clSetKernelArg( kernel, 3, sizeof(cl_mem), &dC );
```

```
kernel void
ArrayMultReduce( global const float *dA, global const float *dB, local float *prods, global float *dC  )
{
        int gid           = get_gloobal_id( 0 );
        int tnum          = get_local_id( 0 );     // thread number
        int wgNum         = get_group_id( 0 ); // work-group number
        int numItems = get_local_size( 0 );

        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
```

$A * B \rightarrow$ prods

Oregon State University
Computer Graphics

mjb – February 18, 2015



Reduction Takes Place in a Single Work-Group

Each work-item is run by a single thread

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];		

offset = 1
mask = 1;
	offset = 2
	mask = 3;
		offset = 4
		mask = 7;

numItems = 8;

Oregon State University
Computer Graphics

mjb – February 18, 2015

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;

numItems = 8;

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

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

```
// all threads execute this code simultaneously:
 for( int offset = 1; offset < numItems; offset *= 2 )
{
        int mask = 2*offset - 1;
        barrier( CLK_LOCAL_MEM_FENCE );
        if(  ( tnum & mask ) == 0 )
        {
                prods[ tnum ] += prods[ tnum + offset ];
        }
}

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

Σ prods → C

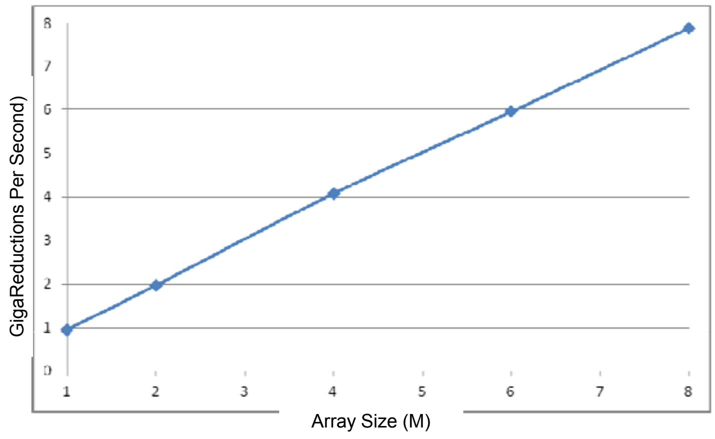
Oregon State Uni
Computer Grap

And, Finally, in your Host Program

```
status = clEnqueueReadBuffer( cmdQueue, dC, CL_TRUE, 0, cSize, hC, 0, NULL, NULL );

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

Oregon State University
Computer Graphics

Reduction Performance:
Work-Group Size = 32



Oregon State University
Computer Graphics

mjb – February 18, 2015