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

A * B → prods
Σ prods → 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.

Here's What You Would Change in your Host Program

size_t numWorkGroups = NUM_ELEMENTS / LOCAL_SIZE;
    
    float * prods, global float * dC

    A * B → prods
    Σ prods → C

    This is how you tell OpenCL that this is a local array, not a global array.

Recall the OpenCL Memory Model

Reduction Takes Place in a Single Work-Group

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(cl_mem), &dC );
status = clSetKernelArg( kernel, 3, sizeof(cl_mem), NULL );
status = clSetKernelArg( kernel, 4, sizeof(cl_mem), NULL );
status = clSetKernelArg( kernel, 5, sizeof(cl_mem), NULL );
status = clSetKernelArg( kernel, 6, sizeof(cl_mem), NULL );

Kernel void ArrayMultReduce

{ int gid = get_global_id( 0 );
  if( thread(x) < numWorkGroups )
    if( work-item number is in prods )
      multiply the two arrays together . . .

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

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

  Thread #2:
    prods[2] += prods[3];
    prods[4] += prods[5];

  Thread #3:
    prods[3] += prods[4];

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

  }
Reduction Takes Place Within a Single Work-Group

<table>
<thead>
<tr>
<th>Thread #0</th>
<th>prods[0] += prods[1]</th>
</tr>
</thead>
</table>

Reduction Takes Place in a Single Work-Group

<table>
<thead>
<tr>
<th>Thread #0</th>
<th>prods[0] += prods[1]</th>
</tr>
</thead>
</table>

And, Finally, in your Host Program

```c
wait(cmdQueue); //wait for completion
for( int offset = 1; offset < numItems; offset *= 2 ) {
    int mask = 2*offset - 1;
    barrier( CLK_LOCAL_MEM_FENCE ); // wait for completion
    if(  (tnum & mask) == 0 ) {
        prods[tnum] += prods[tnum + offset];
    }
}
wait(cmdQueue);
if( tnum == 0 )
    dC[wgNum] = prods[0];
```

Reduction Performance

Work-Group Size = 32

<table>
<thead>
<tr>
<th>Array Size (MegaNumbers)</th>
<th>GigaNumbers Multiplied and Reduced Per Second</th>
<th>Reduction Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>numItems = 8;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>