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.

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

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 \(\text{prods}[\cdot]\) array that is shared within its work-group.

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

Here's What You Would Change in your Host Program

Here's the Arguments to the Kernel

The Arguments to the Kernel

Reduction Takes Place in a Single Work-Group

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

---

**Here's What You Would Change in your Host Program**

```cpp
int numItems = 8;
float *hA = new float[numItems];
float *hB = new float[numItems];
float *hC = new float[numItems];
size_t abSize = numItems * sizeof(float);
size_t cSize = numItems * sizeof(float);
size_t numWorkGroups = numItems / LOCAL_SIZE;
```

**The Arguments to the Kernel**

```cpp
kernel void ArrayMultReduce( global const float *dA, global const float *dB, global float *dC ) {
    int gid = get_global_id(0); // 0 .. total array size-1
    int numWorkGroups = get_work_group_size(); // work-items per work-group
    int threadNumber = get_local_id(0); // thread number in the local group
    int wgNum = get_group_id(0); // which work-group number this is in
    int tnum = threadNumber * numWorkGroups; // a work-item's local thread number
    int gid = get_global_id(0); // global item number
    if (threadNumber == 0) {
        // if thread is the first thread in the local group
        // make a (shared) array, not a global array
        float *prods = (float*)malloc(sizeof(float) * numWorkGroups);
        for (int i = 0; i < numWorkGroups; i++)
            prods[i] = 0;
    }
    int numWorkGroups = get_work_group_size(); // work-items per work-group
    for (int i = 0; i < numWorkGroups; i++)
        prods[i] = dA[i] * dB[i]; // multiply the two arrays together
    prods[0] += prods[1]; // add the first prods element to the next one
    for (int i = 0; i < numWorkGroups - 1; i++)
        prods[i] += prods[i + 1]; // add the following prods elements
    if (threadNumber == 0) {
        // if thread is the first thread in the local group
        // add up a sum per work-group
        dC[gid] = 0;
        for (int i = 0; i < numWorkGroups; i++)
            dC[gid] += prods[i];
    }
}
```
Reduction Takes Place Within a Single Work-Group

Each work-item is run by a single thread

- Thread #0: prods[0] += prods[1]

Reduction Takes Place in a Single Work-Group

Each work-item is run by a single thread

- Thread #0: prods[0] += prods[2]
- Thread #0: prods[0] += prods[4]

And, Finally, in your Host Program

```c
Wait( cmdQueue );
double time0 = omp_get_wtime( );
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );
PrintCLError( status, "clEnqueueNDRangeKernel failed: \n" );
Wait( cmdQueue );

status = clEnqueueReadBuffer( cmdQueue, dC, CL_TRUE, 0, numWorkGroups*sizeof(float), hC, 0, NULL, NULL );
PrintCLError( status, "clEnqueueReadBuffer failed: \n" );
Wait( cmdQueue );
float sum = 0;
for( int i = 0; i < numWorkGroups; i++ )
{    
    sum += hC[ i ];
}
```

Reduction Performance

Work-Group Size = 32

![Graph showing reduction performance](image-url)