Here’s What You Would Change in your Host Program

```c
sizeof(cl_mem), &dC);
```

```c
status = clSetKernelArg( kernel, 0, sizeof(cl_mem), &dA );
```

```c
status = clSetKernelArg( kernel, 1, sizeof(cl_mem),                     &dB );
```

```c
status = clSetKernelArg( kernel, 2, LOCAL_SIZE * sizeof(float),         &status );
```

Here’s the Problem We’re 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.

\[
A \cdot B \rightarrow prods \\
\sum prods \rightarrow C
\]

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.

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...

```c
prods[t] = dA[gid] * dB[gid]; // multiply the two arrays together
```

```c
// now add them up -- come up with sum per work-group
```

```c
// 0 .. numItems - 1
```

```c
Thread 0:
prods[0] += prods[1];
```

```c
Thread 2:
prods[0] += prods[4];
```

```c
Thread 4:
prods[0] += prods[6];
```

```c
Thread 6:
prods[0] += prods[7];
```

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

The Arguments to the Kernel

```c
ArrayMultReduce( global const float *dA, global const float *dB, local float *prods, global float *dC )
```

```c
A \cdot B \rightarrow prods \\
\sum prods \rightarrow C
```

Performing Reductions in OpenCL

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Reduction Takes Place Within a Single Work-Group

Each work-item is run by a single thread

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

Reduction Performance

Work-Group Size = 32

Reduction Takes Place in a Single Work-Group

Each work-item is run by a single thread

Anding bits

\[ \sum \text{prods} \rightarrow C \]

Remember Truth Tables?

\[
\begin{array}{c|c|c|c|c|c}
F & F & F & T & T & T \\
T & F & T & T & T & T \\
T & T & T & T & T & T \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Or, with Bits:

\[
\begin{array}{c|c}
0 & 0 \\
A & 0 \\
& 1 \\
E & 0 \\
& 1 \\
\end{array}
\]

Or, with Multiple Bits:

\[
\begin{array}{c|c}
000 & 011 \\
001 & 011 \\
010 & 011 \\
011 & 011 \\
100 & 011 \\
101 & 011 \\
\end{array}
\]

If it's been a long time since you have looked at bitmask operators (or never!), here is a good review reference: https://en.wikipedia.org/wiki/Bitwise_operations_in_C

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];
}

...