OpenCL Matrix Multiplication

A matrix is a 2D array of numbers, arranged in rows that go across and columns that go down:

\[
\begin{pmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{pmatrix}
\]

4 columns
3 rows

Matrix sizes are termed "rows x columns", so this is a 3x4 matrix

Square Matrices

A square matrix has the same number of rows and columns

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{pmatrix}
\]

3 columns
3 rows

This is a 3x3 matrix

Matrix Multiplication

The basic operation of matrix multiplication is to pair-wise multiply a single row by a single column

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{pmatrix}
\begin{pmatrix}
4 \\
5 \\
6
\end{pmatrix}
\]

\[
\begin{pmatrix}
(1*4) + (2*5) + (3*6)
\end{pmatrix}
\]

1x3 3x1 1x1

Matrix Multiplication in Software

Here's how to remember how to do it:

1. \( C = A \times B \)

2. \([i \times J] = [1 \times K] \times [K \times J]\)

\[
C[i][j] = A[i][k] \times B[k][j];
\]
Matrix Multiplication in CPU Software

for( int i = 0; i < numArows; i++ )
{
    for( int j = 0; j < numBcols; j++ )
    {
        C[i][j] = 0.;
        for( int k = 0; k < numAcols; k++ )
        {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}

Note: numAcols must == numBrows!

Note that saying:
C[i][j] = 0.;
for( int k = 0; k < numAcols; k++ )
{
    C[i][j] += A[i][k] * B[k][j];
}

Is like saying:
size_t globalWorkSize[3] = { MATW, MATW, 1 };  
size_t localWorkSize[3] = { LOCALSIZE, LOCALSIZE, 1 };  
Wait( CmdQueue );  
double time0 = omp_get_wtime();  
status = clEnqueueNDRangeKernel( CmdQueue, Kernel, 1, NULL,  
globalWorkSize, localWorkSize, 0, NULL, NULL );  
Wait( CmdQueue );  
double time1 = omp_get_wtime();  

Executing the Kernel

// performance in giga-multiplies performed per second:  
fprintf(stderr, "GigaMultsPerSecond: %.10lf\n",  
(double)MATW*(double)MATW*(double)MATW/(time1-time0)/1000000000. );

Printing the Performance

status = clEnqueueReadBuffer( CmdQueue, dC, CL_FALSE, 0, cSize, hC, 0, NULL, NULL );  
Wait( CmdQueue );

Copying the Resulting Matrix from the Device back to the Host

clReleaseKernel( Kernel );  
clReleaseProgram( Program );  
clReleaseCommandQueue( CmdQueue );  
clReleaseMemObject( dA );  
clReleaseMemObject( dB );  
clReleaseMemObject( dMW );  
clReleaseMemObject( dC );

Cleaning Up