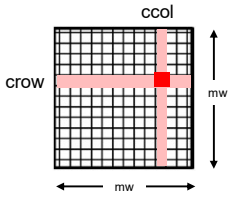
OpenCL Matrix Multiplication

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

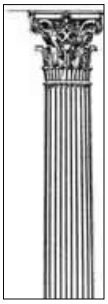


Oregon State University
Computer Graphics

Matrices

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

A column:



4 columns

3 rows

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Matrix sizes are termed “#rows x #columns”, so this is a 3x4 matrix

Oregon State University
Computer Graphics

Square Matrices

A square matrix has the same number of rows and columns

3 columns

3 rows

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

This is a 3x3 matrix

Oregon State University
Computer Graphics

Matrix Multiplication

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

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \longrightarrow 4*1 + 5*2 + 6*3 \longrightarrow \mathbf{32}$$

A **B** **C**

1x3 3x1 1x1

Oregon State University
Computer Graphics

Matrix Multiplication

Two matrices, A and B, can be multiplied if the number of columns in A equals the number of rows in B. The result is a matrix that has the same number of rows as A and the same number of columns as B.

$$\underset{\mathbf{A}}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{\mathbf{B}}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{\mathbf{C}}{\begin{bmatrix} 32 \end{bmatrix}}$$

I x K K x J I x J

Matrix Multiplication in Software

Here's how to remember how to do it:

$$\underset{\mathbf{A}}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{\mathbf{B}}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{\mathbf{C}}{\begin{bmatrix} 32 \end{bmatrix}}$$

1. **C = A * B**
2. **[I x J] = [I x K] * [K x J]**

I x J = I x K K x J

```
C[ i ][ j ] +=      A[ i ][ k ] * B[ k ][ j ] ;
```

Matrix Multiplication in CPU Software

$$\underset{A}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{B}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{C}{\begin{bmatrix} 32 \end{bmatrix}}$$

```
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 ];
                }
        }
}
```

Oregon State University
Computer Graphics

Note: numAcols *must* == numBrows !

Matrix Multiplication in CPU Software

$$\underset{A}{\begin{bmatrix} [1 \quad 2 \quad 3] \end{bmatrix}} * \underset{B}{\begin{bmatrix} \begin{Bmatrix} 4 \\ 5 \\ 6 \end{Bmatrix} \end{bmatrix}} \Rightarrow \underset{C}{\begin{bmatrix} 32 \end{bmatrix}}$$

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:

```
C[ i ][ j ] = A[i][0] * B[0][j]  +  A[i][1] * B[1][j]  +  A[i][2] * B[2][j]  +  A[i][3] * B[3][j] . . .
```

Oregon State University
Computer Graphics

Doing it in OpenCL: #defines, #includes, and Globals

9

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#include "cl.h"
#include "cl_platform.h"

// the matrix-width and the number of work-items per work-group:
// note: the matrices are actually MATWxMATW and the work group sizes are LOCALSIZExLOCALSIZE:
#ifndef MATW
#define MATW            1024
#endif

#ifndef LOCALSIZE
#define LOCALSIZE       8
#endif

// opencl objects:
cl_platform_id          Platform;
cl_device_id            Device;
cl_kernel               Kernel;
cl_program              Program;
cl_context              Context;
cl_command_queue        CmdQueue;

float           hA[MATW][MATW];
float           hB[MATW][MATW];
float           hC[MATW][MATW];

const char *        CL_FILE_NAME = { "proj06.cl" };
```

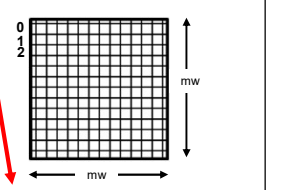
Oreg
Uni
Comput

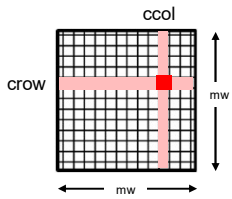
3, 2023

The .cl Kernel Function

10

```
#define IN
#define OUT

kernel
void
MatrixMult( IN global const float *dA, IN global const float *dB, IN global int *dMW, OUT global float *dC )
{
        // [dA] is dMW x dMW
        // [dB] is dMW x dMW
        // [dC] is dMW x dMW
        // but all the matrixs' rows are really linear in memory

        int mw   = *dMW;
        int crow = get_global_id( 0 );      // which row we are filling
        int ccol  = get_global_id( 1 );     // which column we are filling

        int aindex = crow * mw;             // a[i][0]
        int bindex = ccol;                  // b[0][j]
        int cindex = crow * mw + ccol;      // c[i][j]

        float cij = 0.;
        for( int k = 0; k < mw; k++ )
        {
                cij += dA[aindex] * dB[bindex];
                aindex++;
                bindex += mw;
        }
        dC[cindex] = cij;
}
```





Oregon State
University
Computer Graphics

mjb – May 3, 2023

Setting Up the Memory for the Matrices


```
int mw = MATW;
size_t aSize = MATW * MATW * sizeof(float);
size_t bSize = MATW * MATW * sizeof(float);
size_t mwSize = sizeof(mw);
size_t cSize = MATW * MATW * sizeof(float);

cl_mem dA = clCreateBuffer( Context, CL_MEM_READ_ONLY,  aSize, NULL, &status );
. . .

status = clEnqueueWriteBuffer( CmdQueue, dA, CL_FALSE, 0, aSize, hA, 0, NULL, NULL );
. . .


Wait( CmdQueue );
```

Oregon State University
Computer Graphics

Setting up the Kernel Function Arguments

Remember that our kernel function looks like this:

```
kernel
void
MatrixMult( IN global const float *dA, IN global const float *dB, IN global int *dMW, OUT global float *dC )
```

So the definition of the arguments needs to look like this:

```
Kernel = clCreateKernel( Program, "MatrixMult", &status );

status = clSetKernelArg( Kernel, 0, sizeof(cl_mem), &dA );
status = clSetKernelArg( Kernel, 1, sizeof(cl_mem), &dB );
status = clSetKernelArg( Kernel, 2, sizeof(cl_mem), &dMW );
status = clSetKernelArg( Kernel, 3, sizeof(cl_mem), &dC );
```

Oregon State University
Computer Graphics

Executing the Kernel

13

```
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( );
```

Oregon State University
Computer Graphics

Printing the Performance

14

```
// performance in giga-multiplies performed per second:

 fprintf( stderr, "GigaMultsPerSecond: %10.2lf\n",
          (double)MATW*(double)MATW*(double)MATW/(time1-time0)/1000000000. );
```

Oregon State University
Computer Graphics

Copying the Resulting Matrix from the Device back to the Host

15

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

Wait( CmdQueue );
```

Oregon State University
Computer Graphics

mjb – May 3, 2023

Cleaning Up

16

```
clReleaseKernel(          Kernel   );
clReleaseProgram(         Program  );
clReleaseCommandQueue(    CmdQueue );

clReleaseMemObject(       dA  );
clReleaseMemObject(       dB  );
clReleaseMemObject(       dMW  );
clReleaseMemObject(       dC  );
```

Oregon State University
Computer Graphics

mjb – May 3, 2023