

CUDA Matrix Multiplication



Oregon State
University

Mike Bailey

mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State
University
Computer Graphics

Anatomy of the CUDA *matrixMult* Program: #defines, #includes, and Globals

2

```
#include <stdio.h>
#include <assert.h>
#include <malloc.h>
#include <math.h>
#include <stdlib.h>

#include <cuda_runtime.h>
#include "helper_functions.h"
#include "helper_cuda.h"

#ifndef MATRIX_SIZE
#define MATRIX_SIZE 1024
#endif

#define AROWS      MATRIX_SIZE
#define ACOLS      MATRIX_SIZE

#define BROWS      MATRIX_SIZE
#define BCOLS      MATRIX_SIZE
#define ACOLSBROWS ACOLS      // better be the same!
#define CROWS      AROWS
#define CCOLS      BCOLS

#ifndef NUMT
#define NUMT 32
#endif

float hA[AROWS][ACOLS];      // globals
float hB[BROWS][BCOLS];      // globals
float hC[CROWS][CCOLS];      // globals
```

Anatomy of a CUDA Program: Error-Checking

3

```
void  
CudaCheckError( )  
{  
    cudaError_t e = cudaGetLastError( );  
    if( e != cudaSuccess )  
    {  
        fprintf( stderr, "CUDA failure %s:%d: '%s'\n", __FILE__, __LINE__, cudaGetErrorString(e));  
    }  
}
```

Anatomy of a CUDA Program: The Kernel Function

4

```
__global__ void MatrixMul( float *A, float *B, float *C )
{
    // [A] is AROWS x ACOLS
    // [B] is BROWS x BCOLS
    // [C] is CROWS x CCOLS = AROWS x BCOLS

    int blockDim = blockDim.x*blockDim.y;
    int blockIdx = blockIdx.x*blockDim.x + blockIdx.y;
    int gid      = blockIdx.y*blockDim.x + blockIdx.x;

    int crow = gid / CCOLS;
    int ccol = gid % CCOLS;

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

    float cij = 0.;
    for( int k = 0; k < ACOLSBROWS; k++ )
    {
        cij += A[aindex] * B[bindex];
        aindex++;
        bindex += BCOLS;
    }
    C[cindex] = cij;
    // __syncthreads( );
}
```

Anatomy of a CUDA Program: Setting Up the Memory for the Matrices

5

```
// allocate device memory:
```

```
float *dA, *dB, *dC;  
cudaMalloc( (void **>(&dA), AROWS*ACOLS*sizeof(float) );  
cudaMalloc( (void **>(&dB), BROWS*BCOLS*sizeof(float) );  
cudaMalloc( (void **>(&dC), CROWS*CCOLS*sizeof(float) );  
CudaCheckError( );
```

```
// copy host memory to device memory:
```

```
cudaMemcpy( dA, hA, AROWS*ACOLS*sizeof(float), cudaMemcpyHostToDevice );  
cudaMemcpy( dB, hB, BROWS*BCOLS*sizeof(float), cudaMemcpyHostToDevice );
```

This is a defined constant in one of the CUDA .h files

In **cudaMemcpy()**, it's *always* the second argument getting copied to the first!

first

second

Anatomy of a CUDA Program: Getting Ready to Execute

6

```
// setup execution parameters:
dim3 threads( NUMT, NUMT, 1 );
if( threads.x > CROWS )
    threads.x = CROWS;
if( threads.y > CCOLS )
    threads.y = CCOLS;
dim3 grid( CROWS / threads.x, CCOLS / threads.y );

// create cuda events for timing:
cudaEvent_t start, stop;
cudaEventCreate( &start );
cudaEventCreate( &stop );
CudaCheckError( );

// record the start event:
cudaEventRecord( start, NULL );
```

Anatomy of a CUDA Program: Executing the Kernel

7

```
// execute the kernel:
```

```
MatrixMul<<<grid,threads>>>( dA, dB, dC );
```

Function call arguments

of blocks to use # of threads to use per block

- The call to **MatrixMul**() returns *immediately*!
- If you upload the resulting array (dC) right away, it will have garbage in it.
- To block until the kernel is finished, call:
cudaDeviceSynchronize();

Anatomy of a CUDA Program: Getting the Stop Time and Printing Performance

8

```
cudaDeviceSynchronize( );

// record the stop event:
cudaEventRecord( stop, NULL );

// wait for the stop event to complete:
cudaEventSynchronize( stop );

float msecTotal;
cudaEventElapsedTime( &millisecsTotal, start, stop );           // note: this in milliseconds

// performance in multiplies per second:

double secondsTotal = millisecsTotal / 1000.0;           // change it to seconds
double multipliesTotal = (double)CROWS * (double)CCOLS * (double)ACOLSBROWS;
double gigaMultipliesPerSecond = ( multipliesTotal / 1000000000. ) / secondsTotal;
fprintf( stderr, "%6d\t%6d\t%10.3lf\n", CROWS, CCOLS, gigaMultipliesPerSecond );
```



Anatomy of a CUDA Program: Copying the Matrix from the Device back to the Host

```
cudaMemcpy( hC, dC , CROWS*CCOLS*sizeof(float) , cudaMemcpyDeviceToHost );  
CudaCheckError( );
```

// clean up:

```
cudaFree( dA );  
cudaFree( dB );  
cudaFree( dC );  
CudaCheckError( );
```

This is a defined constant in one of the CUDA .h files

In **cudaMemcpy()**, it's *a/ways* the second argument getting copied to the first!

first

second

