
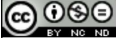


1


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/)



cudaMatrixMult.pptx

mjb - May 4, 2021

1

2

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

```

#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

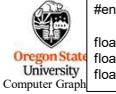
#define AROWS      MATRIX_SIZE
#define ACOLS      MATRIX_SIZE

#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 h[AROWS][ACOLS];
float h[BROWS][BCOLS];
float h[CROWS][CCOLS];

```



mjb - May 4, 2021

2


3

Anatomy of a CUDA Program: Error-Checking

```

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

```



mjb - May 4, 2021

3

4

Anatomy of a CUDA Program: The Kernel Function

```

__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

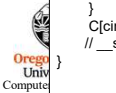
    int blockDim = blockDim.y*gridDim.x + blockDim.x;
    int blockThreads = blockDim*blockDim.x*blockDim.y;
    int gid = blockDim + threadIdx.y*blockDim.x + threadIdx.x;

    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( );
}

```



mjb - May 4, 2021

4

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

5

```


// allocate device memory:
float *dA, *dB, *dC;
cudaMalloc( (void **)&dA, sizeof(hA) );
cudaMalloc( (void **)&dB, sizeof(hB) );
cudaMalloc( (void **)&dC, sizeof(hC) );
CudaCheckError( );

// copy host memory to device memory:
cudaMemcpy( dA, hA, sizeof(hA), cudaMemcpyHostToDevice );
cudaMemcpy( dB, hB, sizeof(hB), 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!



mjb - May 4, 2021

5

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 );

```



mjb - May 4, 2021

6

Anatomy of a CUDA Program: Executing the Kernel

7

```


// execute the kernel:
MatrixMul<<<grid, threads>>>( dA, dB, dC );

```

Function call arguments

of blocks # of threads 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();**



mjb - May 4, 2021

7

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( &msecsTotal, start, stop );      // note: this in milliseconds

// performance in multiplies per second:

double secondsTotal = msecsTotal / 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%10.3f\n", CROWS, CCOLS, gigaMultipliesPerSecond );

```



mjb - May 4, 2021

8

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

9

```
cudaMemcpy( hC, dC, sizeof(hC), 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 always the second argument getting copied to the first!



mjb - May 4, 2021