

```
Anatomy of the CUDA arrayMult Program:
                                                                                                     2
                            #defines, #includes, and Globals
    #include <stdio.h>
    #include <assert.h>
    #include <malloc.h>
    #include <math.h>
    #include <stdlib.h>
    // CUDA runtime
    #include <cuda_runtime.h>
    // Helper functions and utilities to work with CUDA
    #include "helper_functions.h"
    #include "helper_cuda.h"
    #ifndef THREADS_PER_BLOCK
    #define THREADS_PER_BLOCK
#endif
                                           128
                                                      // number of threads in each block
    #ifndef DATASET_SIZE
                                                       // DON'T CALL THIS "ARRAYSIZE"!
    #define DATASET_SIZE
                                     (8*1024*1024) // size of the array
    #endif
    float hA[ DATASET_SIZE ];
    float hB[ DATASET_SIZE ];
                                     The defined constant ARRAYSIZE is already used in one of the CUDA .h files
    float hC[ DATASET_SIZE ];
Computer Graphics
                                                                                             mjb - March 22, 2020
```

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 – March 22, 2020

3

Anatomy of a CUDA Program: The Kernel Function

4

Note: "___" is 2 underscore characters



mjb – March 22, 2020

```
Anatomy of a CUDA Program:

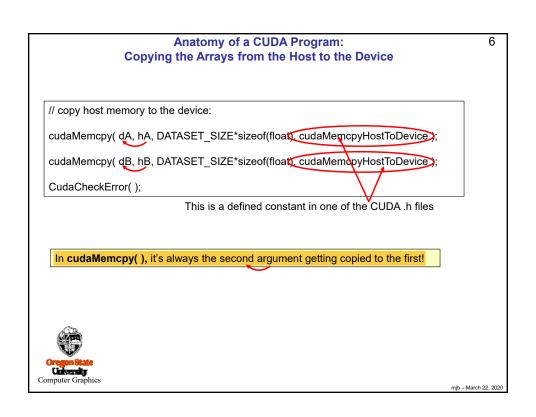
Setting Up the Memory for the Arrays

// fill host memory:
for( int i = 0; i < SIZE; i++ )
{
    hA[i] = hB[i] = (float) sqrtf( (float)i );
}

// allocate device memory:
float *dA, *dB, *dC;

cudaMalloc( (void **)(&dA), sizeof(hA) );
cudaMalloc( (void **)(&dB), sizeof(hB) );
cudaMalloc( (void **)(&dC), sizeof(hC) );

CudaCheckError( );
```



Anatomy of a CUDA Program: Getting Ready to Execute

```
// setup the execution parameters:
dim3 grid( DATASET_SIZE / THREADS_PER_BLOCK, 1, 1 );
dim3 threads( THREADS_PER_BLOCK, 1, 1 );

// create and start the timer:
cudaDeviceSynchronize( );

// allocate the events that we'll use for timing:
cudaEvent_t start, stop;
cudaEventCreate( &start );
cudaEventCreate( &stop );
CudaCheckError( );

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



mjb - March 22, 2020

mjb - March 22, 2020

Anatomy of a CUDA Program: Executing the Kernel // execute the kernel: ArrayMul<<< @rid threads >>> (dA, dB, dC); Function call arguments # of blocks # of threads per block The call to ArrayMul() 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: 9 **Getting the Stop Time and Printing Performance** // record the stop event: cudaEventRecord(stop, NULL); CudaCheckError(); // wait for the stop event to complete: cudaEventSynchronize(stop); CudaCheckError(); float msecTotal; cudaEventElapsedTime(&msecTotal, start, stop); CudaCheckError(); // compute and print the performance double secondsTotal = 0.001 * (double)msecTotal; double multsPerSecond = (double)DATASET_SIZE / secondsTotal; double megaMultsPerSecond = multsPerSecond / 1000000.; fprintf(stderr, "%12d\t%4d\t%10.2lf\n", DATASET_SIZE, THREADS_PER_BLOCK, megaMultsPerSecond);



mjb - March 22, 2020

```
Anatomy of a CUDA Program:
Copying the Array from the Device to the Host

// copy result from the device to the host:
cudaMemcpy( hC, dC, sizeof(hC) cudaMemcpyDeviceToHost );
CudaCheckError();

// clean up:
cudaFree( dA );
cudaFree( dB );
cudaFree( dC );
CudaCheckError();

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

Anatomy of a CUDA Program: Running the Program

11

rabbit 139% cat Makefile

CUDA_PATH = /usr/local/apps/cuda/cuda-10.1

CUDA_BIN_PATH = \$(CUDA_PATH)/bin CUDA_NVCC = \$(CUDA_BIN_PATH)/nvcc

arrayMul: arrayMul.cu

\$(CUDA_NVCC) -o arrayMul arrayMul.cu

rabbit 140% make arrayMul

/usr/local/apps/cuda/cuda-10.1/bin/nvcc -o arrayMul arrayMul.cu

rabbit 141% ./arrayMul

8388608 128 16169.75



mjb - March 22, 2020

Anatomy of a CUDA Program: Running the Program within a Loop

12

rabbit 142% cat loop.csh

#!/bin/csh

foreach t (32 64 128 256)

/usr/local/apps/cuda/cuda-10.1/bin/nvcc -DTHREADS_PER_BLOCK=\$t -o arrayMul arrayMul.cu ./arrayMul

end

rabbit 143% loop.csh

8388608 32 9204.82 8388608 64 13363.10 8388608 128 16576.70 8388608 256 15496.81



mjb – March 22, 2020