CUDA Matrix Multiplication

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

CUDA Matrix Multiplication

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

```c
#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];
float hB[BROWS][BCOLS];
float hC[CROWS][CCOLS];
```

Anatomy of a CUDA Program: Error-Checking

```c
void CudaCheckError()
{
    cudaError_t e = cudaGetLastError();
    if( e != cudaSuccess )
    {
        fprintf(stderr, "CUDA failure %s:%d: '%s'
                            
```

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

```c
// 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

```c
// 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 );
```

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

In cudaMalloc(), it's always the second argument getting copied to the first!
// execute the kernel:
MatrixMul<<< grid, threads >>>( dA, dB, dC );

Anatomy of a CUDA Program:
Executing the Kernel

- 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

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 is in milliseconds
if ( performanceInMultipliesPerSecond )
double multipliesTotal = (double)CROWS * (double)CCOLS * (double)ACOLSBROWS;
double gigaMultipliesPerSecond = ( multipliesTotal / 1000000000. ) / secondsTotal;
fprintf( stderr, "%6d	%6d	%10.3lf
", CROWS, CCOLS, gigaMultipliesPerSecond );

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

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

In "this is a defined constant in one of the CUDA.h files"