

Joshua Caudill

CS6068

Assignment #4

Table of Contents

Introduction..... 4

Tools..... 4

Code..... 4

Results..... 7

Conclusion 10

Table of Figures

Figure 1: enum_gpu.cu Output 7

Figure 2: Generated Image of Julia Set..... 8

Figure 3: Generated Image of Mandelbrot Set 9

Introduction

The goal of this assignment was to practice the workflow of compiling and running CUDA code and to write CUDA code to generate an image in parallel. The `enum_gpu.cu` code taken from *CUDA by Example: An Introduction to General-Purpose GPU Programming* was executed on a Pitzer Desktop on the Ohio Supercomputer Center (OSC) to practice compiling and running CUDA code. The starter code taken from *CUDA by Example: An Introduction to General-Purpose GPU Programming*, `julia_gpu.cu`, generated an image of the Julia Set. This CUDA code was modified to generate an image of the Mandelbrot Set by initially setting the exponential component of the iterative function equal to zero.

Tools

- `julia_set_modified.cu`
- CUDA
- Pitzer Desktop (1 GPU, 48 Cores, 1 Visualization Node)

Code

```
/*
 * Copyright 1993-2010 NVIDIA Corporation. All rights reserved.
 *
 * NVIDIA Corporation and its licensors retain all intellectual property and
 * proprietary rights in and to this software and related documentation.
 * Any use, reproduction, disclosure, or distribution of this software
 * and related documentation without an express license agreement from
 * NVIDIA Corporation is strictly prohibited.
 *
 * Please refer to the applicable NVIDIA end user license agreement (EULA)
 * associated with this source code for terms and conditions that govern
 * your use of this NVIDIA software.
 */

#include "../common/book.h"
#include "../common/cpu_bitmap.h"

#define DIM 1000

struct cuComplex {
    float r;
```

Joshua Caudill - CS6068 - Assignment #4

```
float i;
__device__ cuComplex( float a, float b ) : r(a), i(b) {}
__device__ float magnitude2( void ) {
    return r * r + i * i;
}
__device__ cuComplex operator*(const cuComplex& a) {
    return cuComplex(r*a.r - i*a.i, i*a.r + r*a.i);
}
__device__ cuComplex operator+(const cuComplex& a) {
    return cuComplex(r+a.r, i+a.i);
}
};
```

```
__device__ int julia( int x, int y ) {
    const float scale = 1.5;
    float jx = scale * (float)(DIM/2 - x)/(DIM/2);
    float jy = scale * (float)(DIM/2 - y)/(DIM/2);

    cuComplex c(jx, jy);
    cuComplex a(0, 0);

    int i = 0;
    for (i=0; i<200; i++) {
        a = a * a + c;
        if (a.magnitude2() > 1000)
            return 0;
    }

    return 1;
}
```

```
__global__ void kernel( unsigned char *ptr ) {
    // map from blockIdx to pixel position
    int x = blockIdx.x;
    int y = blockIdx.y;
```

Joshua Caudill - CS6068 - Assignment #4

```
int offset = x + y * gridDim.x;

// now calculate the value at that position
int juliaValue = julia( x, y );
ptr[offset*4 + 0] = 255 * juliaValue;
ptr[offset*4 + 1] = 0;
ptr[offset*4 + 2] = 0;
ptr[offset*4 + 3] = 255;
}

// globals needed by the update routine
struct DataBlock {
    unsigned char    *dev_bitmap;
};

int main( void ) {
    DataBlock    data;
    CPUBitmap bitmap( DIM, DIM, &data );
    unsigned char    *dev_bitmap;

    HANDLE_ERROR( cudaMalloc( (void**)&dev_bitmap, bitmap.image_size() ) );
    data.dev_bitmap = dev_bitmap;

    dim3    grid(DIM,DIM);
    kernel<<<grid,1>>>( dev_bitmap );

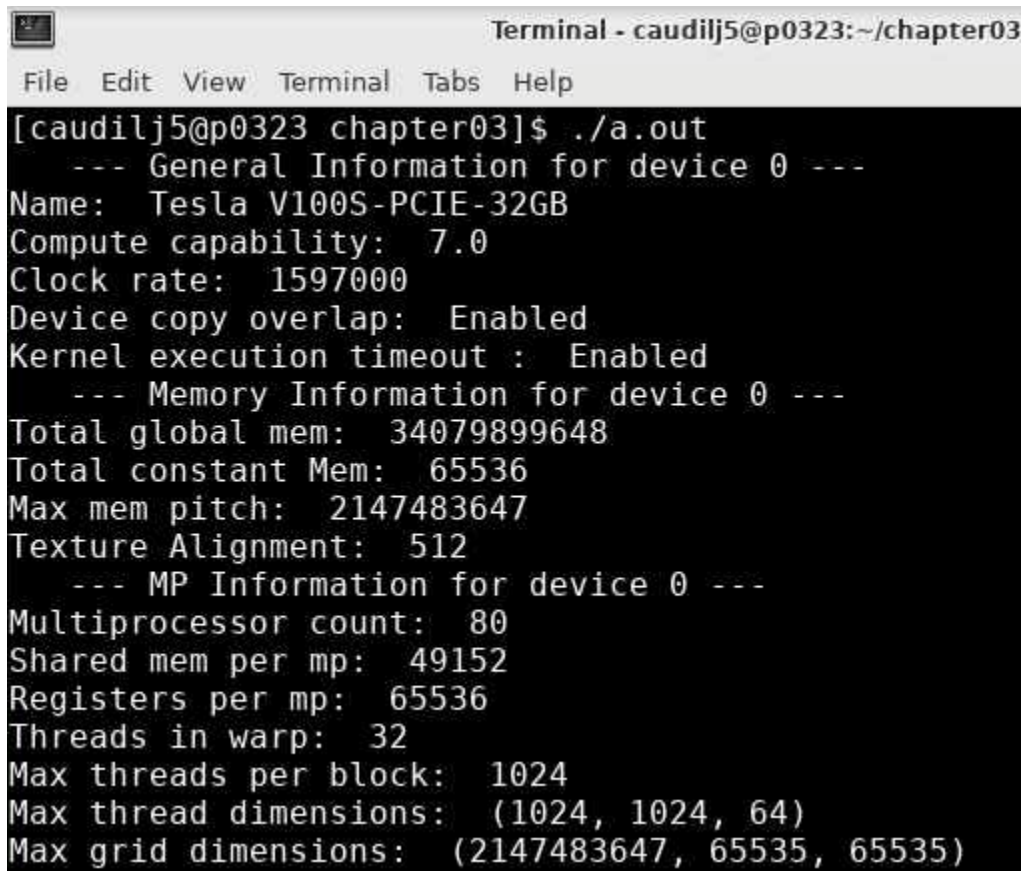
    HANDLE_ERROR( cudaMemcpy( bitmap.get_ptr(), dev_bitmap,
                               bitmap.image_size(),
                               cudaMemcpyDeviceToHost ) );

    HANDLE_ERROR( cudaFree( dev_bitmap ) );

    bitmap.display_and_exit();
}
```

Results

First, `enum_gpu.cu` was executed on the Pitzer Desktop. The Pitzer Desktop had 80 multiprocessors and 49,152 bytes of shared memory per multiprocessor. The results are shown in Figure 1. Second, the starter code from the textbook was executed. The starter code generated an image of the Julia Set in parallel. The generated image of the Julia Set is shown in Figure 2. Third, the modified CUDA code, `julia_set_modified.cu`, was executed. The generated image of the Mandelbrot Set is shown in Figure 3. This CUDA code was modified to generate an image of the Mandelbrot Set by initially setting the exponential component of the iterative function equal to zero.



```
Terminal - caudilj5@p0323:~/chapter03
File Edit View Terminal Tabs Help
[caudilj5@p0323 chapter03]$ ./a.out
--- General Information for device 0 ---
Name: Tesla V100S-PCIE-32GB
Compute capability: 7.0
Clock rate: 1597000
Device copy overlap: Enabled
Kernel execution timeout : Enabled
--- Memory Information for device 0 ---
Total global mem: 34079899648
Total constant Mem: 65536
Max mem pitch: 2147483647
Texture Alignment: 512
--- MP Information for device 0 ---
Multiprocessor count: 80
Shared mem per mp: 49152
Registers per mp: 65536
Threads in warp: 32
Max threads per block: 1024
Max thread dimensions: (1024, 1024, 64)
Max grid dimensions: (2147483647, 65535, 65535)
```

Figure 1: `enum_gpu.cu` Output

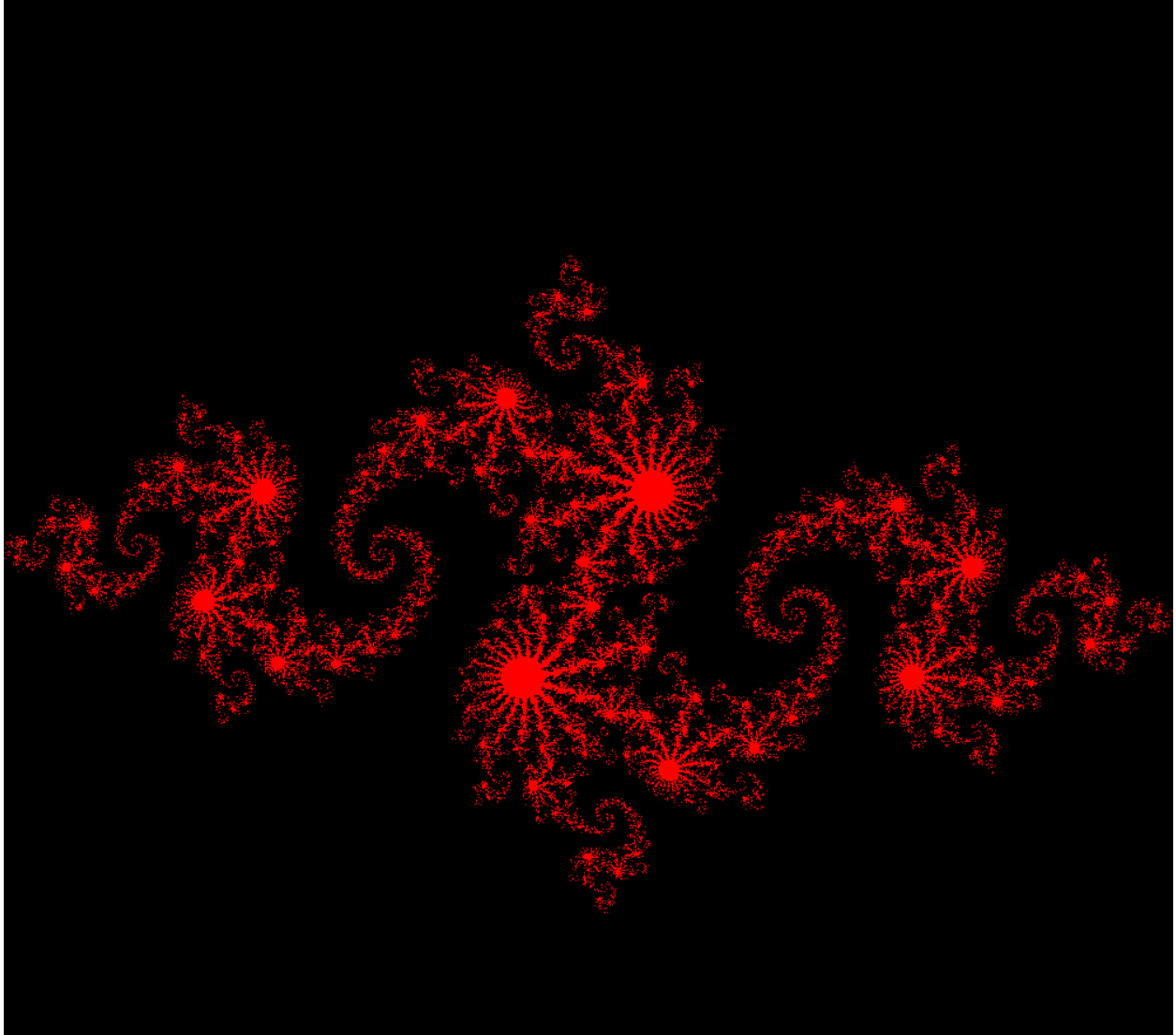


Figure 2: Generated Image of Julia Set

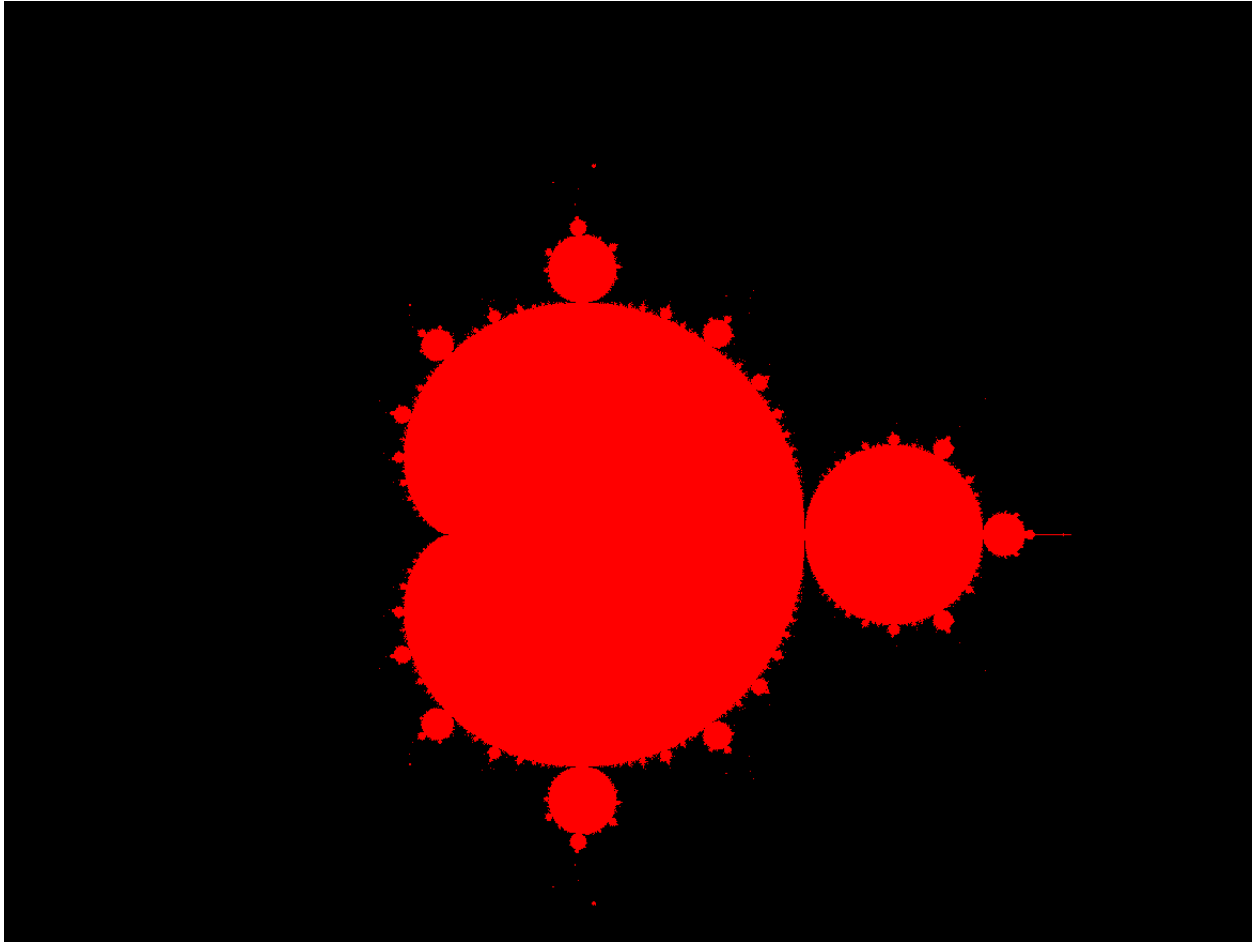


Figure 3: Generated Image of Mandelbrot Set

Conclusion

The `enum_gpu.cu` code taken from *CUDA by Example: An Introduction to General-Purpose GPU Programming* was executed on a Pitzer Desktop on the Ohio Supercomputer Center (OSC) to practice compiling and running CUDA code. The starter code taken from *CUDA by Example: An Introduction to General-Purpose GPU Programming*, `julia_gpu.cu`, generated an image of the Julia Set. This CUDA code was modified to generate an image of the Mandelbrot Set by initially setting the exponential component of the iterative function equal to zero.