Assignment 4 - CS671

Mandelbrot using CUDA

- Ashish Jindal (AJ523)

Q 1 -

Using GPU data as input -
In this implementation different available GPU functions are invoked to manipulate the memory on GPU and get the result. For .e.g for matlab the class gpuArray provides a set of such functions like ones(), linespace(), logspace() etc.

Performing the algorithm on each element independently -
In this methodology, the code for the algorithm is placed inside a new helper function which is then compiled to native GPU code before invocation.

In the previous method multiple GPU optimized functions are called each time we need to do an operation on the input via GPU, but in the later method the entire algorithm is first converted into native GPU code and then invoked only once, thus reducing significant overhead.

The disadvantage of later methodology is that, the code will take more invocation time although less computation time, so if the computation is small then converting the code to native code might be an overkill.

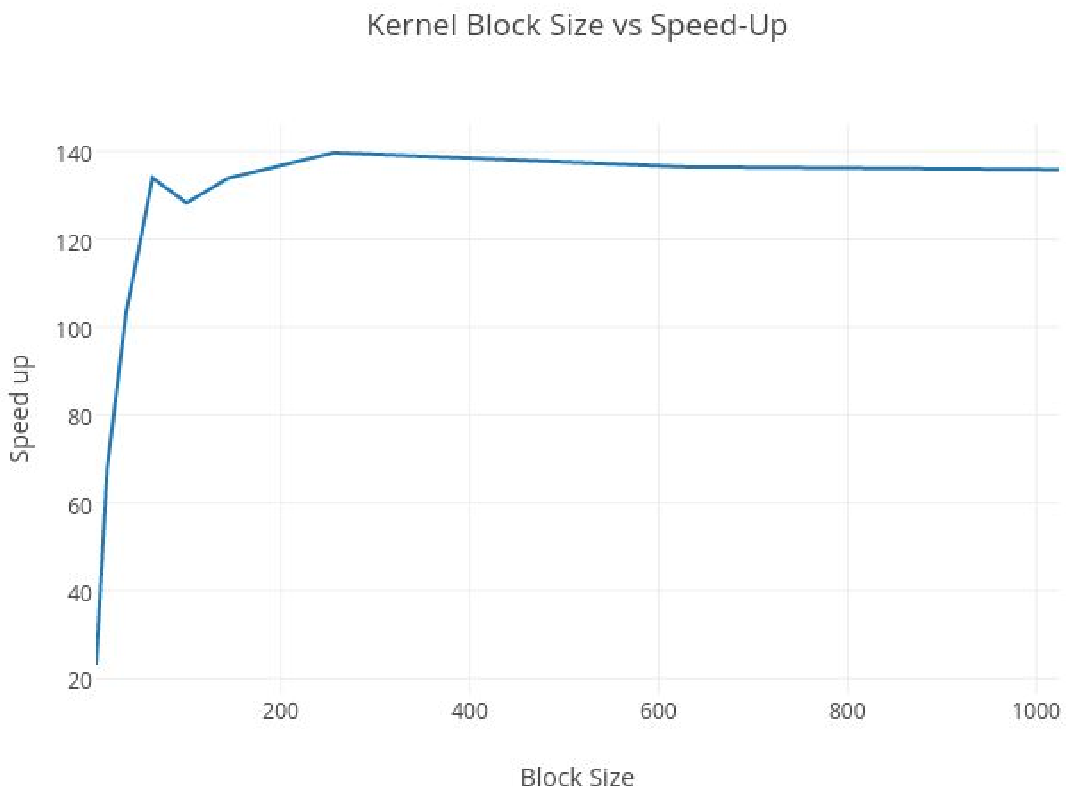
Q 2 -

1. Code - ./mandelbrot/mandelbrotGpu.cu

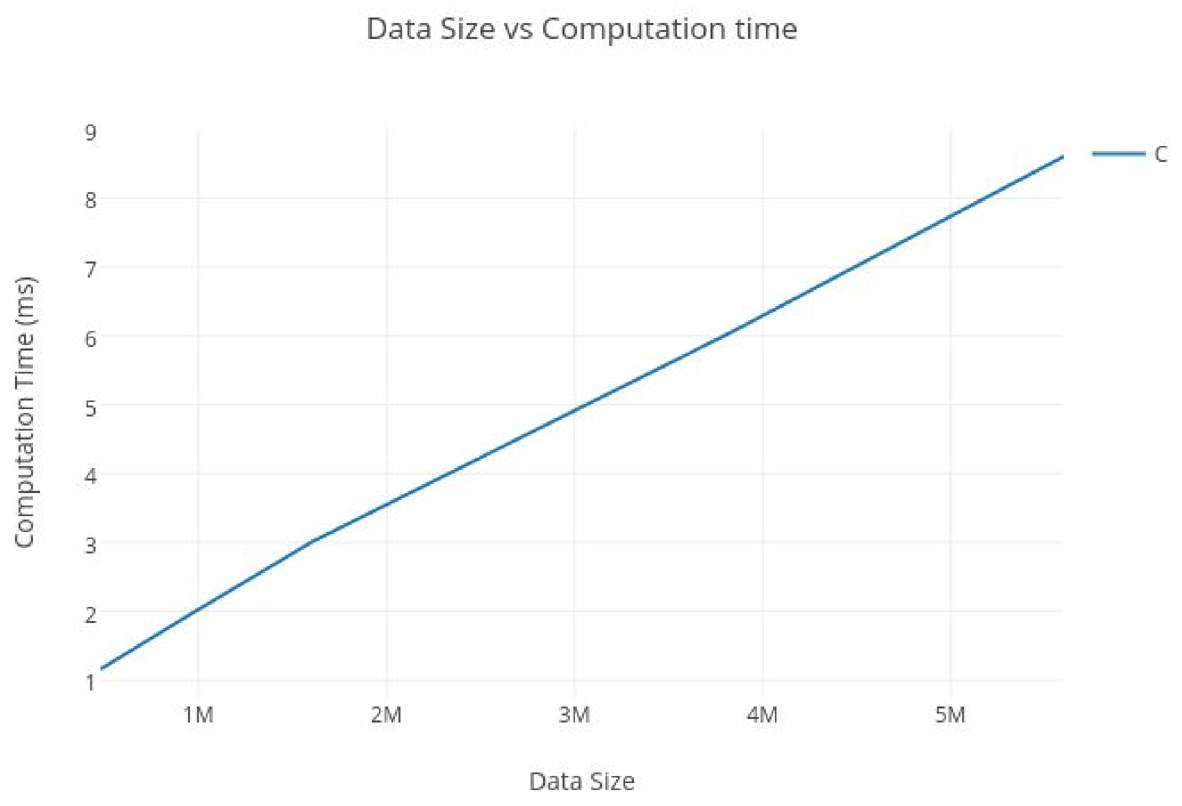
Q 3 -

Block Size (x * y)	Computation Time (ms)	Speed up
4	10.974	22.97
16	3.716	67.8
36	2.442	103.22

64	1.882	133.96
100	1.963	128.3
144	1.882	133.89
256	1.804	139.71
625	1.84	136.52
1024	1.855	135.88



Scaling -



Q 4 -

Kernel -

```
__global__ void mandelbrot_kernel(
float x0, float y0,
float x1, float y1,
int width, int height, int maxIterations, int* output) {
        int col = blockIdx.x*blockDim.x + threadIdx.x;
        int row = blockIdx.y*blockDim.y + threadIdx.y;
        if (row < height && col < width) {
                output[row*width + col] = mandelbrot_calc(x0, y0, x1, y1, width, height, row,
col, maxIterations);
        }
}
```

Kernel Invocation -

```
#define TILE_WIDTH 10
#define TILE_HEIGHT 10

int nTilesX = (width / TILE_WIDTH) + ((width % TILE_WIDTH == 0) ? 0 : 1);
nt nTilesY = (height / TILE_HEIGHT) + ((height % TILE_HEIGHT == 0) ? 0 : 1);

dim3 threadsPerBlock(TILE_WIDTH, TILE_HEIGHT);
dim3 blocksPerGrid(nTilesX, nTilesY);

mandelbrot_kernel<<<blocksPerGrid, threadsPerBlock>>>(x0, y0, x1, y1, width, height,
maxIterations, d_output);
```

Timing code -

If we simply enclose the kernel invocation code between timing code, then we will not get the correct timing as it will only give us the kernel launch time, instead we want the launch time as well as the computation time. I use following CUDa events to measure the time of computation in CUDA -

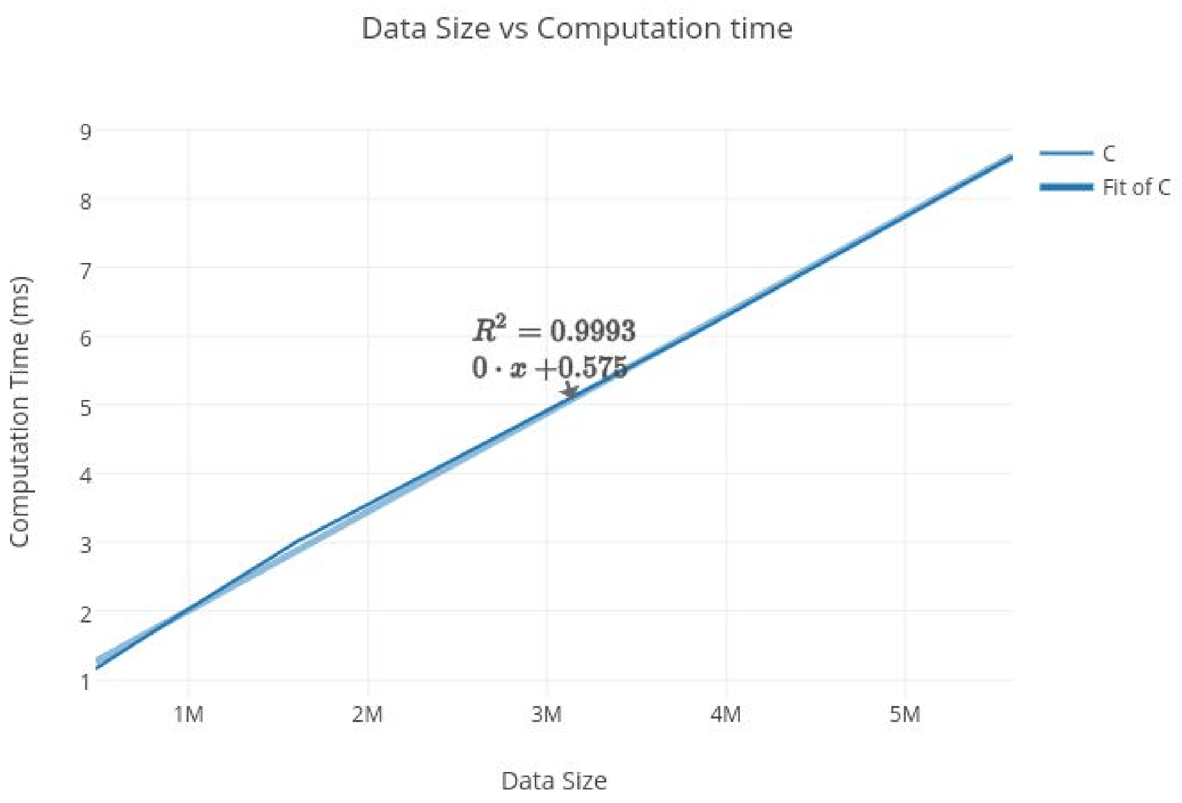
```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);

….

cudaEventRecord(stop);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&millisec, start, stop);
```

Analysis -

We can develop a model of following type -

$T_{Comm} = \alpha + \beta M$

1. T_{Comm}= Processing time
2. α = latency
3. β = inverse bandwidth
4. M = Data size



From the resulting graphs we can see that, the values of α and β are as follows -

α = 0.5745896873285 ms

β = 0.0000014363823 ms