Report

YUEFENG LIANG 201350099

Problem:

The discretized function is

$$f(x) = e^{-(x-2)^2} + e^{-(x-6)^2/10} + \frac{1}{x^2+1}$$

The values x[i] are over the range -100 to 100, and the values for F(x[i]) are related to each x[i]. After that, you are asked to find the MAXIMUN of F(x[i]) and the related i. The CUDA kernel should be used to determine F(x[i]), though OpenMp or CUDA could be used to parallelize the formation of x[i]. It is required that data points between -100 to 100 are limited to 10 million, 30 million and 50 million.

Parallelism:

Firstly, declare both host and device variables on the cpu.

```
//memory for device(GPU)
  float *x_dev, *y_dev;
  //set up array;
  x = (float *)malloc(N * sizeof(*x));
  y = (float *)malloc(N * sizeof(*y));
  //set up device memory
  cudaMalloc(&x_dev, N * sizeof(*x));
  cudaMalloc(&y_dev, N * sizeof(*y));
```

The OpenMp parallelism code are used to assign 10 million, 30 million or 50 million data points to x[i], which is between -100 and 100. N=10000000, 30000000, 50000000; len = (100-(-100))/N;

```
#pragma omp parallel for
        for (i = 0; i < N ; i++)
        {
           x[i] = (-100)+len*i;
        }
  finish_cpu = clock();
```

Then, initialize on host and copy to device

```
// copy to device
cudaMemcpy(x_dev, x, N * sizeof(float), cudaMemcpyHostToDevice);
// see if any errors
cudaError err = cudaGetLastError();
if ( err != cudaSuccess) {
    printf("(1) CUDA RT error: %s \n",cudaGetErrorString(err));
}
```

The follow code is initializing the CUDA kernel function and setting up threads on per block (according to parameters of GPU, Tesla, the maximum threads per block are 1024).

```
// init the kernel
int threadsPerBlock;
float maxThreadsPerBlock = 1024;
threadsPerBlock = (N > maxThreadsPerBlock ? maxThreadsPerBlock : N);
int blocks = ceil((float)N / (float)threadsPerBlock);
printf("eg %d threads/block on %d blocks\n", threadsPerBlock, blocks);

cudaEventRecord(start,0);
vecKernel <<<blocks, threadsPerBlock >>> (x_dev,y_dev, N);
cudaEventRecord(stop);
```

CUDA kernel function declaration, and the code is used to calculate the formula to obtain F(x[i]):

```
__global__ void vecKernel(float*x, float *y, int num) {
  // parallel control via varying index
  int my_i = threadIdx.x + blockIdx.x*blockDim.x;
  // handle my_i exceeds num
 if(my_i<num){
  y[my_i] = 1 / exp((x[my_i] - 2)*(x[my_i] - 2))
            + 1 / exp((x[my_i] - 6)*(x[my_i] - 6) / 10)
            + 1 / (x[my_i] * x[my_i] + 1);
  }
}
```

Moreover, copy result back to host.

```
// copy results back
cudaMemcpy(y, y_dev, N * sizeof(float), cudaMemcpyDeviceToHost);
// see if any errors
err = cudaGetLastError();
if (err != cudaSuccess) {
  printf("(3) CUDA RT error: %s \n", cudaGetErrorString(err));
}
```

Finally, I used OpenMp parallelism code to find the maximum values of F(x[i]).

```
#pragma omp parallel for
for (i = 0; i < N ; i++)
{
  if (max < y[i]) {
    max = y[i];
    index = i;
  }
}
```

Results:

Data points = 10000000;

```
eg 1024 threads/block on 9766 blocks
CUDA: Max is 5100047th number = 1.4019
Serial: Max is 5100047th number = 1.4019
elapsed GPU time: 21.272224 millisecs
elapsed CPU time for init+cpy:   0.080000 secs
elapsed CPU time for kernel:   0.020000 secs
elapsed CPU time for finding Max:   0.050000 secs
elapsed CPU time *total*:   0.150000 secs
Serial Code: 0.470 seconds
```

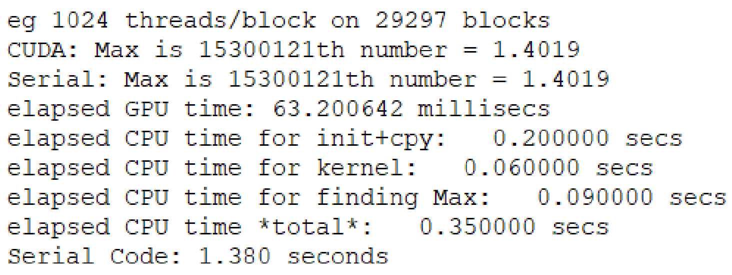
When x[i] = 5100047, F(x[i]) get the maximum value 1.4019;

Time (GPU):0.021secs;

Efficiency(GPU): 0.021/0.15 ≈14%;

Efficiency (OpenMp): (0.05+0.08)/0.15 ≈ 86%

Data points = 30000000;



```
eg 1024 threads/block on 29297 blocks
CUDA: Max is 15300121th number = 1.4019
Serial: Max is 15300121th number = 1.4019
elapsed GPU time: 63.200642 millisecs
elapsed CPU time for init+cpy:   0.200000 secs
elapsed CPU time for kernel:   0.060000 secs
elapsed CPU time for finding Max:   0.090000 secs
elapsed CPU time *total*:   0.350000 secs
Serial Code: 1.380 seconds
```

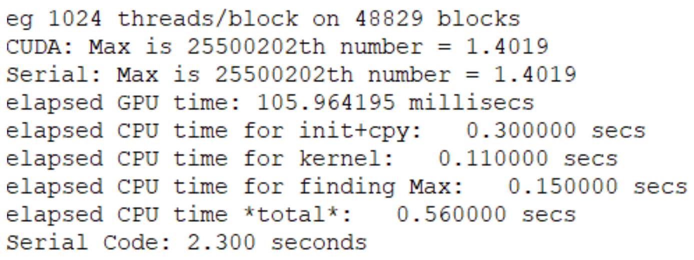
When x[i] = 15300121, F(x[i]) get the maximum value 1.4019;

Time (GPU):0.063secs;

Efficiency(GPU): 0.063/0.34 ≈18.53%;

Efficiency (OpenMp): (0.09+0.2)/0.35 ≈82.86%

Data points = 50000000;



```
eg 1024 threads/block on 48829 blocks
CUDA: Max is 25500202th number = 1.4019
Serial: Max is 25500202th number = 1.4019
elapsed GPU time: 105.964195 millisecs
elapsed CPU time for init+cpy:   0.300000 secs
elapsed CPU time for kernel:   0.110000 secs
elapsed CPU time for finding Max:   0.150000 secs
elapsed CPU time *total*:   0.560000 secs
Serial Code: 2.300 seconds
```

When x[i] = 25500202, F(x[i]) get the maximum value 1.4019;

Time (GPU):0.106secs;

Efficiency(GPU): 0.106/0.56 ≈ 18.93%;

Efficiency (OpenMp): (0.3+0.15)/0.56 ≈ 80.36%

Conclusion:

According to results, with the increase of data points, the efficiency of GPU improves, which show that it is easier to represent the performance of the GPU by computing large amount of data. However, there is some limitation of using GPU. According to different types of GPU, the maximum threads in each block is different. Also, the limitation of my project is the lack of GPU performance comparison to the different number of threads in different numbers of blocks. Appropriate amounts of threads in each block may lead the GPU to the highest efficiency.