REPORT

#problem2_document

과목명	멀티코어컴퓨팅 01분반
교수명	손봉수
학번	20206319
학과	소프트웨어학부 소프트웨어전공
이름	김가연

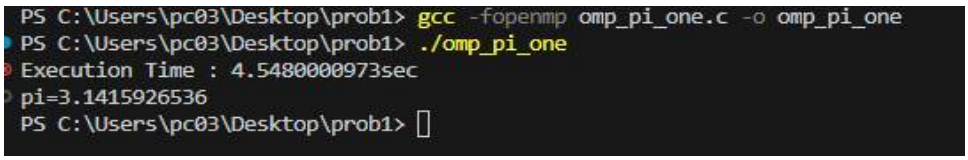
(a) source code

```
#include <stdio.h>
#include <thrust/device_vector.h>
#include <thrust/functional.h>
#include <thrust/system/cuda/error.h>
#include <thrust/system/cuda/execution_policy.h>
#include <chrono>

#define NUM_STEPS 1000000000

struct AtomicAdd{
    template <typename T>
    __device__ void operator()(T* addr, T val) const{
        atomicAdd(addr, val);
    }
};

struct ComputePi{
    double step;

    __device__ double operator()(int i) const{
        double x = (i + 0.5) * step;
        double fx = 4.0 / (1.0 + (x * x));
        return fx * step;
    }
};

int main(){
    double step = 1.0 / (double)NUM_STEPS;
    thrust::device_vector<double> sum_d(1, 0.0);

    ComputePi computePi{step};

    auto start_time = std::chrono::steady_clock::now();
    double pi = thrust::transform_reduce(thrust::cuda::par,
                                         thrust::counting_iterator<int>(0),
                                         thrust::counting_iterator<int>(NUM_STEPS + 1),
                                         computePi,
                                         0.0,
                                         thrust::plus<double>());

    cudaDeviceSynchronize();
    auto end_time = std::chrono::steady_clock::now();

    cudaError_t cuda_error = cudaGetLastError(); =
```

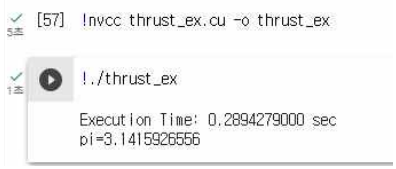
```
    if (cuda_error != cudaSuccess){
        fprintf(stderr, "CUDA error: %s\n", cudaGetErrorString(cuda_error));
        return 1;
    }
    std::chrono::duration<double> elapsed_seconds = end_time - start_time;
    printf("Execution Time: %.10lf sec\n", elapsed_seconds.count());
    printf("pi=%.10lf\n", pi);

    return 0;
}
```

(b) Execution time table and graphs of sequential program using only CPU [omp_pi_one.c (using N=1,000,000,000 and one thread)] and program using thrust

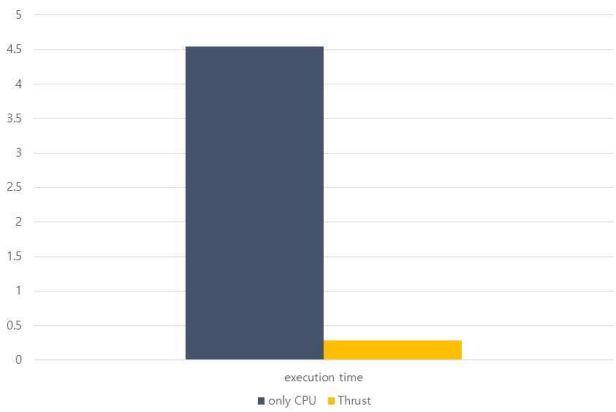
#omp_pi_one.c



#thrust_ex.cu



	Execution time
omp_pi_one.c	4.5480000973
thrust_ex.cu	0.2894279000



(c) Explanation/ interpretation on the results.

We can see that the execution time of only CPU and thrust is quite different. The reason can be guessed as follows.

1. With Thrust, GPUs process in parallel, and a single CPU has relatively low parallelism.

2. Integral is used using the Thrust library function, and this Thrust is designed to use an efficient GPU algorithm, and Thrust is more efficient because only CPU is performed with a simple loop.