Project 4: Problem 2

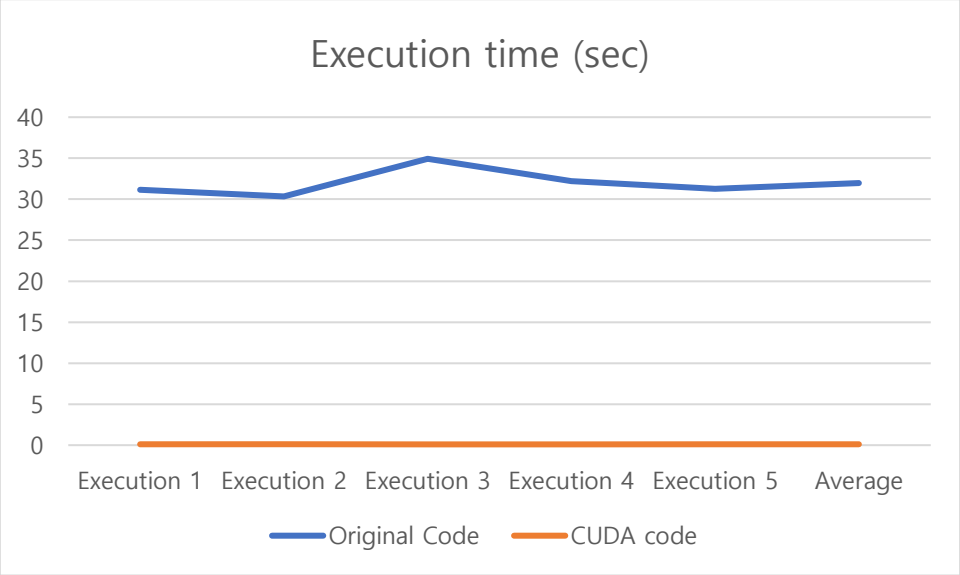
20204898 박소은

Source code

```
#include <stdio.h>
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>
#include <thrust/reduce.h>

#define NUM_STEPS 200000
#define STEP 1.0/NUM_STEPS

struct calculation {
  __host__ __device__
  double operator()(double i) {
    double x = (i+0.5)*STEP;
    return 4.0/(1.0+x*x);
  }
};

int main ()
{
  clock_t start_time = clock();

  // value "i" initialization
  thrust::device_vector<double> i(NUM_STEPS); // same with "i" in omp code
  thrust::sequence(i.begin(), i.end());    // 0 to NUM_STEPS in "i" vector

  thrust::device_vector<double> sum(NUM_STEPS); // vector to store the "sum"
  // same with the "for" statement in the original code
  thrust::transform(i.begin(), i.end(), sum.begin(), calculation());

  double result = thrust::reduce(sum.begin(), sum.end());    // summation
  double pi = STEP * result;

  clock_t end_time = clock();
  clock_t exe_time = end_time - start_time;
  double exe_time_sec = (double)(exe_time) / CLOCKS_PER_SEC;
  printf("Execution Time : %.10lf \n", exe_time_sec);
  printf("pi = %.10lf \n",pi);
}
```

Execution timetable & graphs

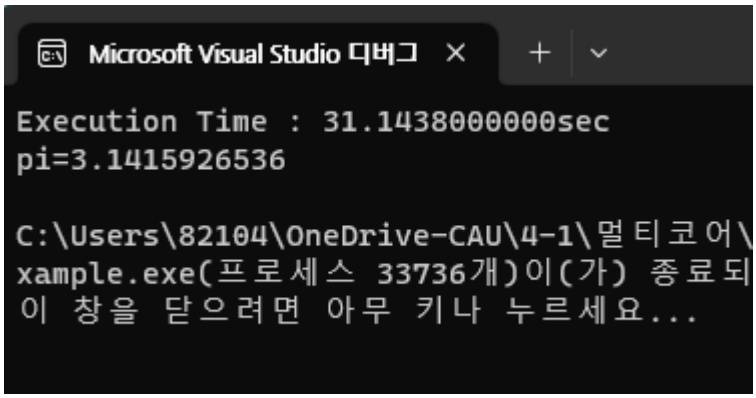
	Original Code	CUDA code
Execution 1	31.1438	0.103011
Execution 2	30.3359999	0.115084
Execution 3	34.9246633	0.094868
Execution 4	32.2134621	0.094868
Execution 5	31.2766153	0.095936
Average	**31.97890812**	**0.1007534**



Explanation/Interpretation on the results

Screen captures of output results.

- Original code(omp_pi_one.c)



- thrust_ex.cu

Explanation / Interpretation

The average execution time of comp_pi_one.c is 31.97 sec, and the average execution time of CUDA code using thrust is 0.10 sec. Thrust_ex.cu performed about 317.39 times better than the comp code. This is predictable because thrust_ex.cu used GPU and thrust library, and the comp_pi_one.c was done with one thread and CPU.

Although the thread count is not directly specified in the thrust_ex.cu code, thrust library internally creates a CUDA kernel to perform operations. The number of threads depends on the thrust implementation and GPU, and it is processed by internally creating an optimal thread configuration.

When running this example, I used google colab's GPU T4. T4 is Tesla T4 GPU developed by NVIDIA. With T4 and Thrust, a maximum of 1024 threads can be used, but it is difficult to accurately predict the number of threads actually used. However, as can be seen from the results, it is clear that it is calculated more efficient and faster than comp_pi_one.c, which specifies the number of threads as 1.