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)

## 텍스트, 폰트, 스크린샷이(가) 표시된 사진 자동 생성된 설명

* thrust\_ex.cu

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

## **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.