2023.1 Multicore Computing, Project #4

Problem 2

Course / Class:	Multicore Computing / Class 01
Instructor:	Bong-Soo Sohn
Date:	2023. 06. 13
Student ID:	20184757
Student Name:	Youngseok Joo

Table of Contents

1. Environment

- Hardware
    - CPU: Intel® Core™ i5-10400 CPU @ 2.90GHz
    - Memory: 16.0 GB, 2667MHz
    - GPU: NVIDIA GeForce GTX 1660 SUPER (1408 CUDA Cores, 6144MiB Memory)
- Software
    - Windows 10 Home
    - WSL2 (Windows Subsystem for Linux) – Ubuntu 20.04.6 LTS
    - CUDA Version: 12.1
    - NVIDIA Driver Version: 531.79
    - nvcc: V12.1.105, build cuda_12.1.r12.1/compiler.32688072_0

2. Source Code

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>

#include <iostream>
#include <iomanip>
#include <chrono>

using namespace std;
using namespace std::chrono;

int num_steps = 1000000000;
double step;

// Make function object for calculating pi
struct pi_functor
{
    const double step;

    // Get step on constructor
    pi_functor(double _step) : step(_step) {}

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

int main()
{
    double pi, sum = 0.0;

    step = 1.0 / (double) num_steps;

    // Start timer
    auto start_time = high_resolution_clock::now();

    // Allocate int vector with size num_steps on device
    thrust::device_vector<int> i(num_steps);
    // Set vector elements to 0, 1, 2, ...
    thrust::sequence(i.begin(), i.end());

    // Calculate sum with transform_reduce
    // First, calculate transformation with pi_functor function object
    // Then, add all vector elements with reduce
    sum = thrust::transform_reduce(i.begin(), i.end(), pi_functor(step), 0.0, thrust::plus<double>());

    pi = step * sum;

    // End timer, calculate execution time
    auto end_time = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(end_time - start_time);
    double execution_time = duration.count() / 1000000000.0;

    // Print execution time and result
    cout << "Execution Time : " << setprecision(11) <<  execution_time << "sec" << endl;
    cout << "pi=" << setprecision(11) << pi << endl;

    return 0;
}
```

3. Experimental Results

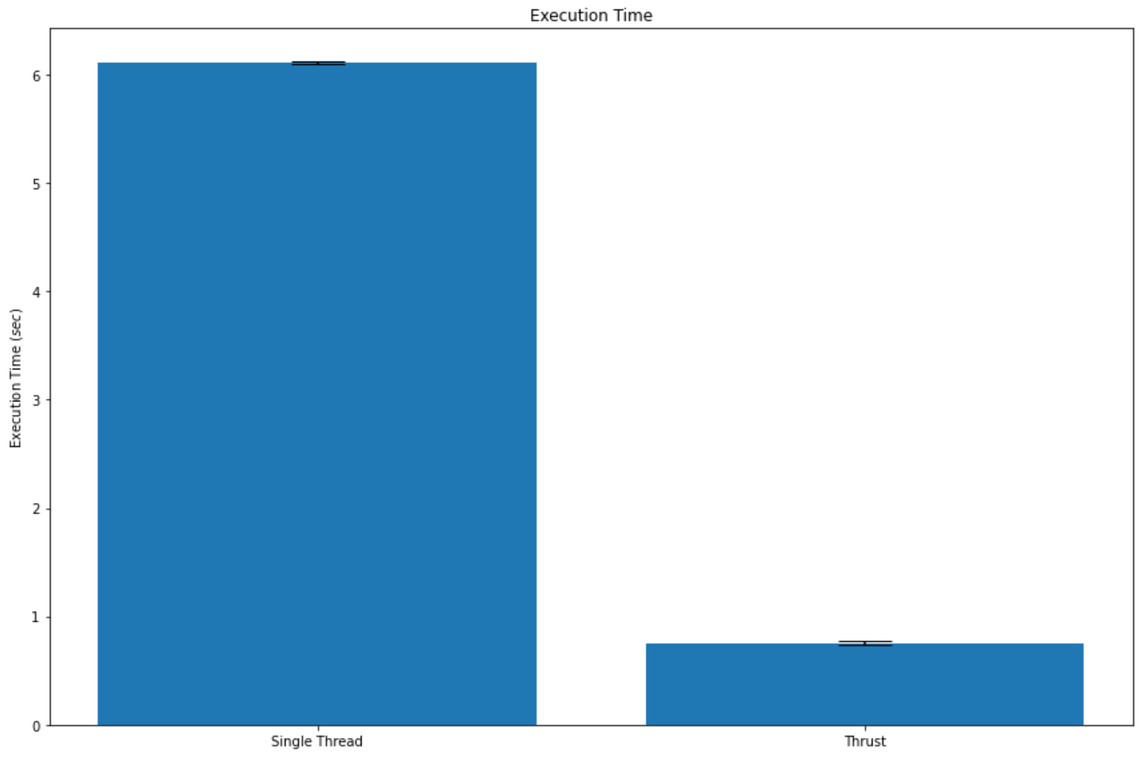

Figure 1. Bar Graph showing Execution Times of Single Thread and Thrust Implementation (10-Fold).

The Thrust version of the pi calculation was implemented by utilizing the thrust library. The repeated calculation inside the for-loop of the original single-thread implementation was made into a function object for thrust functionality. Then the function object was used in the thrust::transform_reduce function, which calculates the transformation function for each element in the vector and reduces by adding all elements.

Thrust implementation of pi calculation was tested by comparing execution time with the original single thread version. Compared to the single-thread version, the execution time of the thrust version was significantly shorter, where the single-thread version took 6.1101422976 seconds, and the thrust version took 0.7525024204 seconds on average, which is about eight times faster.