Project #4

Multicore Computing

Problem 2

Date	Jun 20, 2020
Instructor	Bongsoo Sohn
Std. Name	Junhyuck Woo
Std. ID	20145337

INDEX

INDEX **1**

ENVIRONMENT **2**

Single Thread Version *2*

Thrust Version *2*

SOURCE CODE **3**

Single Thread Version *3*

Thrust Version *4*

OUTPUT **5**

Single Thread Version *5*

Thrust Version *5*

EXPERIMENTAL RESULT **6**

ENVIRONMENT

Single Thread Version

Hardware
- MacBook Pro (15-inch, 2017)
- Processor: 2.8 GHz Quad-Core Intel Core i7
- Memory: 16GB 2133 MHz LPDDR3

Operating System
- macOS Catalina, ver: 10.15.4

IDE (Integrated Development Environment)
- Visual Studio Code 1. 45.1
- gcc version 8.4.0 (Homebrew GCC 8.4.0_1)

Testing Environment
- iTerm2
- Build 3.3.9
- openjdk 14.0.1 2020-04-14
  OpenJDK Runtime Environment (build 14.0.1+7)
  OpenJDK 64-Bit Server VM (build 14.0.1+7, mixed mode, sharing)

Thrust Version

Hardware
- Desktop
- Processor: AMD Ryzen 5 2600X Six-Core Processor
- Memory: 16GB
- GPU: GeForce GTX 107

Operating System
- Ubuntu, ver: 20.04 LTS

IDE (Integrated Development Environment)
- Visual Studio Code 1. 45.1
- CUDA ver: 10.2
- gcc (Ubuntu 7.5.0-6ubuntu2) 7.5.0
- g++ (Ubuntu 7.5.0-6ubuntu2) 7.5.0
- nvcc ver 10.2.89

Testing Environment
- iTerm2 (Used for terminal)
- Build 3.3.9

SOURCE CODE

Single Thread Version

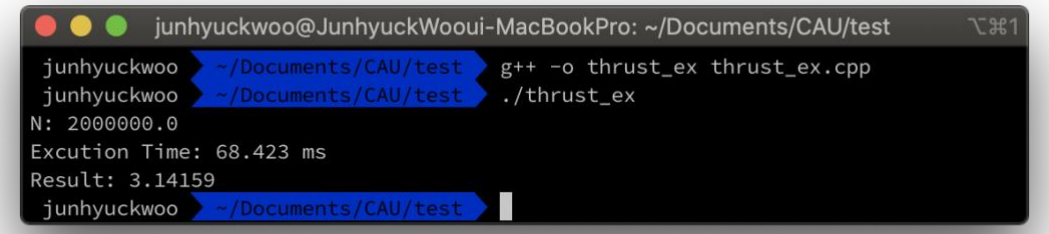
```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June3 20, 2020
// Project #4
// - Single Thread

#include <iostream>
#include <time.h>
#include <vector>

using namespace std;
#define N 2000000.0

int increase()
{
    static int i = 1;
    return i++;
}

int main(int argc, char* argv[])
{
    double sum = 0;
    double exec_time = 0;
    float tmp = 0;
    clock_t start_time, end_time;
    vector<double> X(N);
    vector<double> Z(N);

    // Start timer
    start_time = clock();

    // initialize X to 0,1,2,3, ....
    generate(X.begin(), X.end(), increase);
    // Divide X as N
    for (int i=0; i<N; i++) {
        X[i] = X[i] / N;
    }

    // Calculation
    for (int i=0; i<N; i++) {
        Z[i] = 4.0 / (X[i] * X[i] + 1); // Z = 4.0 / (X*X + 1)
        Z[i] = Z[i] / N;                // Z = Z * ( 1/N )
    }

    // Sum the calculation result
    for (int i=0; i<N; i++) {
        sum += Z[i];
    }

    // End timer
    end_time = clock();
    exec_time = (double)(end_time - start_time)*1000 / CLOCKS_PER_SEC;

    // Print the result
    cout << "N: 2000000.0" << endl;
    cout << "Excution Time: " << exec_time << " ms" << endl;
    cout << "Result: " << sum << endl;

    return 0;
}
```

Thrust Version

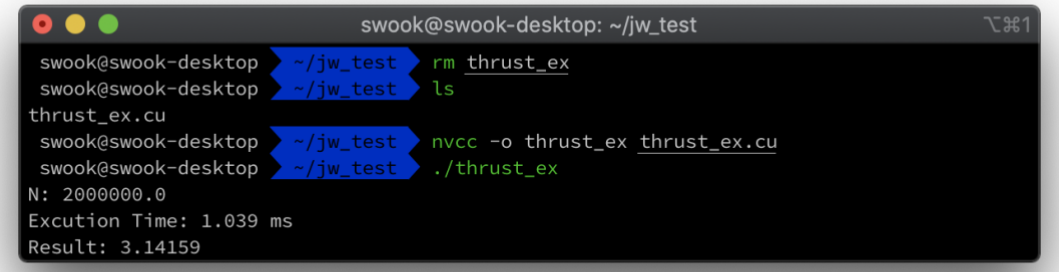
```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June3 20, 2020
// Project #4
// - Thrust

#include <thrust/fill.h>
#include <thrust/reduce.h>
#include <thrust/sequence.h>
#include <thrust/transform.h>
#include <thrust/device_vector.h>
#include <iostream>
#include <time.h>

using namespace std;
#define N 2000000.0

int main(int argc, char* argv[])
{
   float sum = 0;
   clock_t start_time, end_time;
   double exec_time = 0;

   // allocate three device_vectors
   thrust::device_vector<float> X(N);
   thrust::device_vector<float> Y(N);
   thrust::device_vector<float> Z(N);

   // Start timer;
   start_time = clock();

   // initialize X to 0,1,2,3, ....
   thrust::sequence(X.begin(), X.end());
   thrust::fill(Y.begin(), Y.end(), N);
   // Divide X as N
   thrust::transform(X.begin(), X.end(), Y.begin(), X.begin(), thrust::divides<float>());

   // Calculation
   thrust::transform(X.begin(), X.end(), X.begin(), X.begin(), thrust::multiplies<float>());// X = X*X
   thrust::fill(Y.begin(), Y.end(), 1.0); // Y <- 1.0
   thrust::transform(X.begin(), X.end(), Y.begin(), X.begin(), thrust::plus<float>()); // X = X + 1
   thrust::fill(Y.begin(), Y.end(), 4.0); // Y <- 4.0
   thrust::transform(Y.begin(), Y.end(), X.begin(), Z.begin(), thrust::divides<float>()); // z = 4.0 / X
   thrust::fill(Y.begin(), Y.end(), N);
   thrust::transform(Z.begin(), Z.end(), Y.begin(), Z.begin(), thrust::divides<float>()); // z = z / N

   // Sum the calculation result
   sum = thrust::reduce(Z.begin(), Z.end(), (float)0.0, thrust::plus<float>());

   // End timer
   end_time = clock();
   exec_time = (double)(end_time - start_time)*1000 / CLOCKS_PER_SEC;

   // Print the result
   cout << "N: 2000000.0" << endl;
   cout << "Excution Time: " << exec_time << " ms" << endl;
   cout << "Result: " << sum << endl;

   return 0;
}
```

OUTPUT

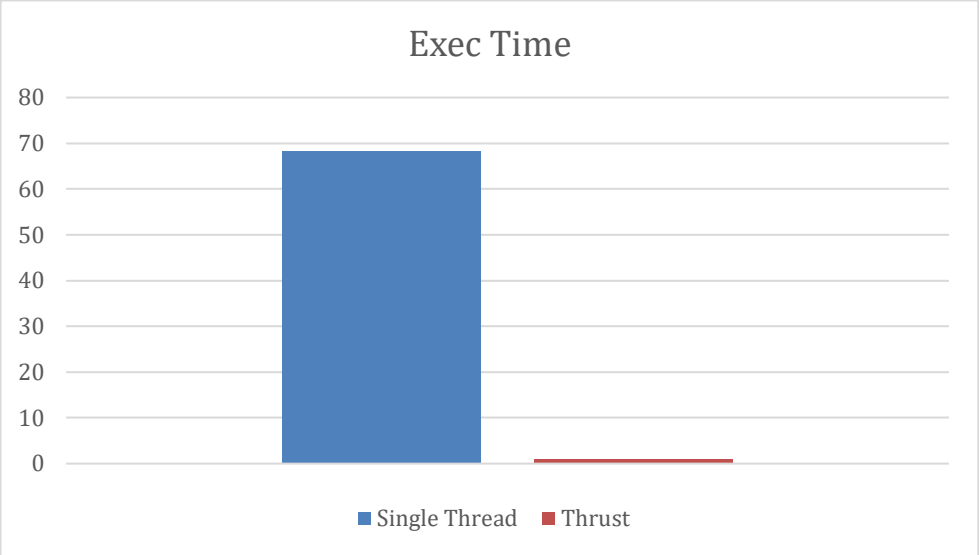
Single Thread Version



Thrust Version



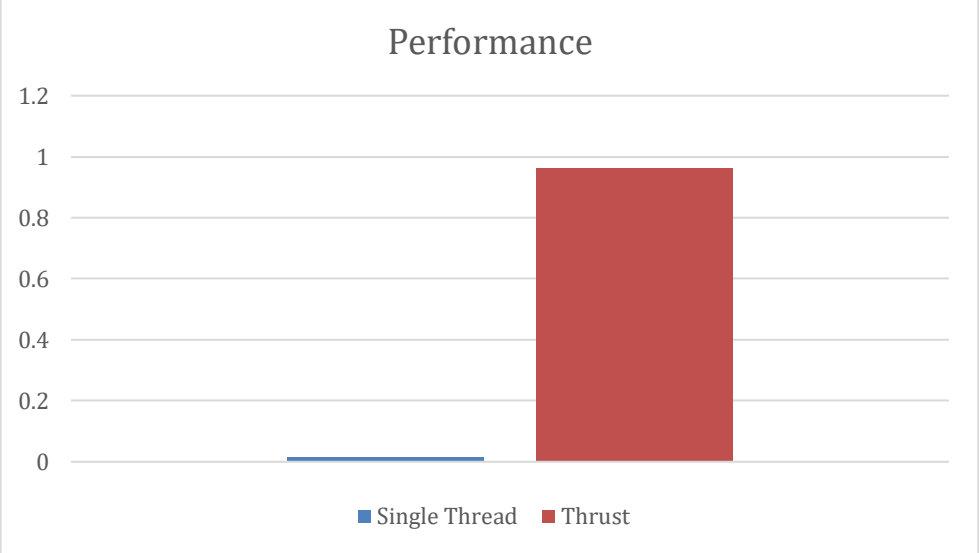
EXPERIMENTAL RESULT



▲ Fig. 1. Exec Time of integration

Type	Exec Time (ms)
Single Thread	68.423
Thrust	1.039

▲ TABLE 1



▲ Fig. 2. Performance of integration

Type	Performance (1/exec time)
Single Thread	0.0146
Thrust	0.9625

▲ TABLE 2

From the result of execution, I could check the software which uses Thrust library shows the better performance. Because the given workload is the same, but the Thrust version finished the calculation in a short time.