2022.1 Multicore Computing, Project #4

Problem 2

Document

소프트웨어학부

20176342 송민준

- (a) execution environment (OS/CPU/GPU type or Colab?)

I use my Computer to run Project #4 code.

CPU : AMD Ryzen 5 5600X Six-Core Processor (12 CPUs), 3.7GHz

Memory : DDR4 16384MB RAM

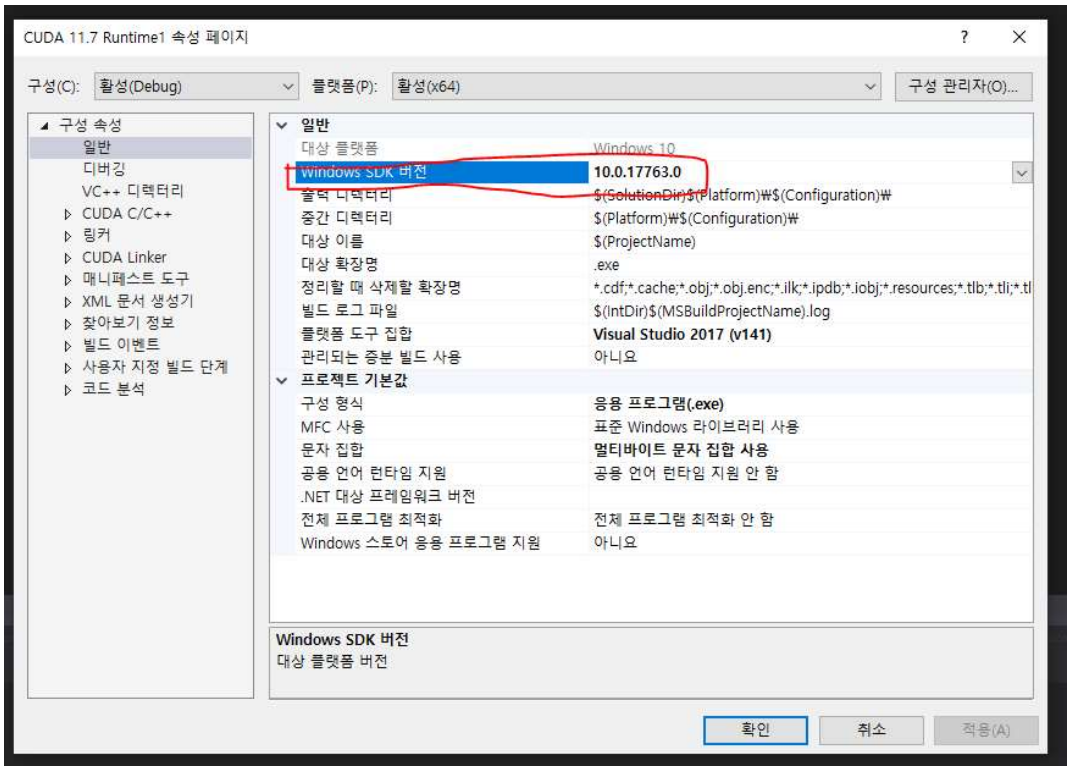
OS : Windows 10

(b) how to compile

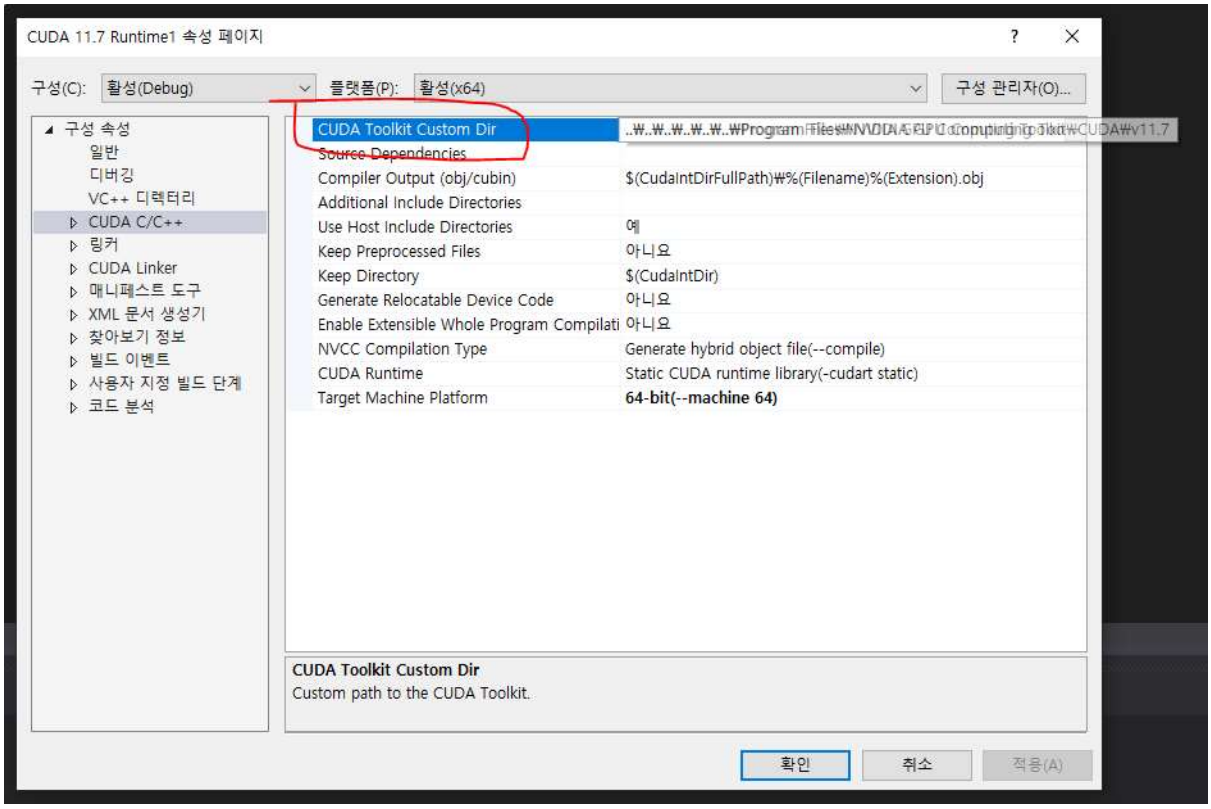
- thrust_ex.cu

Use Microsoft visual studio Community 2017 version.

Install cuda 11.7 version



Set Windows SDK version to 10.0.17763.0



Set CUDA Toolkit Custom Dir like above picture

1>C:₩Users₩song₩source₩repos₩CUDA 11.7 Runtime1₩CUDA 11.7 Runtime1>"..₩..₩..₩..₩..₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v11.7₩bin₩nvcc.exe" -gencode=arch=compute_52,code=₩"sm_52,compute_52₩" --use-local-env -ccbin "C:₩Program Files (x86)₩Microsoft Visual Studio₩2017₩Community₩VC₩Tools₩MSVC₩14.16.27023₩bin₩HostX86₩x64" -x cu -I"..₩..₩..₩..₩..₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v11.7₩include" -I"..₩..₩..₩..₩..₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v11.7₩include" -G --keep-dir x64₩Debug -maxrregcount=0 --machine 64 --compile -cudart static -g -DWIN32 -DWIN64 -D_DEBUG -D_CONSOLE -D_MBCS -Xcompiler "/EHsc /W3 /nologo /Od /Fdx64₩Debug₩vc141.pdb /FS /Zi /RTC1 /MDd " -o "C:₩Users₩song₩source₩repos₩CUDA 11.7 Runtime1₩CUDA 11.7 Runtime1₩x64₩Debug₩cuda_ray.cu.obj" "C:₩Users₩song₩source₩repos₩CUDA 11.7 Runtime1₩CUDA 11.7 Runtime1₩cuda_ray.cu"

And compile project like above. Nvcc ~~~ thrust_ex.cu

(c) how to execute

'thrust_ex.exe' (no parameter)

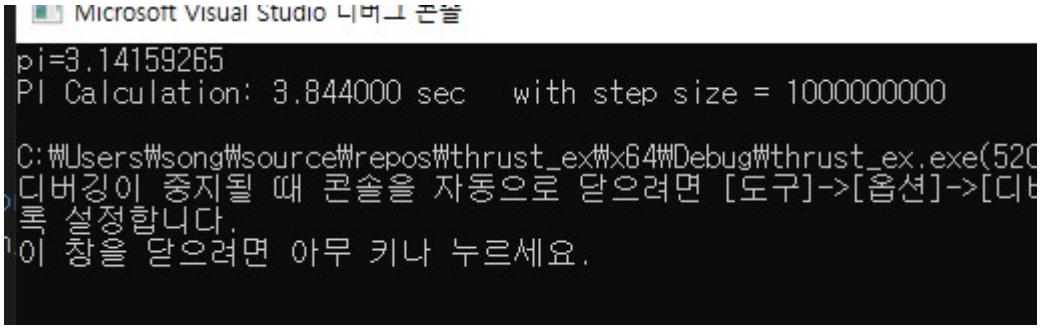
Entire source code

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#include <omp.h>
#include <cuda.h>

#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/transform_reduce.h>

#define CUDA 0
#define OPENMP 1

#define rnd( x ) (x * rand() / RAND_MAX)
#define INF 2e10f

#define STEPS 1000000000
#define STEP 1/STEPS


struct saxpy_functor // define functor to calculate pie
{

    __host__ __device__
        double operator()(const int& x) const {
        double temp = (x + 0.5)*STEP;

        return (4.0 / (1.0 + temp * temp));
    }
};

int main()
{
    clock_t start, end;
```

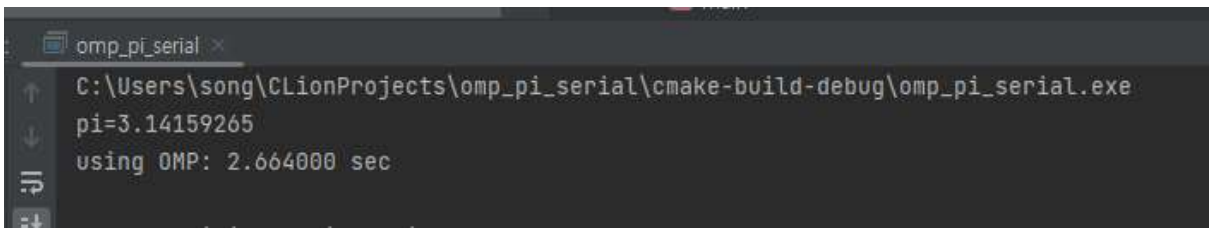
```
    start = clock();
    double x, pi, sum = 0.0;

    thrust::counting_iterator<int> a(0); //define counting iterator
    double result = thrust::transform_reduce(a,a+STEPS,saxpy_functor(), 0.0,
thrust::plus<double>()); // reduce all result (add) with functor

    pi = result * STEP; // calculate pi
    printf("pi=%.8lf\n", pi);

    end = clock();//measure program execution time

    printf("PI Calculation: %lf sec   with step size = %d \n", (double)(end -
start) / 1000.0,STEPS );

}
```

Program output result

- thrust_ex.cu



- omp_pi_serial.c



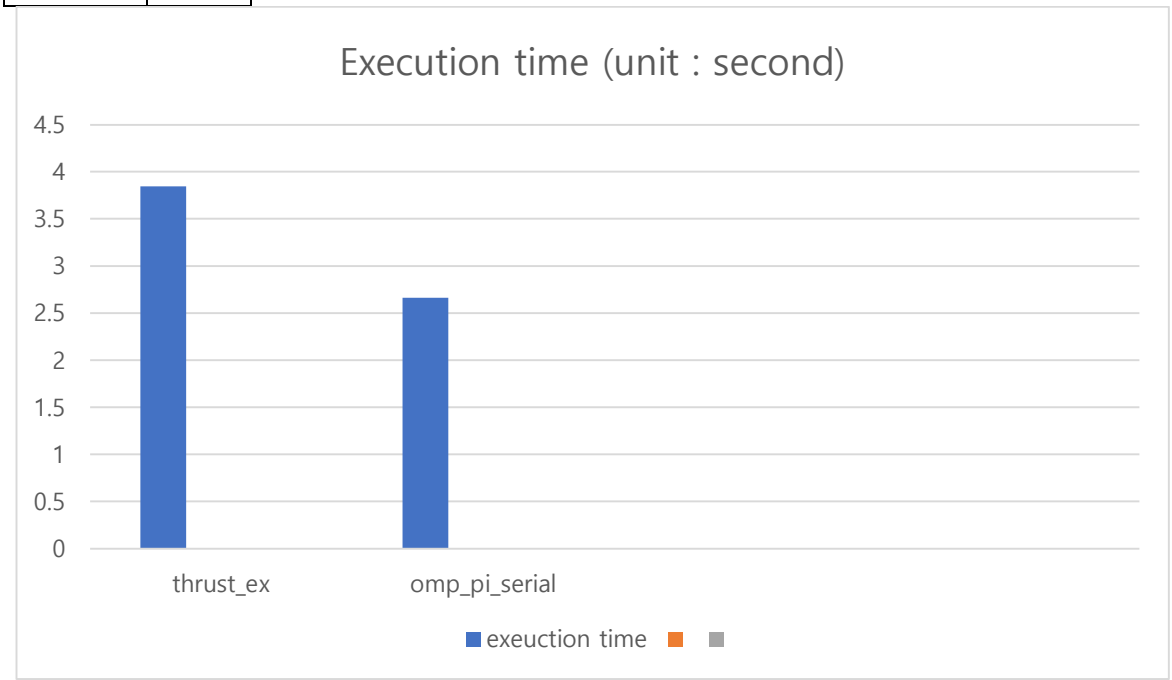
Experimental results

Thrust_ex

Exec time(unit: sec)	
	3.844

Omp_pi_serial

Exec time(unit: sec)	1
	2.664



Interpretation/explanation

We calculated the pie value with n as 1 billion and calculated it through integration using openmp and 'cuda thrust'.

I think the reason why the performance of the open mp is better is because my cpu is the latest model, but I think it came out like this because the gpu is old.

And in the case of cuda thrust, the overhead of moving the value from the host vector to the device vector and moving the computation result back from the device vector to the host vector is too large, so if it is optimized, I think we can reduce the bus bottleneck and speed it up.