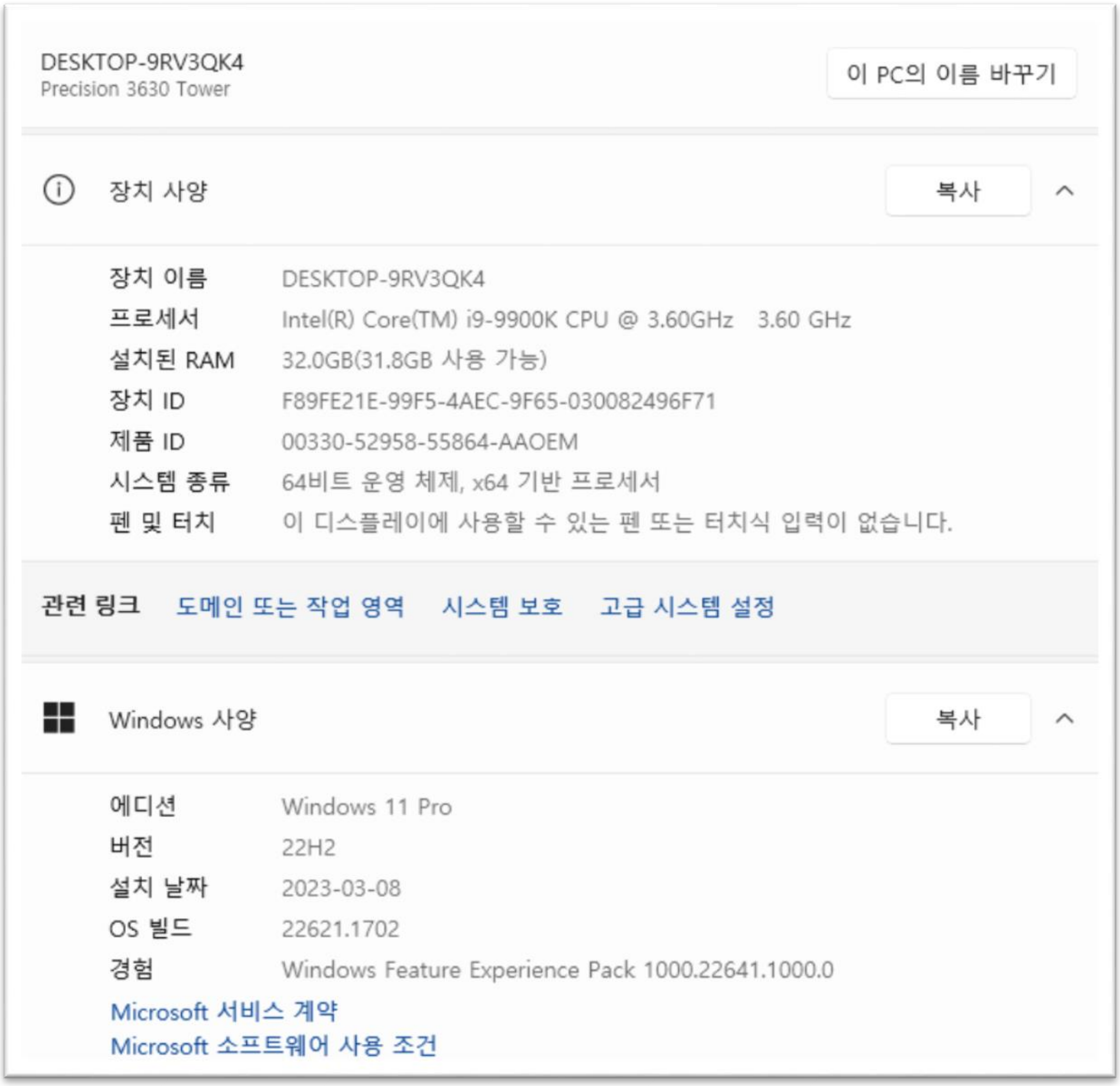
Problem1 Report

Student NO: 20183784
Student Name: 노현진

[Execution environment]

- **OS**



DESKTOP-9RV3QK4
Precision 3630 Tower

장치 사양

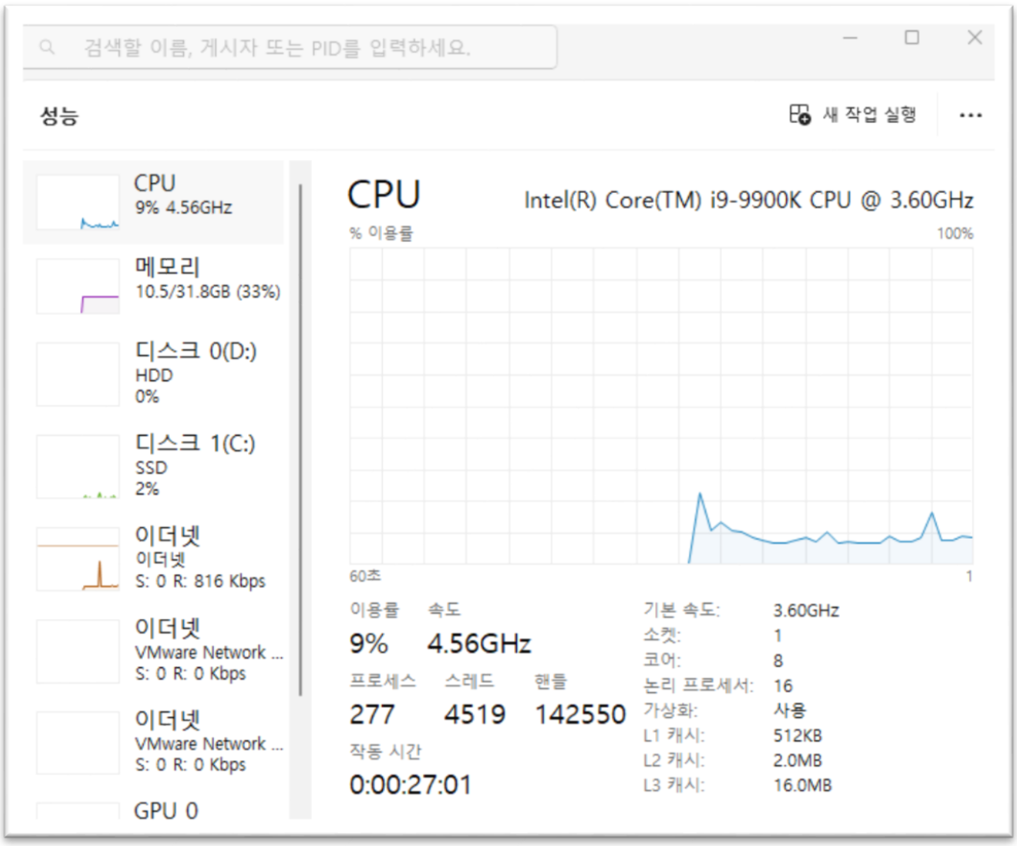
장치 이름	DESKTOP-9RV3QK4
프로세서	Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz 3.60 GHz
설치된 RAM	32.0GB(31.8GB 사용 가능)
장치 ID	F89FE21E-99F5-4AEC-9F65-030082496F71
제품 ID	00330-52958-55864-AAOEM
시스템 종류	64비트 운영 체제, x64 기반 프로세서
펜 및 터치	이 디스플레이에 사용할 수 있는 펜 또는 터치식 입력이 없습니다.

Windows 사양

에디션	Windows 11 Pro
버전	22H2
설치 날짜	2023-03-08
OS 빌드	22621.1702
경험	Windows Feature Experience Pack 1000.22641.1000.0

OS type: Windows 11

- **CPU**



검색할 이름, 게시자 또는 PID를 입력하세요.

성능 새 작업 실행

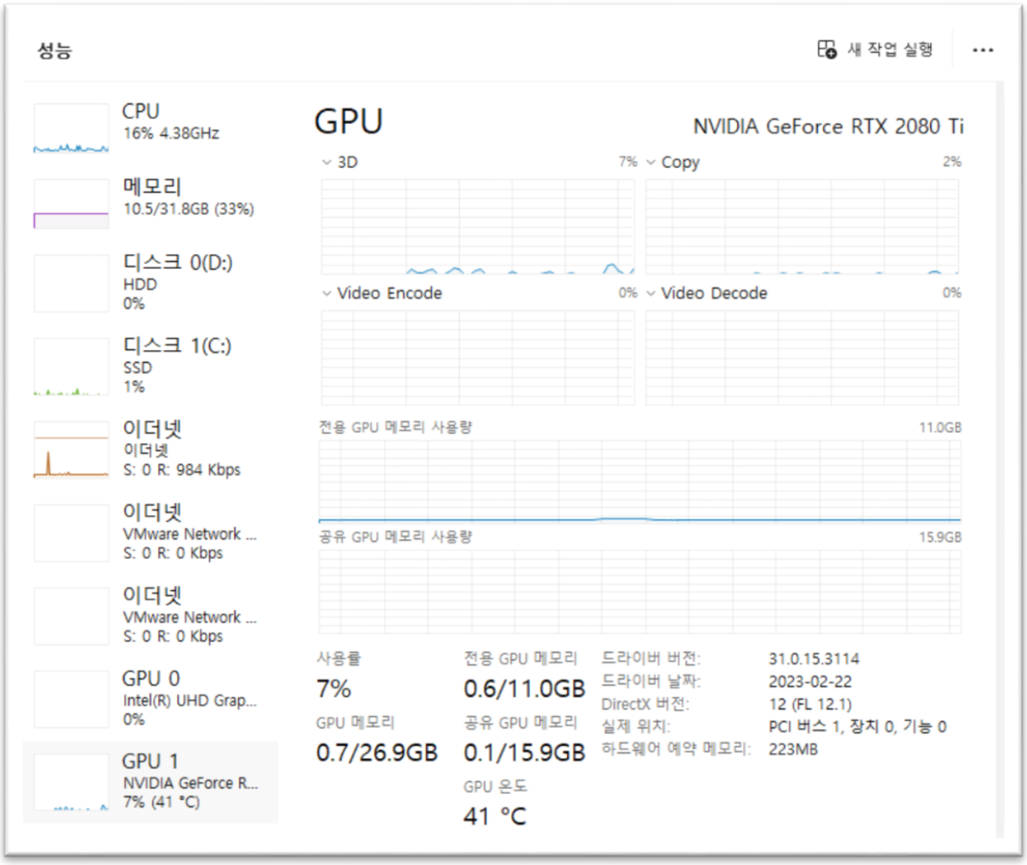
CPU 9% 4.56GHz
메모리 10.5/31.8GB (33%)
디스크 0(D:) HDD 0%
디스크 1(C:) SSD 2%
이더넷 이더넷 S: 0 R: 816 Kbps
이더넷 VMware Network ... S: 0 R: 0 Kbps
이더넷 VMware Network ... S: 0 R: 0 Kbps
GPU 0

CPU Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz

이용률	속도		기본 속도:	3.60GHz
9%	4.56GHz		소켓:	1
프로세스	스레드	핸들	코어:	8
277	4519	142550	논리 프로세서:	16
작동 시간			가상화:	사용
0:00:27:01			L1 캐시:	512KB
			L2 캐시:	2.0MB
			L3 캐시:	16.0MB

CPU type: Intel® Core™ i9-9900K CPU

- **GPU**



성능 새 작업 실행

CPU 16% 4.38GHz
메모리 10.5/31.8GB (33%)
디스크 0(D:) HDD 0%
디스크 1(C:) SSD 1%
이더넷 이더넷 S: 0 R: 984 Kbps
이더넷 VMware Network ... S: 0 R: 0 Kbps
이더넷 VMware Network ... S: 0 R: 0 Kbps
GPU 0 Intel(R) UHD Grap... 0%
GPU 1 NVIDIA GeForce R... 7% (41 °C)

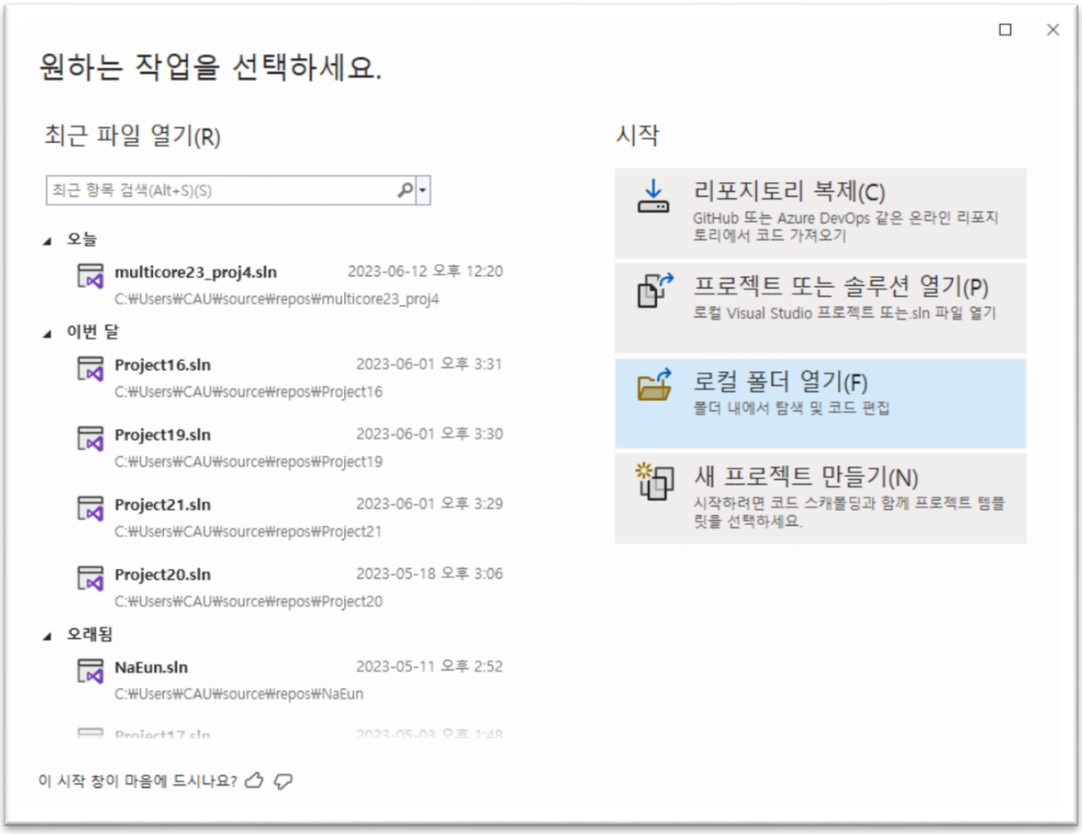
GPU NVIDIA GeForce RTX 2080 Ti

사용률	전용 GPU 메모리	드라이버 버전:	31.0.15.3114
7%	0.6/11.0GB	드라이버 날짜:	2023-02-22
GPU 메모리	공유 GPU 메모리	DirectX 버전:	12 (FL 12.1)
0.7/26.9GB	0.1/15.9GB	실제 위치:	PCI 버스 1, 장치 0, 기능 0
	GPU 온도	하드웨어 예약 메모리:	223MB
	41 °C		

GPU type: NVIDIA GeForce RTX 2080 Ti

[How to compile and execute]

1. Firstly, install Visual Studio 2022.
2. Secondly, install CUDA.
3. After installations, open the Visual Studio 2022.


원하는 작업을 선택하세요.

최근 파일 열기(R)

최근 항목 검색(Alt+S)(S)

오늘

multicore23_proj4.sln 2023-06-12 오후 12:20
C:₩Users₩CAU₩source₩repos₩multicore23_proj4

이번 달

Project16.sln 2023-06-01 오후 3:31
C:₩Users₩CAU₩source₩repos₩Project16

Project19.sln 2023-06-01 오후 3:30
C:₩Users₩CAU₩source₩repos₩Project19

Project21.sln 2023-06-01 오후 3:29
C:₩Users₩CAU₩source₩repos₩Project21

Project20.sln 2023-05-18 오후 3:06
C:₩Users₩CAU₩source₩repos₩Project20

오래됨

NaEun.sln 2023-05-11 오후 2:52
C:₩Users₩CAU₩source₩repos₩NaEun

Project17.sln 2023-05-03 오후 1:48

시작

리포지토리 복제(C)
GitHub 또는 Azure DevOps 같은 온라인 리포지토리에서 코드 가져오기

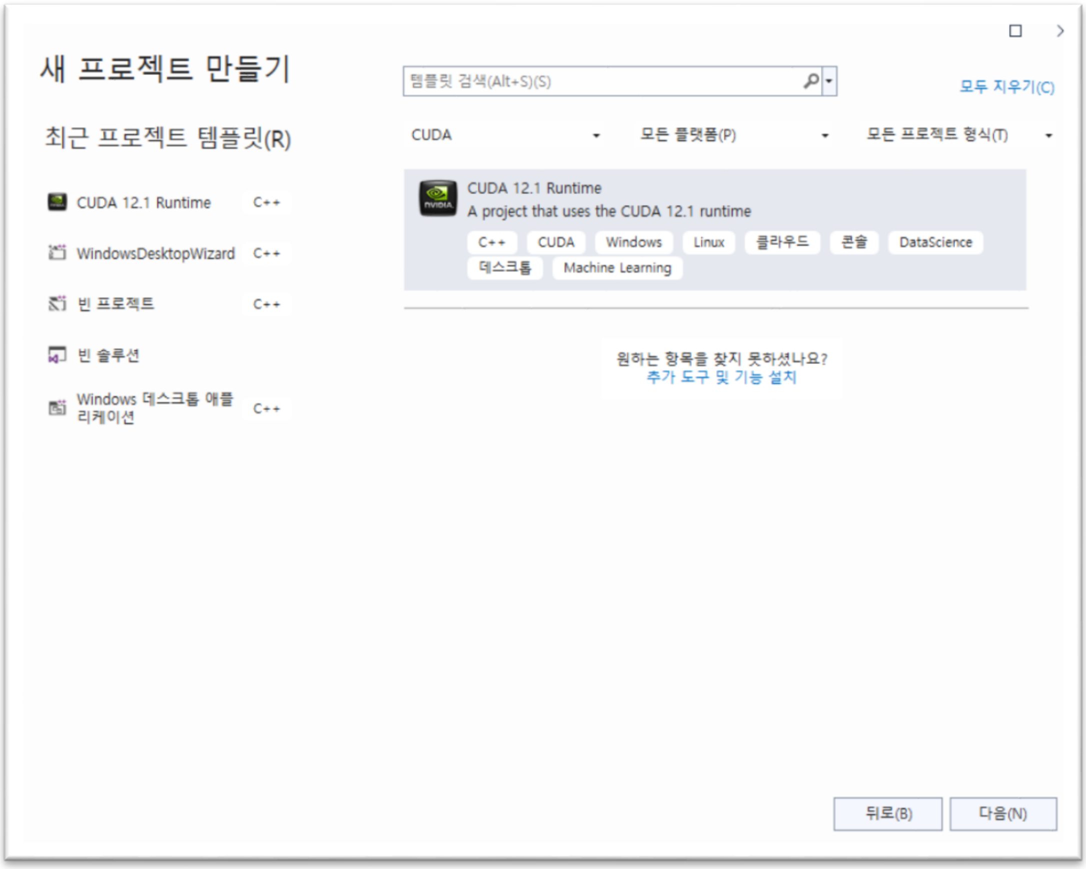
프로젝트 또는 솔루션 열기(P)
로컬 Visual Studio 프로젝트 또는 .sln 파일 열기

로컬 폴더 열기(F)
폴더 내에서 탐색 및 코드 편집

새 프로젝트 만들기(N)
시작하려면 코드 스캐폴딩과 함께 프로젝트 템플릿을 선택하세요.

이 시작 창이 마음에 드시나요?

4. Create new project. The project should be CUDA Run time project.


새 프로젝트 만들기

최근 프로젝트 템플릿(R)

CUDA 12.1 Runtime C++
WindowsDesktopWizard C++
빈 프로젝트 C++
빈 솔루션
Windows 데스크톱 애플리케이션 C++

템플릿 검색(Alt+S)(S)

모두 지우기(C)

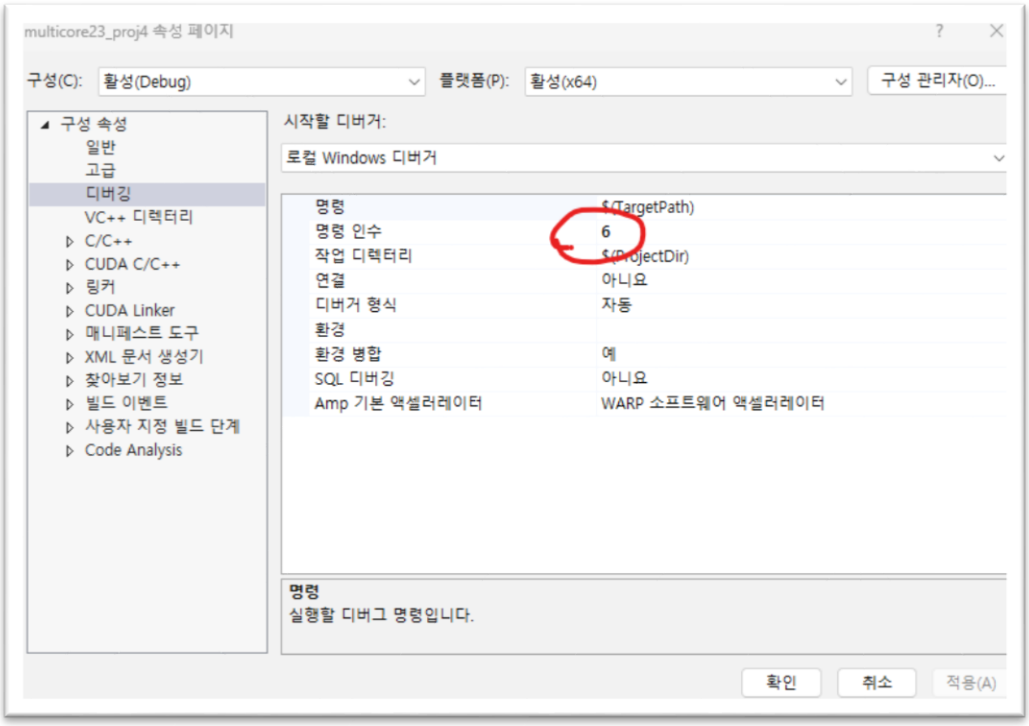
CUDA | 모든 플랫폼(P) | 모든 프로젝트 형식(T)

CUDA 12.1 Runtime
A project that uses the CUDA 12.1 runtime
C++ CUDA Windows Linux 클라우드 콘솔 DataScience 데스크톱 Machine Learning

원하는 항목을 찾지 못하셨나요?
추가 도구 및 기능 설치

뒤로(B) 다음(N)

5. Put the source code that you want to run into the project.
6. Before execution, set the parameter and make OpenMP available.

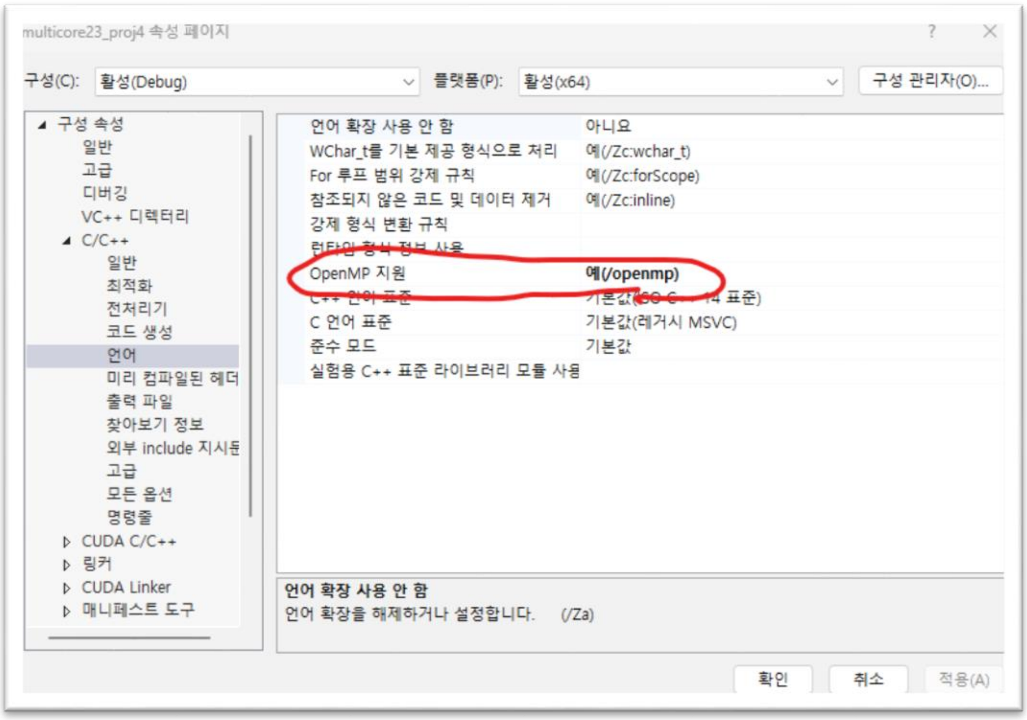

multicore23_proj4 속성 페이지

구성(C): 활성(Debug) 플랫폼(P): 활성(x64) 구성 관리자(O)...

시작할 디버거: 로컬 Windows 디버거

명령	$(TargetPath)
명령 인수	6
작업 디렉터리	$(ProjectDir)
연결	아니요
디버거 형식	자동
환경	
환경 병합	예
SQL 디버깅	아니요
Amp 기본 액셀러레이터	WARP 소프트웨어 액셀러레이터

명령
실행할 디버그 명령입니다.

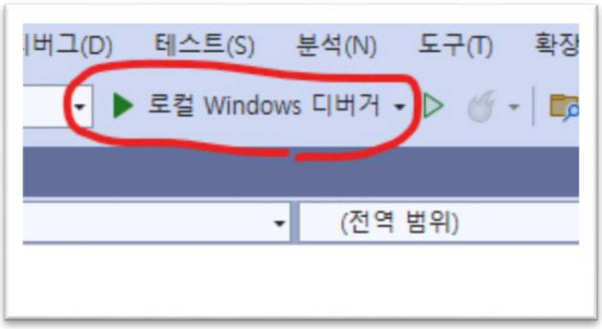

multicore23_proj4 속성 페이지

구성(C): 활성(Debug) 플랫폼(P): 활성(x64) 구성 관리자(O)...

언어 확장 사용 안 함	아니요
WChar_t를 기본 제공 형식으로 처리	예(/Zc:wchar_t)
For 루프 범위 강제 규칙	예(/Zc:forScope)
참조되지 않은 코드 및 데이터 제거	예(/Zc:inline)
강제 형식 변환 규칙	
런타임 형식 정보 사용	
OpenMP 지원	**예(/openmp)**
C++ 언어 표준	기본값(ISO C++14 표준)
C 언어 표준	기본값(레거시 MSVC)
준수 모드	기본값
실험용 C++ 표준 라이브러리 모듈 사용	

언어 확장 사용 안 함
언어 확장을 해제하거나 설정합니다. (/Za)

7. Finally, run the source code.



[Source Code]

- openmp_ray.cpp

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

#define SPHERES 20
#define rnd( x ) (x * rand() / RAND_MAX)
#define INF 2e10f
#define DIM 2048

struct Sphere {
	float r, b, g;
	float radius;
	float x, y, z;
	float hit(float ox, float oy, float* n) {
		float dx = ox - x;
		float dy = oy - y;
		if (dx * dx + dy * dy < radius * radius) {
			float dz = sqrtf(radius * radius - dx * dx - dy * dy);
			*n = dz / sqrtf(radius * radius);
			return dz + z;
		}
		return -INF;
	}
};

void kernel(int x, int y, Sphere* s, unsigned char* ptr) {
	int offset = x + y * DIM;
	float ox = (x - DIM / 2);
	float oy = (y - DIM / 2);

	//printf("x:%d, y:%d, ox:%f, oy:%f₩n",x,y,ox,oy);

	float r = 0, g = 0, b = 0;
	float maxz = -INF;

	for (int i = 0; i < SPHERES; i++) {
		float n;
		float t = s[i].hit(ox, oy, &n);
		if (t > maxz) {
			float fscale = n;
			r = s[i].r * fscale;
			g = s[i].g * fscale;
			b = s[i].b * fscale;
			maxz = t;
		}
	}

	ptr[offset * 4 + 0] = (int)(r * 255);
	ptr[offset * 4 + 1] = (int)(g * 255);
	ptr[offset * 4 + 2] = (int)(b * 255);
	ptr[offset * 4 + 3] = 255;
}
```

```
void ppm_write(unsigned char* bitmap, int xdim, int ydim, FILE* fp) {
        int i, x, y;
        fprintf(fp, "P3₩n");
        fprintf(fp, "%d %d₩n", xdim, ydim);
        fprintf(fp, "255₩n");

        for (y = 0; y < ydim; y++) {
                for (x = 0; x < xdim; x++) {
                        i = x + y * xdim;
                        fprintf(fp, "%d %d %d ", bitmap[4 * i], bitmap[4 * i + 1], bitmap[4 * i + 2]);
                }
                fprintf(fp, "₩n");
        }
}

int main(int argc, char* argv[]) {
        int i;
        int num_threads;
        int x, y;
        unsigned char* bitmap;
        double start_time, end_time;
        Sphere* temp_s = (Sphere*)malloc(sizeof(Sphere) * SPHERES);
        FILE* fp = fopen("result.ppm", "w");
        srand(time(NULL));

        if (argc != 2) {
                printf("> a.out [number of threads]₩n");
                printf("for example, '> a.out 8' means executing OpenMP with 8 threads₩n");
                exit(0);
        }
        else {
                num_threads = atoi(argv[1]);
        }

        omp_set_num_threads(num_threads);
        start_time = omp_get_wtime();

        for (i = 0; i < SPHERES; i++) {
                temp_s[i].r = rnd(1.0f);
                temp_s[i].g = rnd(1.0f);
                temp_s[i].b = rnd(1.0f);
                temp_s[i].x = rnd(2000.0f) - 1000;
                temp_s[i].y = rnd(2000.0f) - 1000;
                temp_s[i].z = rnd(2000.0f) - 1000;
                temp_s[i].radius = rnd(200.0f) + 40;
        }

        bitmap = (unsigned char*)malloc(sizeof(unsigned char) * DIM * DIM * 4);

        #pragma omp parallel for default(shared) private(x, y)
        for (x = 0; x < DIM; x++)
                for (y = 0; y < DIM; y++) kernel(x, y, temp_s, bitmap);

        end_time = omp_get_wtime();

        ppm_write(bitmap, DIM, DIM, fp);

        fclose(fp);
        free(bitmap);
        free(temp_s);
```

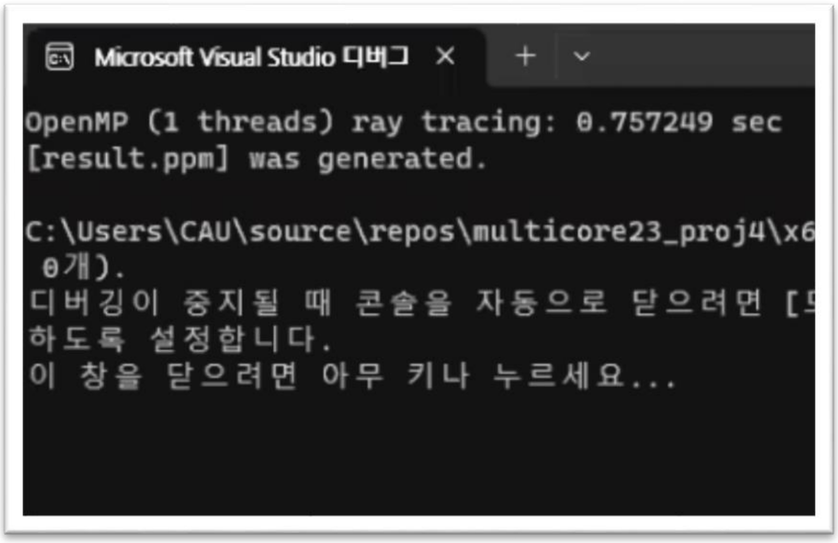
```
	printf("OpenMP (%d threads) ray tracing: %lf sec\n", num_threads, end_time - start_time);
	printf("[result.ppm] was generated.\n");
	return 0;
}
```

- **cuda_ray.cu**

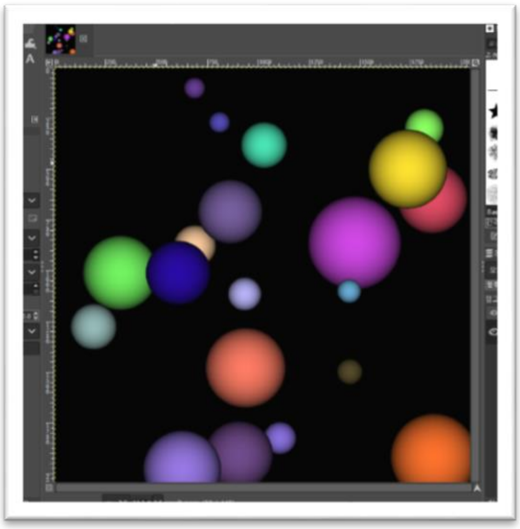
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <chrono>
#include <iostream>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

using namespace std;
using namespace std::chrono;

#define SPHERES 20
#define rnd( x ) (x * rand() / RAND_MAX)
#define INF 2e10f
#define DIM 2048

struct Sphere {
	float r, b, g;
	float radius;
	float x, y, z;
	float hit(float ox, float oy, float* n) {
		float dx = ox - x;
		float dy = oy - y;
		if (dx * dx + dy * dy < radius * radius) {
			float dz = sqrtf(radius * radius - dx * dx - dy * dy);
			*n = dz / sqrtf(radius * radius);
			return dz + z;
		}
		return -INF;
	}
};

__global__ void kernel(Sphere* s, unsigned char* ptr) {
	int x = threadIdx.x + blockIdx.x * blockDim.x;
	int y = threadIdx.y + blockIdx.y * blockDim.y;
	int offset = x + y * DIM;
	float ox = (x - DIM / 2);
	float oy = (y - DIM / 2);

	//printf("x:%d, y:%d, ox:%f, oy:%f\n", x, y, ox, oy);

	int i;
	float r = 0, g = 0, b = 0;
	float maxz = -INF;
	for (i = 0; i < SPHERES; i++) {
		float n;
		//float t = s[i].hit(ox, oy, &n);
		float t;
		float dx = ox - s[i].x;
		float dy = oy - s[i].y;
		float radius = s[i].radius;
		if (dx * dx + dy * dy < radius * radius) {
			float dz = sqrtf(radius * radius - dx * dx - dy * dy);
			n = dz / sqrtf(radius * radius);
			t = dz + s[i].z;
		}
		else {
			t = -INF;
		}
		if (t > maxz) {
			float fscale = n;
			r = s[i].r * fscale;
			g = s[i].g * fscale;
			b = s[i].b * fscale;
			maxz = t;
		}
	}

	ptr[offset * 4 + 0] = (int)(r * 255);
	ptr[offset * 4 + 1] = (int)(g * 255);
	ptr[offset * 4 + 2] = (int)(b * 255);
	ptr[offset * 4 + 3] = 255;
}

void ppm_write(unsigned char* bitmap, int xdim, int ydim, FILE* fp) {
	int i, x, y;
	fprintf(fp, "P3₩n");
	fprintf(fp, "%d %d₩n", xdim, ydim);
	fprintf(fp, "255₩n");

	for (y = 0; y < ydim; y++) {
		for (x = 0; x < xdim; x++) {
			i = x + y * xdim;
			fprintf(fp, "%d %d %d ", bitmap[4 * i], bitmap[4 * i + 1], bitmap[4 * i + 2]);
		}
		fprintf(fp, "₩n");
	}
}

int main() {
	int i;
	Sphere* temp_s;
	Sphere* d_temp_s;
	unsigned char* bitmap;
	unsigned char* d_bitmap;
	int size1 = sizeof(Sphere) * SPHERES;
	int size2 = sizeof(unsigned char) * DIM * DIM * 4;
	FILE* fp = fopen("result.ppm", "w");
	srand(time(NULL));

	// Allocate space for device copies of temp_s and bitmap
	cudaMalloc((void**)&d_temp_s, size1);
	cudaMalloc((void**)&d_bitmap, size2);
	// Allocate space for host copies of temp_s and bitmap
	temp_s = (Sphere*)malloc(size1);
	bitmap = (unsigned char*)malloc(size2);

	// Setup initial values
	for (i = 0; i < SPHERES; i++) {
		temp_s[i].r = rnd(1.0f);
		temp_s[i].g = rnd(1.0f);
		temp_s[i].b = rnd(1.0f);
		temp_s[i].x = rnd(2000.0f) - 1000;
		temp_s[i].y = rnd(2000.0f) - 1000;
		temp_s[i].z = rnd(2000.0f) - 1000;
		temp_s[i].radius = rnd(200.0f) + 40;
	}

	auto start_time = high_resolution_clock::now();

	// Copy values to device
	cudaMemcpy(d_temp_s, temp_s, size1, cudaMemcpyHostToDevice);

	// Setup the execution configuration
	dim3 dimBlock(32, 32, 1);
	dim3 dimGrid(64, 64, 1);

	kernel<<<dimGrid, dimBlock>>>(d_temp_s, d_bitmap);

	// Copy result back to host
	cudaMemcpy(bitmap, d_bitmap, size2, cudaMemcpyDeviceToHost);

	auto end_time = high_resolution_clock::now();
	auto duration = duration_cast<microseconds>(end_time - start_time);

	ppm_write(bitmap, DIM, DIM, fp);

	fclose(fp);
	free(bitmap);
	free(temp_s);
	cudaFree(d_bitmap);
	cudaFree(d_temp_s);

	cout << "CUDA ray tracing: " << duration.count() / 1000000.0 << " sec" << endl;
	cout << "[result.ppm] was generated." << endl;
	return 0;
}
```

[Results]

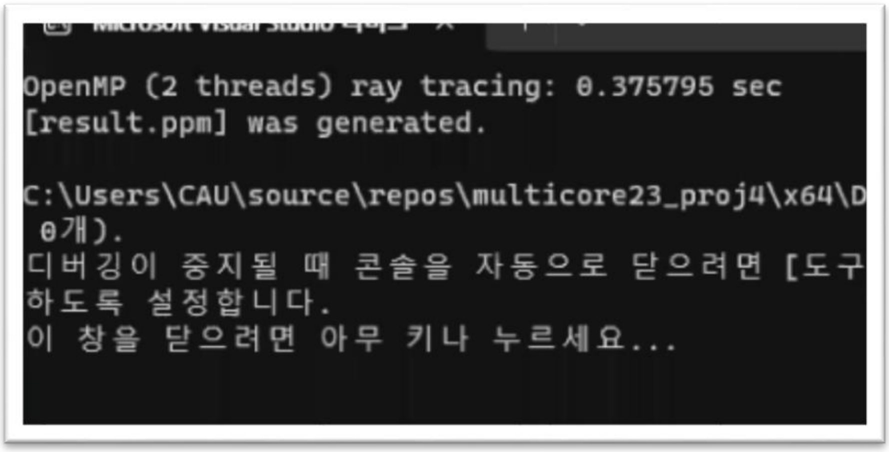
- <u>openmp_ray.cpp</u>
  - 1 thread



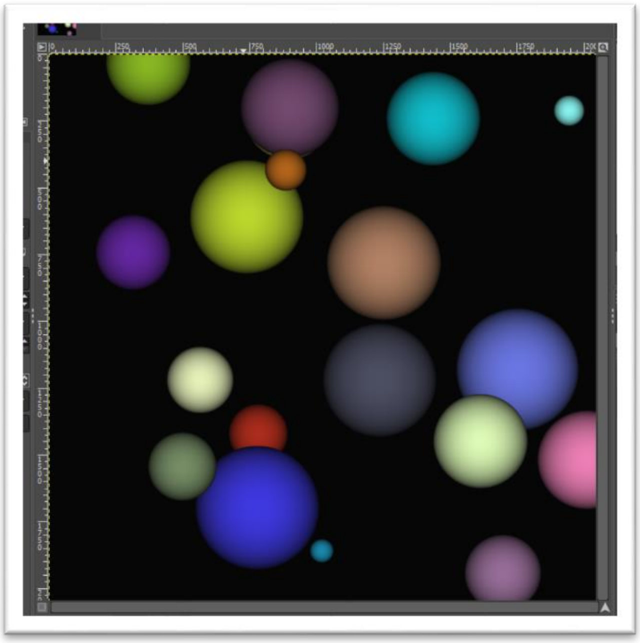
```
OpenMP (1 threads) ray tracing: 0.757249 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x6
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



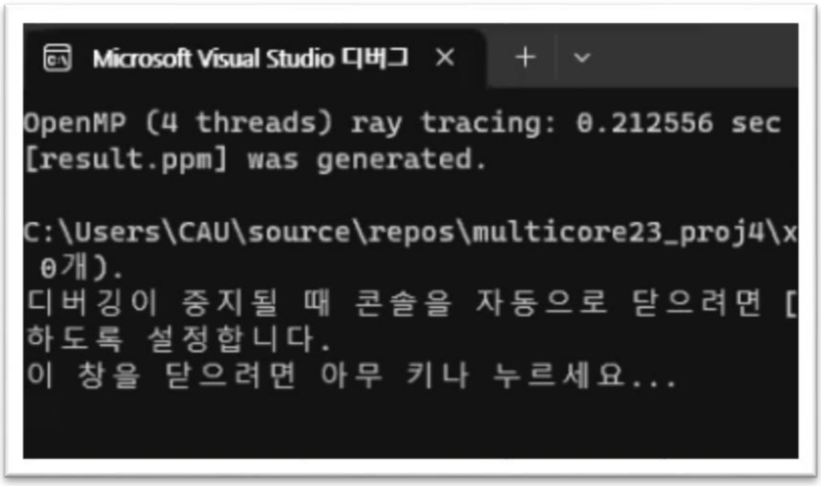
  - 2 threads



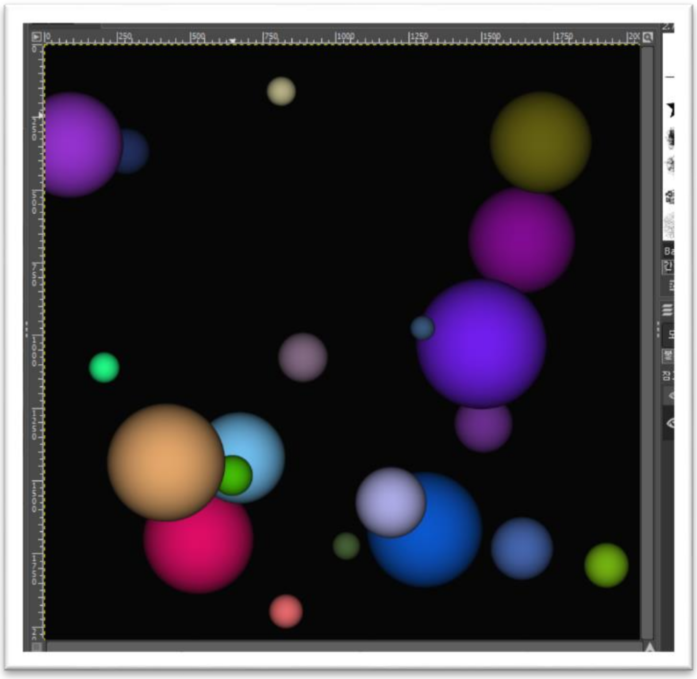
```
OpenMP (2 threads) ray tracing: 0.375795 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x64\D
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



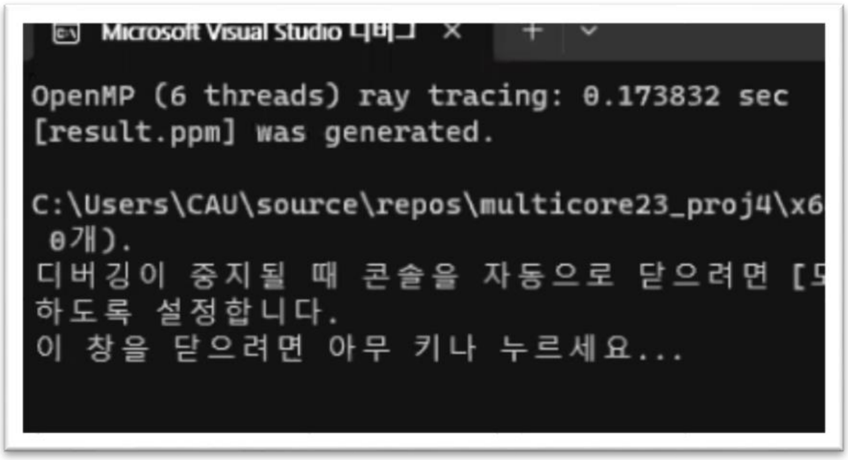
- 4 threads



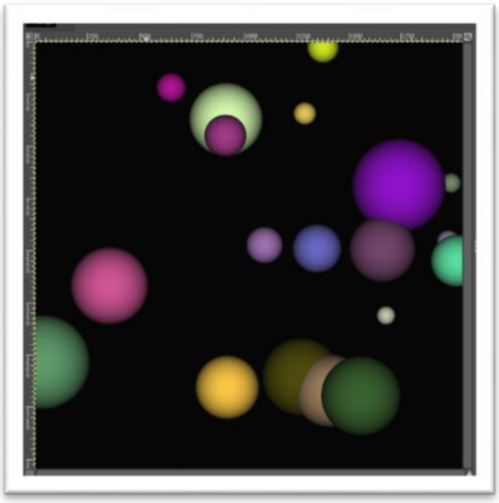
```
OpenMP (4 threads) ray tracing: 0.212556 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



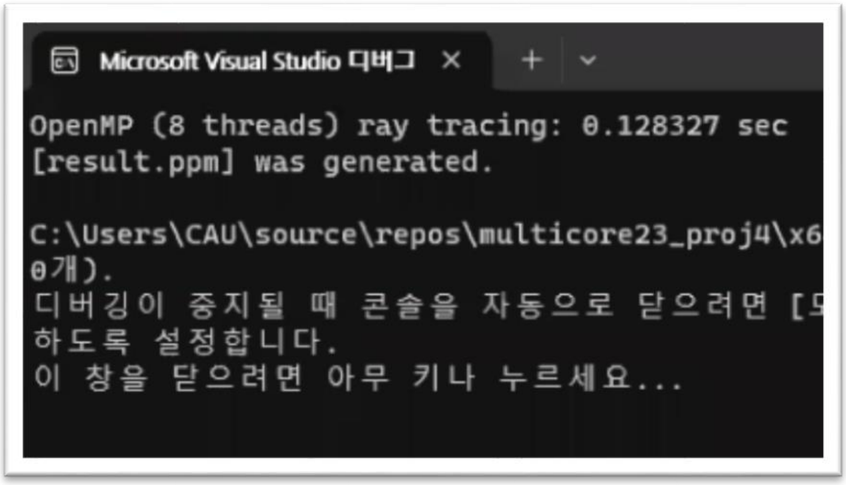
- 6 threads



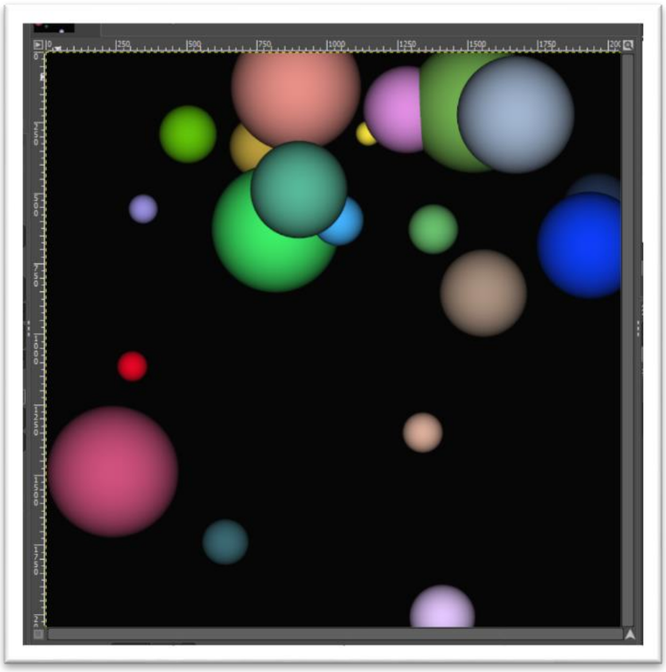
```
OpenMP (6 threads) ray tracing: 0.173832 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x6
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



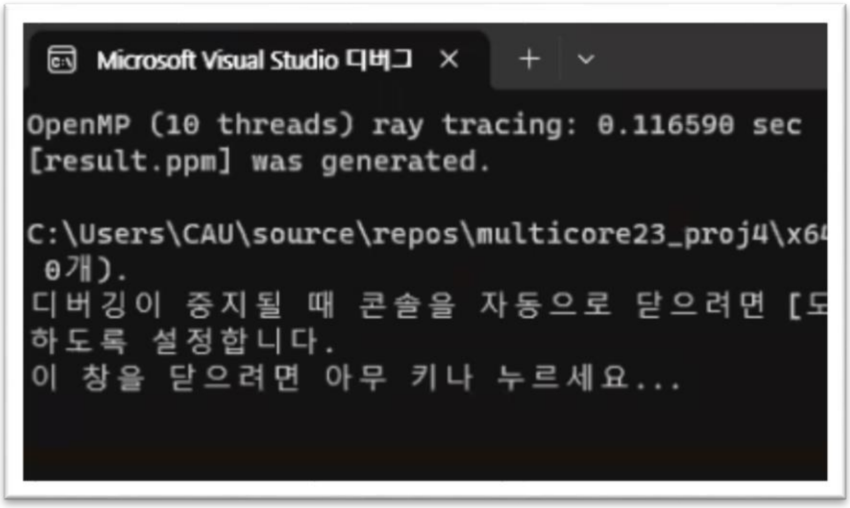
- 8 threads



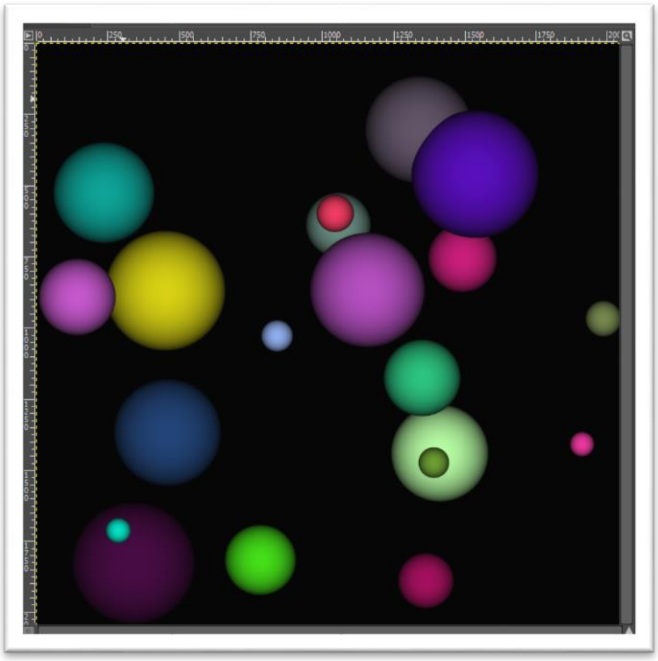
```
OpenMP (8 threads) ray tracing: 0.128327 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x6
0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



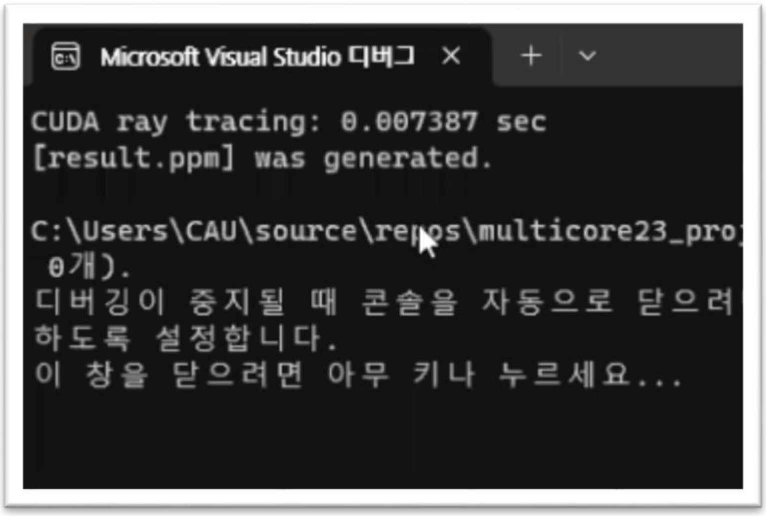
- 10 threads



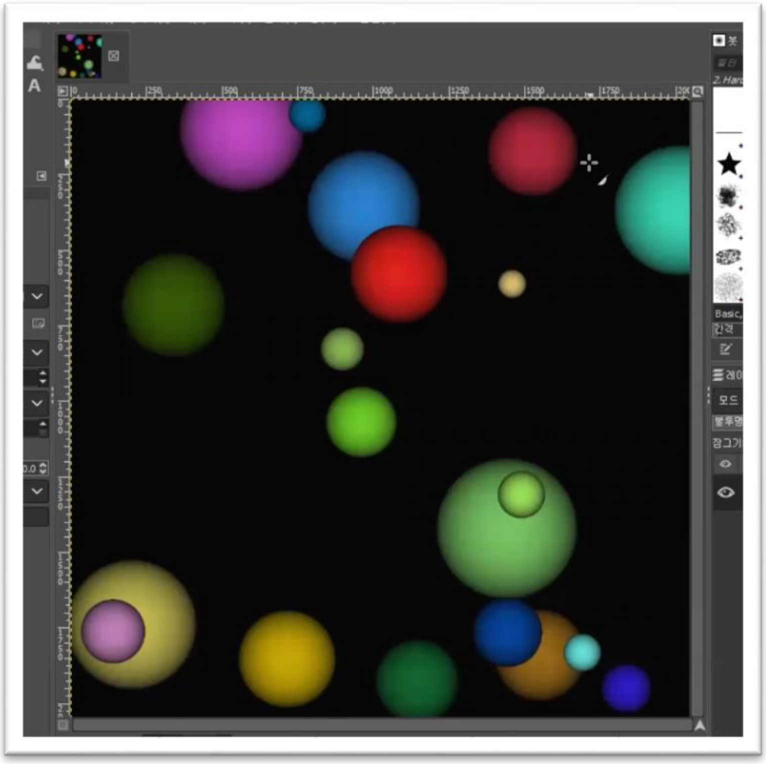
```
OpenMP (10 threads) ray tracing: 0.116590 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj4\x64
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



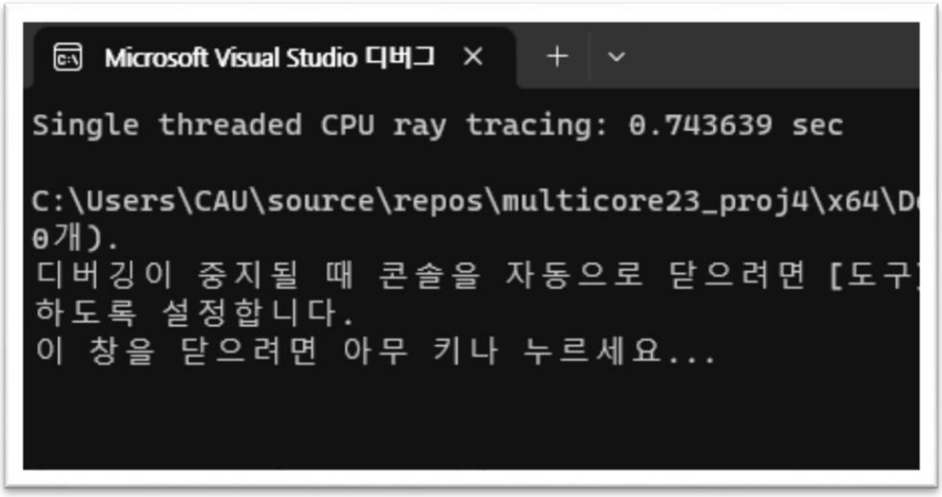
- **cuda_ray.cu**



```
CUDA ray tracing: 0.007387 sec
[result.ppm] was generated.

C:\Users\CAU\source\repos\multicore23_proj
 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



- **raytracing.cpp (Given source code)**



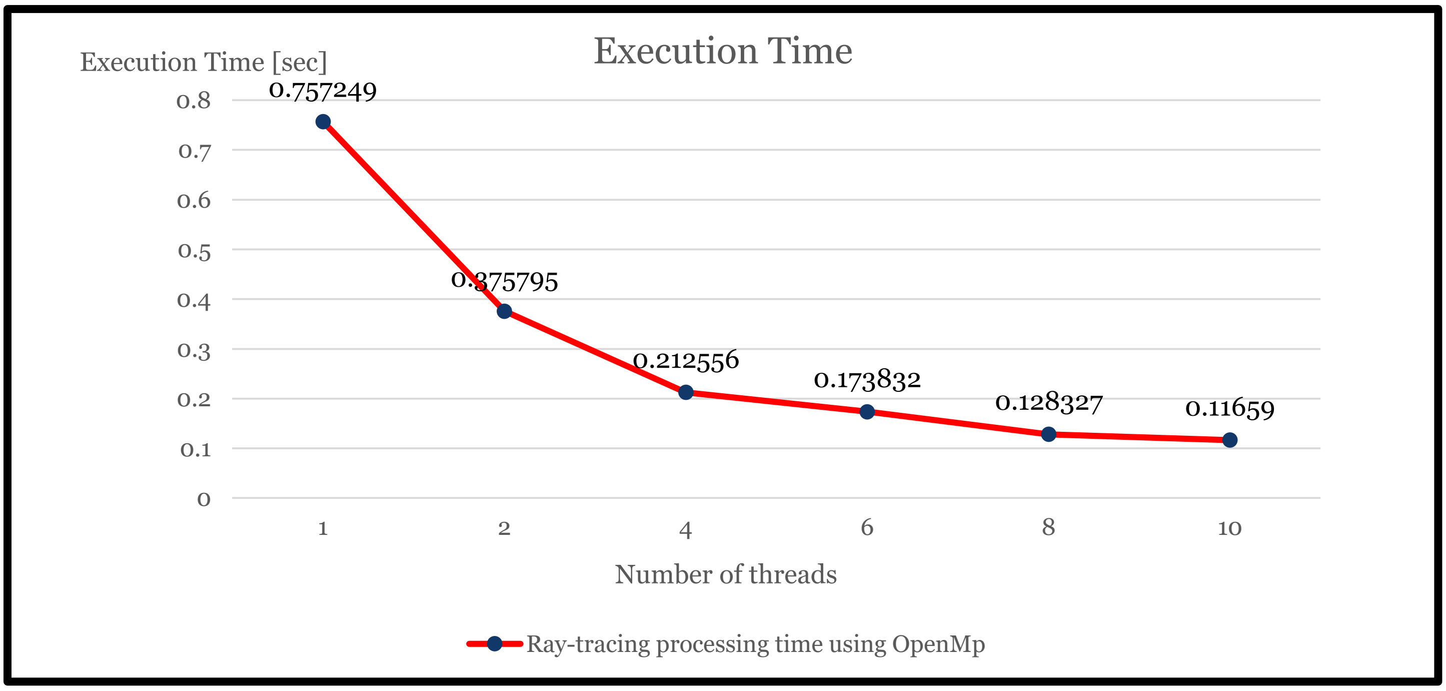
```
Single threaded CPU ray tracing: 0.743639 sec

C:\Users\CAU\source\repos\multicore23_proj4\x64\D
0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

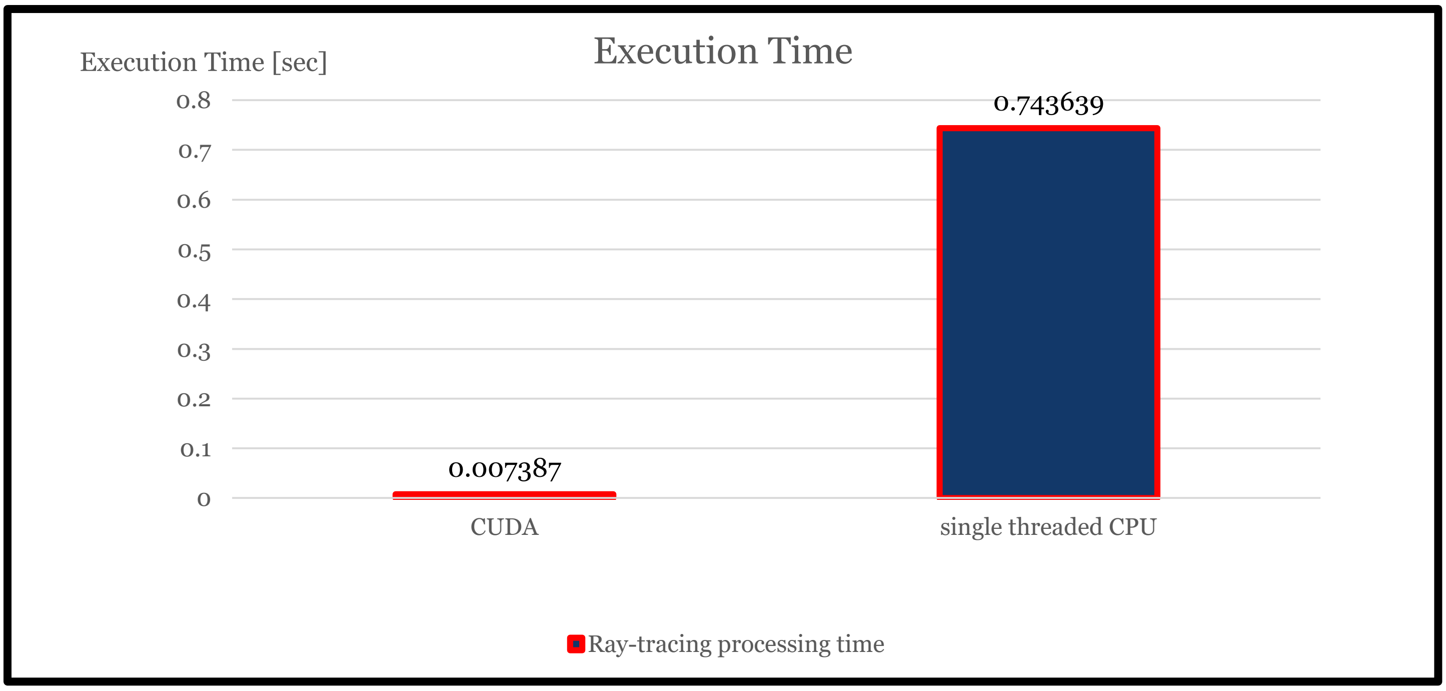
- **Execution Time**

OpenMp	1	2	4	6	8	10
Exec time	0.757249 sec	0.375795 sec	0.212556 sec	0.173832 sec	0.128327 sec	0.116590 sec

CUDA	
Exec time	0.007387 sec

Single threaded CPU	
Exec time	0.743639 sec



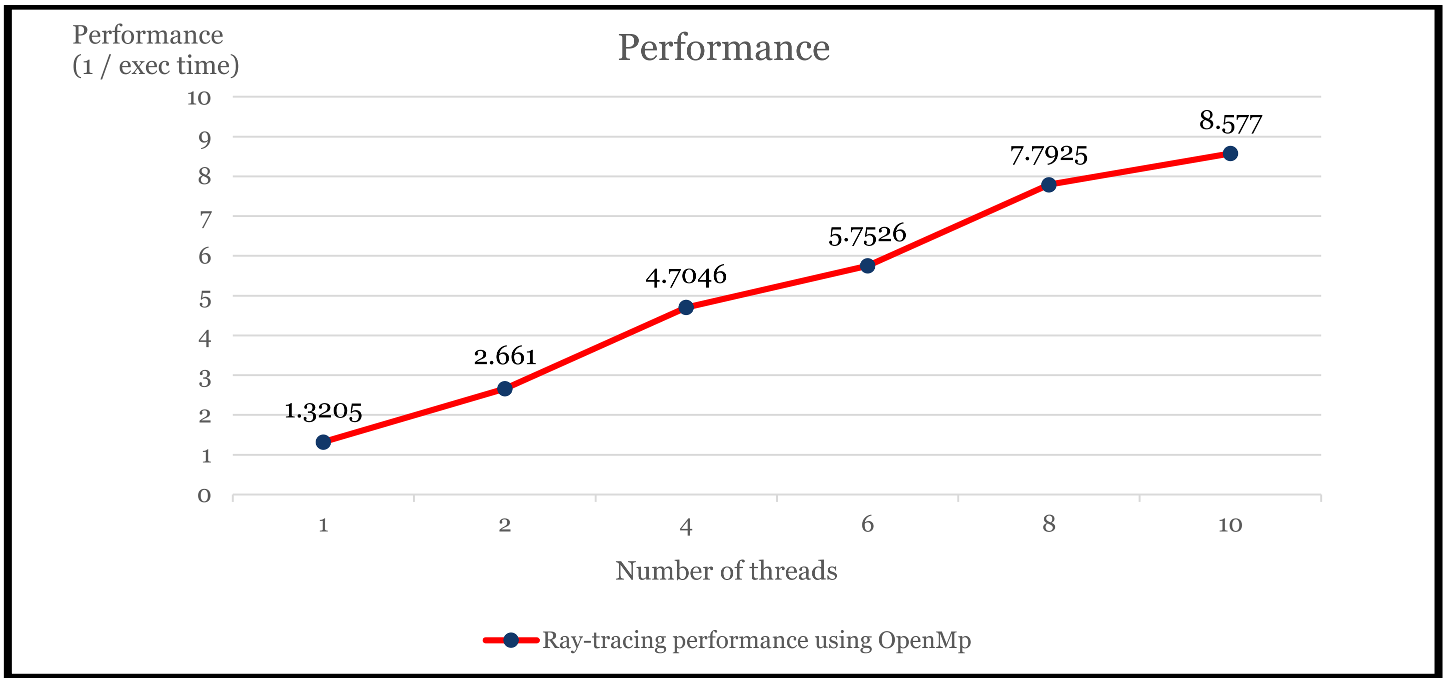


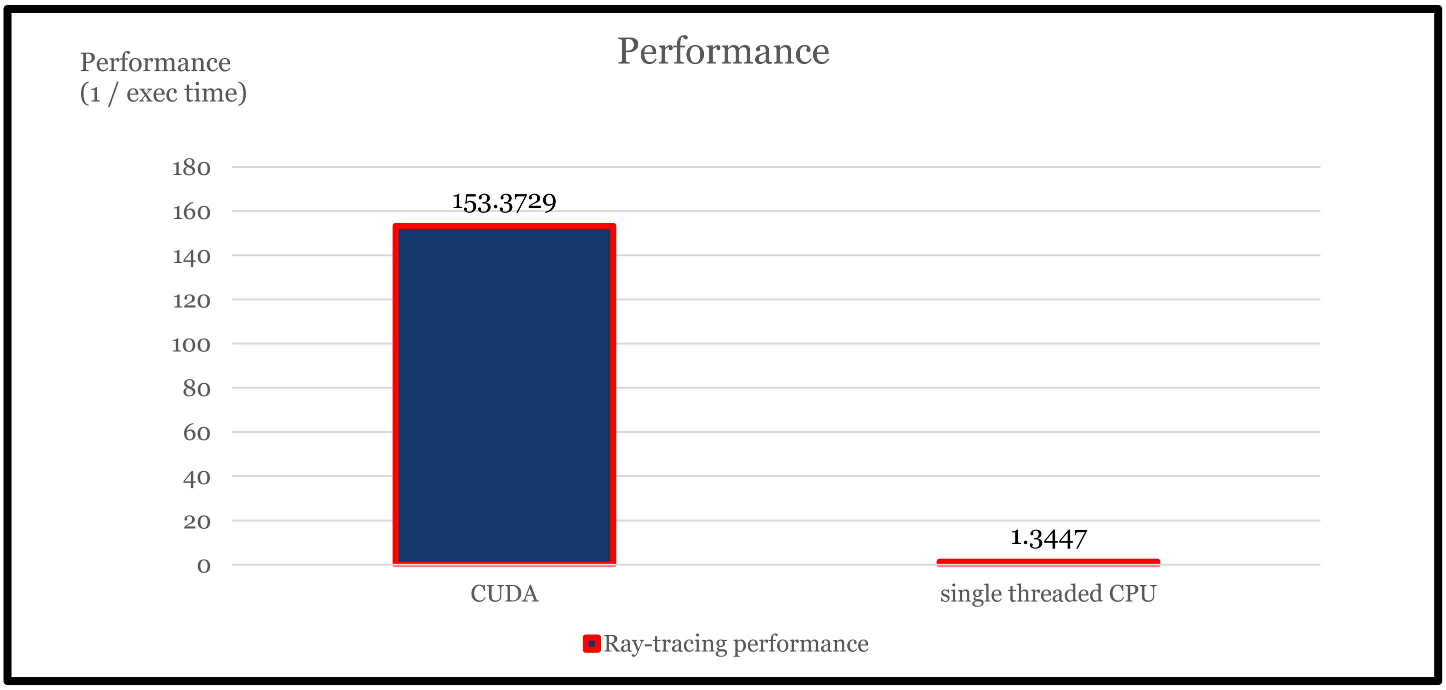
- **Performance**

OpenMp	1	2	4	6	8	10
Performance (1/exec time)	1.3205	2.6610	4.7046	5.7526	7.7925	8.5770

CUDA	
Performance (1/exec time)	153.3729

Single threaded CPU	
Performance (1/exec time)	1.3447





[Interpretation/Explanation on the Results]

The above results shows that the more number of threads, the better performance. When the number of threads is small (ex. number of threads = 1), it takes 0.7 seconds. But, when the number of threads is large, program performance is improved dramatically. In other words, using OpenMp with large number of threads has good performance. Also, using CUDA library has much better performance than OpenMp. Using CUDA library is 20 times faster than OpenMp. It’s because GPU has better performance in highly parallel applications than CPU.