2022.1 Multicore Computing, Project #4

Problem 1

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

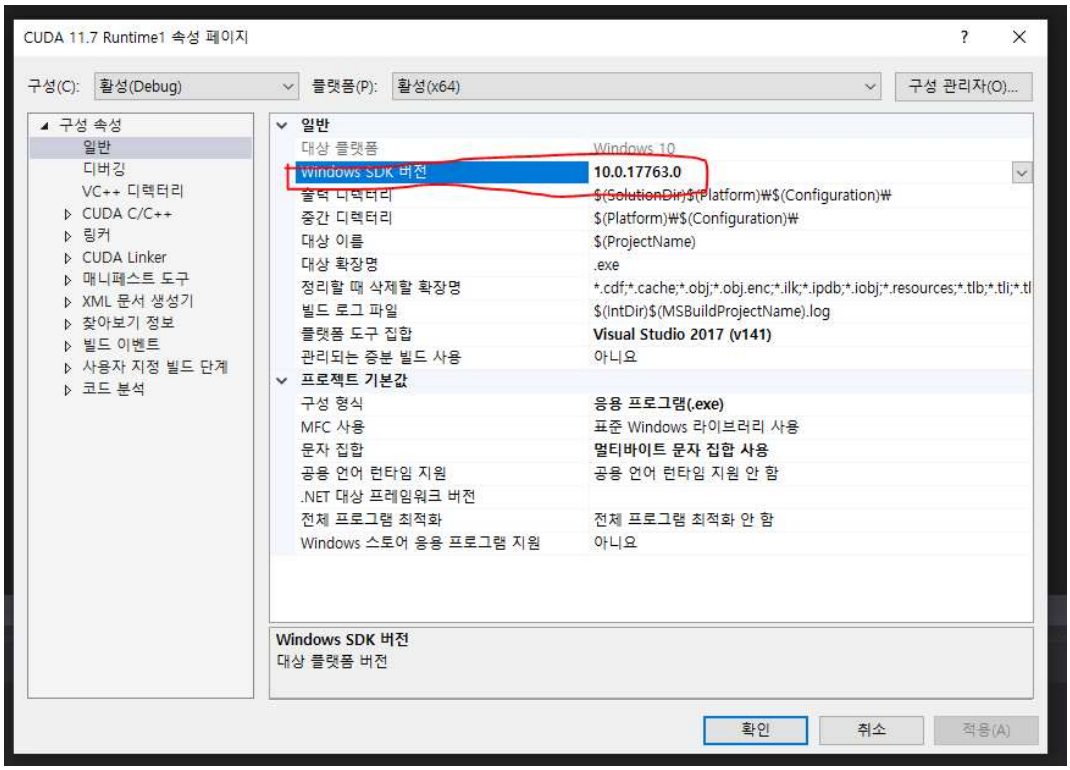
- openmp_ray.c

gcc -g openmp_ray.c -o openmp_ray -fopenmp

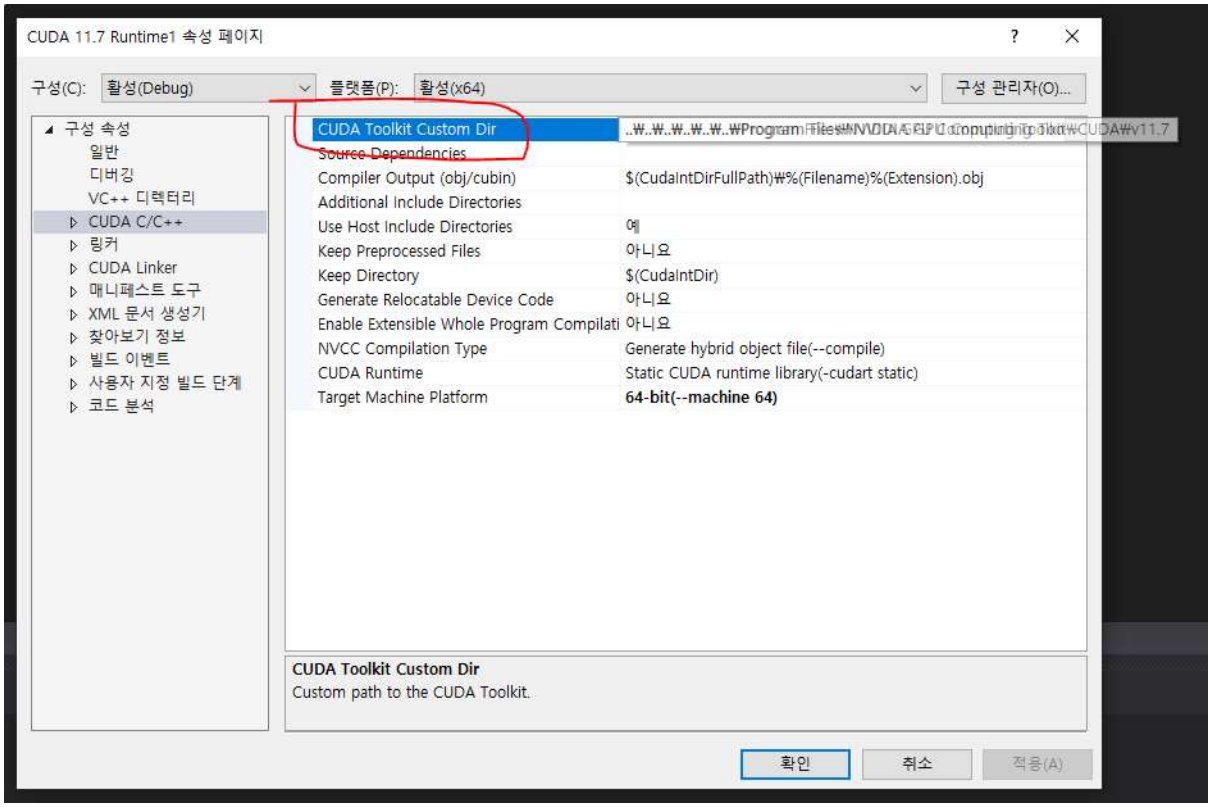
- Cuda_ray.cu

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 ~~~ cuda_ray.cu

(c) how to execute

- openmp_ray.c

Just type 'openmp_ray.exe 4' (4 is # of threads)

- cuda_ray.cu

Build visual studio target solution, and type

'cuda_ray.exe' (no parameter)

Entire source code

- openmp_ray.c

```
-  //
-  // Created by song on 2022-05-30.
-  //
-
-  #include <omp.h>
-  #include <stdio.h>
-  #include <stdlib.h>
-  //#include <sys/time.h>
-  //#include <windows.h>
-  #include <time.h>
-  #include <math.h>
-
-  #define CUDA 0
-  #define OPENMP 1
-  #define SPHERES 20
-
-  #define rnd( x ) (x * rand() / RAND_MAX)
-  #define INF 2e10f
-  #define DIM 2048
-  void ppm_writes();
-
-  void kernel();
-
-  struct Sphere {
-      //define Sphere ( there is no hit function in sphere)
-      float   r,b,g;
-      float   radius;
-      float   x,y,z;
-
-  };
-
-  void ppm_writes(unsigned char* bitmap, int xdim,int ydim, FILE* fp)
```

```
-   {
-       //ppm write function
-       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 isPrime(int);
-
-   int main (int argc, char** args){
-
-   //    unsigned long startTime = timeGetTime();
-       clock_t start, end;
-
-       start = clock(); //define start time
-
-
-       int num_threads;
-
-
-       omp_set_num_threads(atoi(args[1])); //parse num of thread parameter
-       num_threads = atoi(args[1]);
-
-       int x,y;
-       unsigned char* bitmap;
-
-
-       FILE* fp = fopen("result.ppm","w"); //write result.ppm file
-
-
-       struct Sphere *temp_s = (struct Sphere*)malloc( sizeof(struct Sphere)
    * SPHERES ); //define spheres array
```

```
-       for (int 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);
    //define bitmap
-
-
-   #pragma omp parallel for schedule(guided) collapse(2) // run kernel
    function with omp parallel (collapse(2)) nested loop
-       for (x = 0; x < DIM; x++){
-           for (y = 0; y < DIM; y++) {
-               kernel(x, y, temp_s, bitmap);
-           }
-       }
-
-       end = clock();
-
-       printf("OpenMP (%d threads) ray tracing: %lf
    sec\n",num_threads,(double)(end - start)/1000.0);
-
-       ppm_writes(bitmap, DIM, DIM, fp); //ppm write
-
-       fclose(fp);
-       free(bitmap);
-       free(temp_s);
-
-
-       printf("[result.ppm] was generated.");
-
-   }
-
-   float hit(struct Sphere s, float ox, float oy, float *n){
-       //hit function
-       float dx = ox - s.x;
```

```
-       float dy = oy - s.y;
-       if (dx*dx + dy*dy < s.radius*s.radius) {
-           float dz = sqrtf( s.radius*s.radius - dx*dx - dy*dy );
-           *n = dz / sqrtf( s.radius * s.radius );
-           return dz + s.z;
-       }
-       return -INF;
-   }
-
-   void kernel(int x, int y, struct Sphere* s, unsigned char* ptr)
-   {
-       //kernel function to point a pixel which has circle area
-       int offset = x + y*DIM;
-       float ox = (x - DIM/2);
-       float oy = (y - DIM/2);
-
-
-       float r=0, g=0, b=0;
-       float   maxz = -INF;
-
-
-       for(int i=0; i<SPHERES; i++) {
-           //check all spheres
-           float   n;
-           float   t = hit( s[i],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;
-   }
-
```

- Cuda_ray.cu

```
-
-   #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>
-
-   #define CUDA 0
-   #define OPENMP 1
-   #define SPHERES 20
-
-   #define rnd( x ) (x * rand() / RAND_MAX)
-   #define INF 2e10f
-   #define DIM 2048
-
-   #define GRID_SIZE 128
-   #define BLOCK_SIZE 16
-
-   struct Sphere {
-       //define Sphere with hit function
-       float   r, b, g;
-       float   radius;
-       float   x, y, z;
-       __device__ 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(unsigned char *c, Sphere* s)
-   {
-       int x = blockIdx.x*blockDim.x + threadIdx.x; //define x axis
-       int y = blockIdx.y*blockDim.y + threadIdx.y; //define y axis
-
-       int offset = x + y * DIM;
-       float ox = (x - DIM / 2);
-       float oy = (y - DIM / 2);
-
-       float r = 0, g = 0, b = 0;
-       float   maxz = -INF;
-       for (int i = 0; i < SPHERES; i++) { //find all Spheres to print a
    pixel
-           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;
-           }
-       }
-
-       c[offset * 4 + 0] = (int)(r * 255);
-       c[offset * 4 + 1] = (int)(g * 255);
-       c[offset * 4 + 2] = (int)(b * 255);
-       c[offset * 4 + 3] = 255;
-
-   }
-
-   void ppm_write(unsigned char* bitmap, int xdim, int ydim, FILE* fp)
-   {
-       //ppm write function to write result.ppm file
-       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");
-       }
-   }
-
-   cudaError_t cudaRun(); // cudaRun function to help run cuda function
-
-
-   int main()
-   {
-
-       cudaError_t cudaStatus = cudaRun(); // cuda Run
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaRun failed!");
-           return -1;
-       }
-
-       cudaStatus = cudaDeviceReset();
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaRun failed!");
-           return 1;
-       }
-
-       return 0;
-   }
-
-   cudaError_t cudaRun()
-   {
-       FILE *fp = fopen("result.ppm", "w"); //write empty ppm file
-
-       Sphere* dev_s = 0;
-
-       unsigned char *dev_bitmap;
-       unsigned char *bitmap;
-
```

```
-       bitmap = (unsigned char*)malloc(sizeof(unsigned char)*DIM*DIM * 4); //
    allocate memory to host
-
-       cudaError_t cudaStatus;
-
-       Sphere *temp_s = (Sphere *) malloc(sizeof(Sphere) * SPHERES); //define
    random sphere array
-
-       for (int 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;
-       }
-
-       cudaStatus = cudaSetDevice(0); //set cuda device
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "CudaSetDevice failed! Do you have a CUDA-capable
    GPU installed?");
-           goto Error;
-       }
-
-       clock_t start, end;
-
-
-       start = clock();
-
-       cudaStatus = cudaMalloc((void**)&dev_s, SPHERES * sizeof(Sphere)); //
    memory allocate for gpu(device) with spheres
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaMalloc failed!");
-           goto Error;
-       }
-
-       cudaStatus = cudaMalloc((void**)&dev_bitmap, sizeof(unsigned char) *
    DIM * DIM * 4); // memory allocate for gpu(device) with bitmap
-       if (cudaStatus != cudaSuccess)
-       {
```

```
-           fprintf(stderr, "cudaMalloc failed!");
-           goto Error;
-       }
-
-       cudaStatus = cudaMemcpy(dev_s, temp_s, SPHERES * sizeof(Sphere),
   cudaMemcpyHostToDevice); // copy variable from host to device (spheres)
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaMemcpy failed!");
-           goto Error;
-       }
-       cudaStatus = cudaMemcpy(dev_bitmap, bitmap, sizeof(unsigned char) *
   DIM * DIM * 4, cudaMemcpyHostToDevice);  // copy variable from host to
   device (bitmap) maybe empty bitmap
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaMemcpy failed!");
-           goto Error;
-       }
-
-       dim3 dimGrid(GRID_SIZE, GRID_SIZE, 1); // define grid
-       dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE, 1); //define block
-
-       kernel<<<dimGrid, dimBlock>>>(dev_bitmap, dev_s); // run kernel
   function
-
-       cudaStatus = cudaGetLastError();
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaRun launch failed : %s\n",
   cudaGetErrorString(cudaStatus));
-           goto Error;
-       }
-
-       cudaStatus = cudaDeviceSynchronize();
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaDeviceSynchronize returned error code %d
   after launching addKernel!\n", cudaStatus);
-           goto Error;
-       }
-
```

```
-       cudaStatus = cudaMemcpy(bitmap, dev_bitmap,DIM*DIM * sizeof(unsigned
    char)*4, cudaMemcpyDeviceToHost); // copy result device to host
-       if (cudaStatus != cudaSuccess)
-       {
-           fprintf(stderr, "cudaMemcpy failed!");
-           goto Error;
-       }
-
-       end = clock(); // measure cuda program run time
-
-       printf("CUDA ray tracing: %lf sec\n", (double)(end - start) / 1000.0);
-
-       ppm_write(bitmap,DIM,DIM,fp); // ppm write
-       printf("[result.ppm] was generated.\n");
-
-       fclose(fp);
-

-   Error:
-       //free memory allocate to gpu memory
-       cudaFree(dev_s);
-       cudaFree(dev_bitmap);
-
-       return cudaStatus;
-   }
-
```

Program output result

- openmp_ray.c

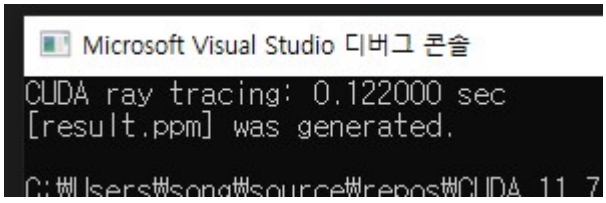
[result.ppm] by openmp_ray

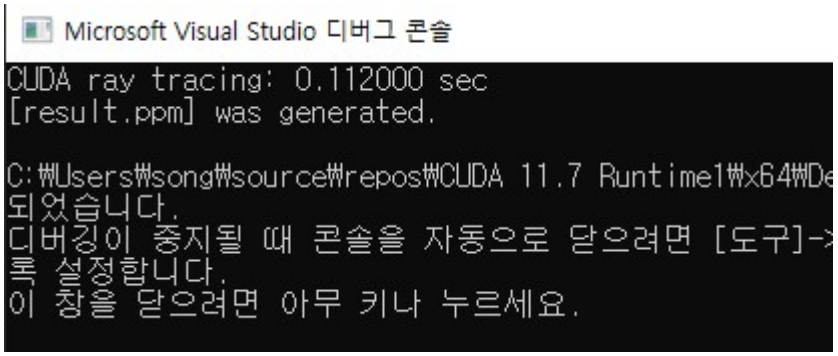
```
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 1
OpenMP (1 threads) ray tracing: 1.100000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 2
OpenMP (2 threads) ray tracing: 0.560000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 4
OpenMP (4 threads) ray tracing: 0.284000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 8
OpenMP (8 threads) ray tracing: 0.177000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 6
OpenMP (6 threads) ray tracing: 0.203000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 10
OpenMP (10 threads) ray tracing: 0.142000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 12
OpenMP (12 threads) ray tracing: 0.129000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 14
OpenMP (14 threads) ray tracing: 0.137000 sec
[result.ppm] was generated.
PS C:\Users\song\CLionProjects\openmp_ray> ./openmp_ray.exe 16
OpenMP (16 threads) ray tracing: 0.133000 sec
[result.ppm] was generated.
```

- cuda_ray.cu

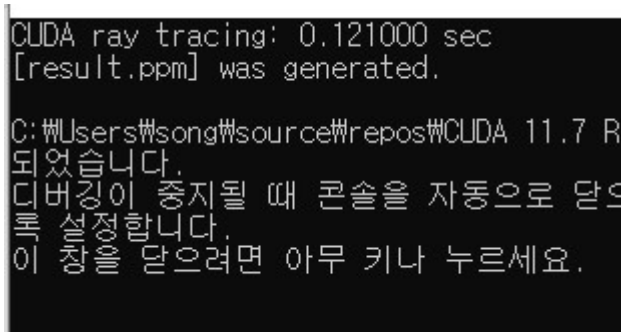
- 

[result.ppm] by cuda_ray

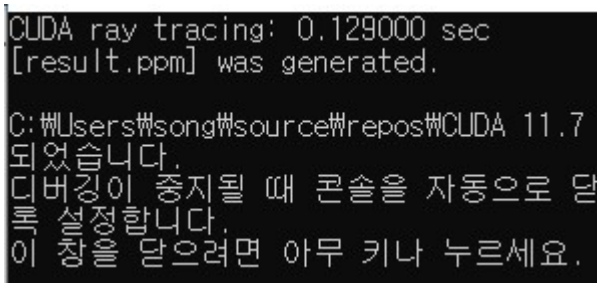




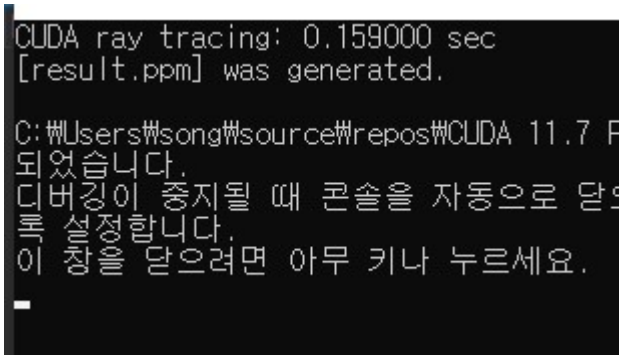
Block size = (32,32)



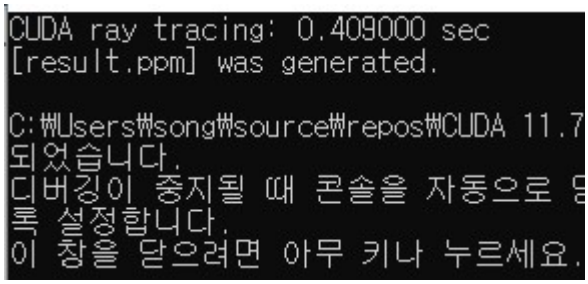
Block size = (16,16)



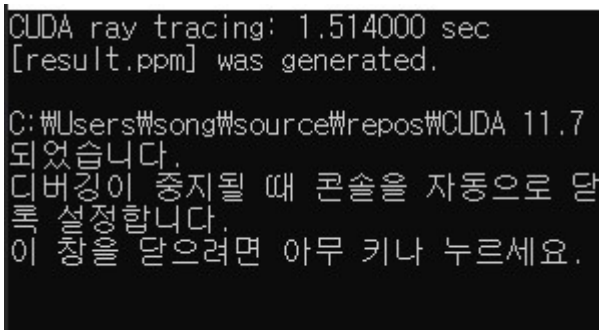
Block size = (8,8)



Block size = (4,4)



Block size = (2,2)



Block size = (1,1)

Program execution

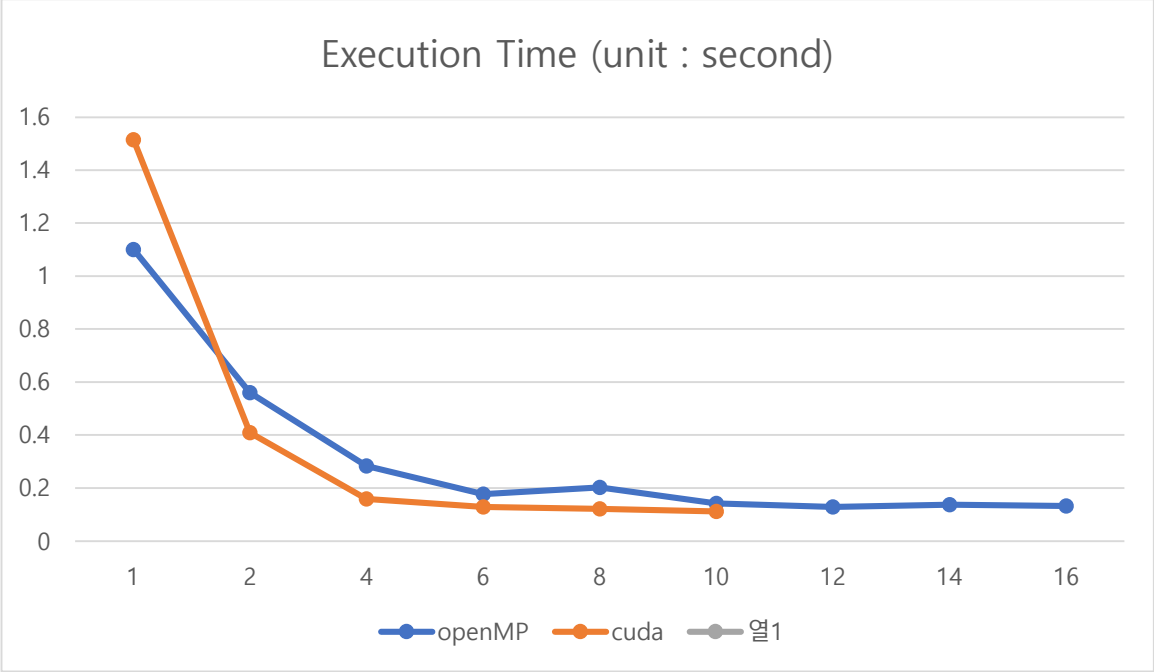
Experimental results

Openmp_ray

Exec time(unit: sec)	1	2	4	6	8	10	12	14	16
	1.1	0.56	0.284	0.177	0.203	0.142	0.129	0.137	0.133

Cuda_ray

Block size(unit: sec)	1	2	4	8	16	32
	1.514	0.409	0.159	0.129	0.121	0.112



Interpretation/explanation

Openmp is slowest when there is one thread, and performance increases rapidly until the number of threads increases to six, with little increase in performance since the number of threads exceeds eight.

This is because it is about that time when the overhead of switching is greater as the context increases rather than the performance increases due to the increase in threads.

The cuda can use the concept of grid and block to parallelize arrays of sizes 2048x2048. The grid is a little bit bigger, and the block is how you're going to split the threads in the grid.

The larger the block size, the more likely it is to be parallel. Therefore, when the block size was (1,1), it was slower than openmp, and while the block size increased to (2,2), (4,4), (8,8), (16, 16), (32, 32), the performance increased very quickly.

Because cuda has a block size limit (approximately 1024), it has not been used since 64x64=4096.