Multicore Computing Project4

- Problem 1 -

과목명	멀티코어컴퓨팅
제출일	2023.06.12.
학 번	20183901
학 과	소프트웨어학부
이 름	김상민

목 차

1. Program setup

(a). Execution environment

Execution environment: Google Colab

GPU: Python 3 Google Compute Engine 백엔드

(b). How to compile

CUDA: nvcc cuda_ray.cu

OpenMP: g++ openmp_ray.cpp

(c). How to execute

CUDA: a.out 0 result.ppm

OpenMP: a.out <threads from 1 to 16> result.ppm

2. Source code

(a). cuda_ray.cu

```
#include <assert.h>
#include <cuda.h>
#include <cuda_runtime.h>

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

#define CUDA 0
#define OPENMP 1
#define SPHERES 20 // 구체 개수

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

// 구체 정의
struct Sphere {
    float   r,b,g;
    float   radius;
    float   x,y,z;

    // ray hit
    __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;
    }
};

// 병렬화 부분
__global__ void kernel(Sphere* s, unsigned char* ptr)
{
  //픽셀 인덱스
  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);

  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");
  }
}

//메인코드
int main(int argc, char* argv[])
{
  int no_threads;
  int option;
  int x,y;
  unsigned char* bitmap;

  srand(time(NULL));
```

```
  if (argc!=3) {
    printf("> a.out [option] [filename.ppm]\n");
    printf("[option] 0: CUDA, 1~16: OpenMP using 1~16 threads\n");
    printf("for example, '> a.out 8 result.ppm' means executing OpenMP with 8 threads\n");
    exit(0);
  }
  FILE* fp = fopen(argv[2],"w");

  if (strcmp(argv[1],"0")==0) option=CUDA;
  else {
    option=OPENMP;
    no_threads=atoi(argv[1]);
  }

  //구체 생성
  Sphere *temp_s = (Sphere*)malloc( sizeof(Sphere) * SPHERES );
  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);

  //CUDA로 복사할 변수
  Sphere *dev_temp_s;
  unsigned char *dev_bitmap;

  //CUDA 메모리 할당
  cudaMalloc((void**)&dev_temp_s, sizeof(Sphere) * SPHERES);
  cudaMalloc((void**)&dev_bitmap, sizeof(unsigned char) * DIM * DIM * 4);

  //sphere: host -> device 복사
  cudaMemcpy(dev_temp_s, temp_s, sizeof(Sphere) * SPHERES, cudaMemcpyHostToDevice);


  int thread_num = 16;
  dim3 thread_per_block(thread_num, thread_num); //블록 당 쓰레드 수: 16x16
  dim3 block_num(DIM / thread_per_block.x, DIM / thread_per_block.y); //블록 수: 2048/16 x 2048/

  clock_t start_time = clock(); // 시간 측정 시작
  kernel<<<block_num, thread_per_block>>>(dev_temp_s, dev_bitmap); //ray tracing 계산
```

```
  clock_t end_time = clock(); //시간 측정 끝

  clock_t diff_time = end_time - start_time;
  printf("CUDA ray tracing: %f sec. \n", (double)diff_time/CLOCKS_PER_SEC); // 시간 출력

  //tracing 결과: device -> host
  cudaMemcpy(bitmap, dev_bitmap, sizeof(unsigned char) * DIM * DIM * 4, cudaMemcpyDeviceToHost

  ppm_write(bitmap,DIM,DIM,fp);

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

  cudaFree(dev_temp_s);
  cudaFree(dev_bitmap);

  return 0;
}
```

(b). openmp_ray.cpp

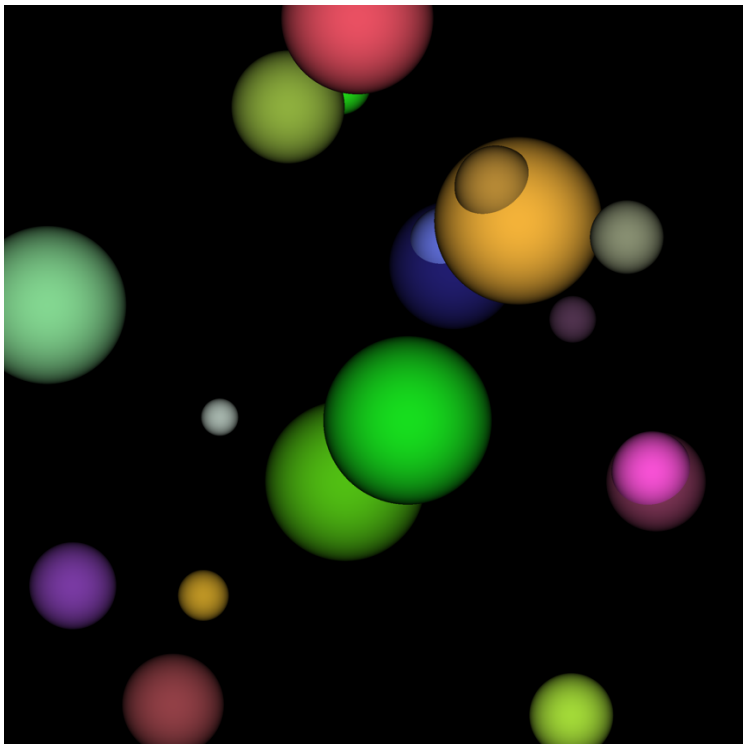
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define CUDA 0
#define OPENMP 1
#define SPHERES 20

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

#include <omp.h> //OpenMP 라이브러리

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 no_threads;
  int option;
  int x,y;
  unsigned char* bitmap;

  srand(time(NULL));

  if (argc!=3) {
    printf("> a.out [option] [filename.ppm]\n");
    printf("[option] 0: CUDA, 1~16: OpenMP using 1~16 threads\n");
    printf("for example, '> a.out 8 result.ppm' means executing OpenMP with 8 threads\n");
    exit(0);
  }
  FILE* fp = fopen(argv[2],"w");

  if (strcmp(argv[1],"0")==0) option=CUDA;
  else {
```

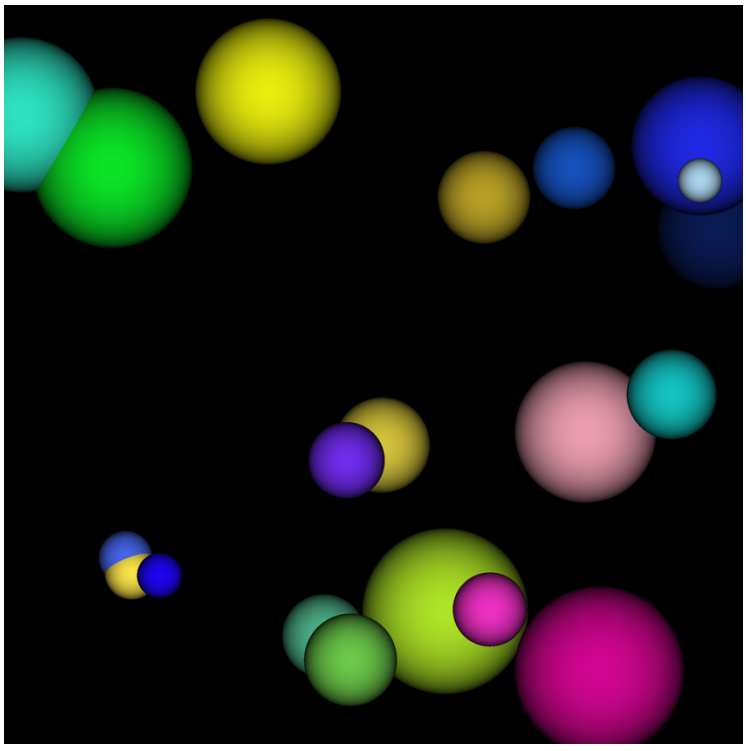
```
    option=OPENMP;
    no_threads=atoi(argv[1]);
  }

  Sphere *temp_s = (Sphere*)malloc( sizeof(Sphere) * SPHERES );
  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);

  clock_t start_time = clock(); //시간 측정

#pragma omp parallel for schedule(guided) num_threads(no_threads) //Guided 스케줄링 병렬화
  for (x=0;x<DIM;x++)
    for (y=0;y<DIM;y++) kernel(x,y,temp_s,bitmap); //ray tracing

  clock_t end_time = clock(); //시간 출력
  ppm_write(bitmap,DIM,DIM,fp);

  clock_t diff_time = end_time - start_time;
  printf("openmp ray tracing: %f sec. \n", (double)diff_time/CLOCKS_PER_SEC); //시간 결과

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

  return 0;
}
```

3. Program output results

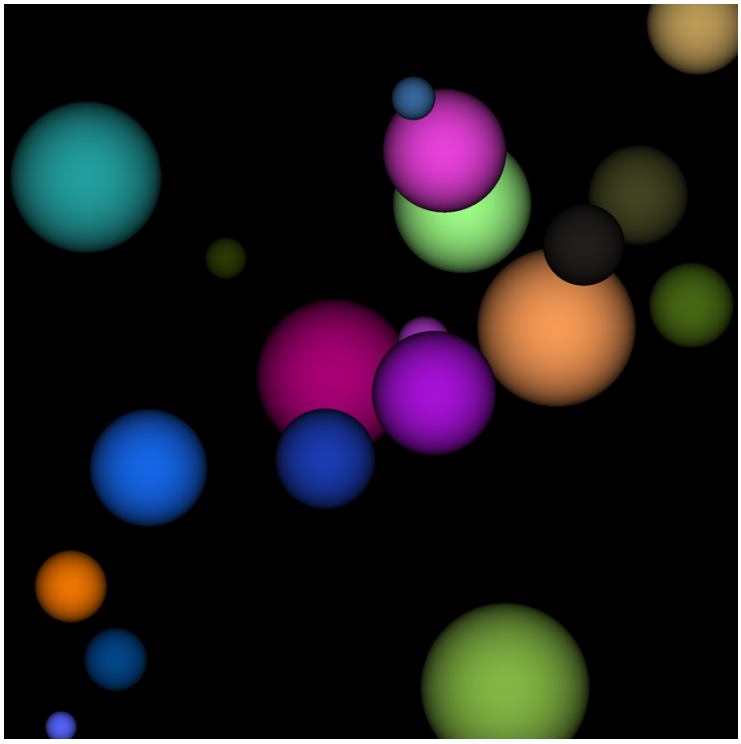
Original



(a). CUDA



(b). OpenMP



4. Experiment results

Original

```
!./a.out 0 result.ppm
origin ray tracing: 1.434041 sec.
```

(a). CUDA

- threads_per_block: 1x1

```
!./a.out 0 result.ppm
CUDA ray tracing: 0.000039 sec.
```

- threads_per_block: 4x4

```
!./a.out 0 result.ppm
CUDA ray tracing: 0.000027 sec.
```

- threads_per_block: 8x8

```
!./a.out 0 result.ppm
CUDA ray tracing: 0.000018 sec.
```

- threads_per_block: 16x16

```
!./a.out 0 result.ppm
CUDA ray tracing: 0.000017 sec.
```

(b). OpenMP

- number of threads: 1

```
!./a.out 16 result.ppm
openmp ray tracing: 0.831140 sec.
```

- number of threads: 4

```
!./a.out 8 result.ppm
openmp ray tracing: 0.821818 sec.
```

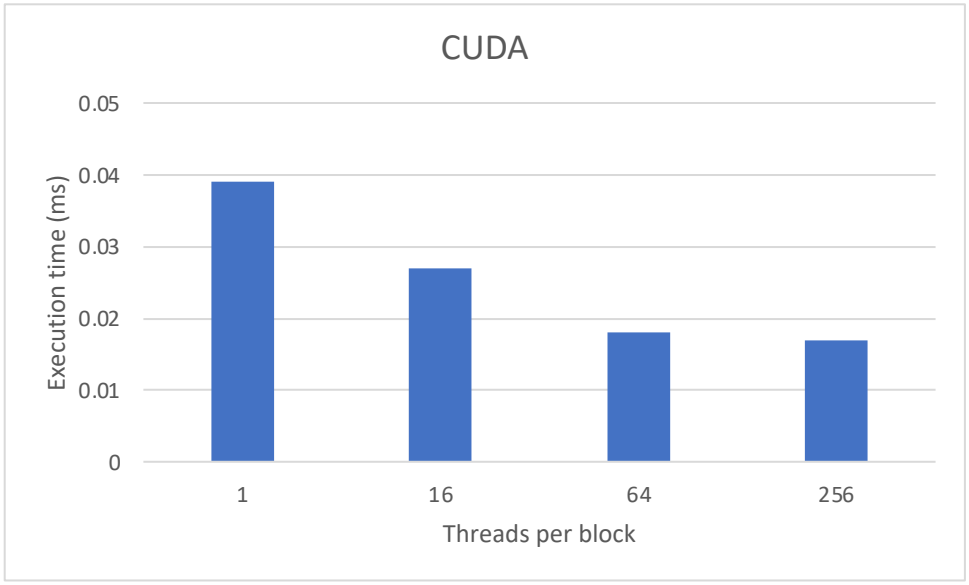
- number of threads: 8

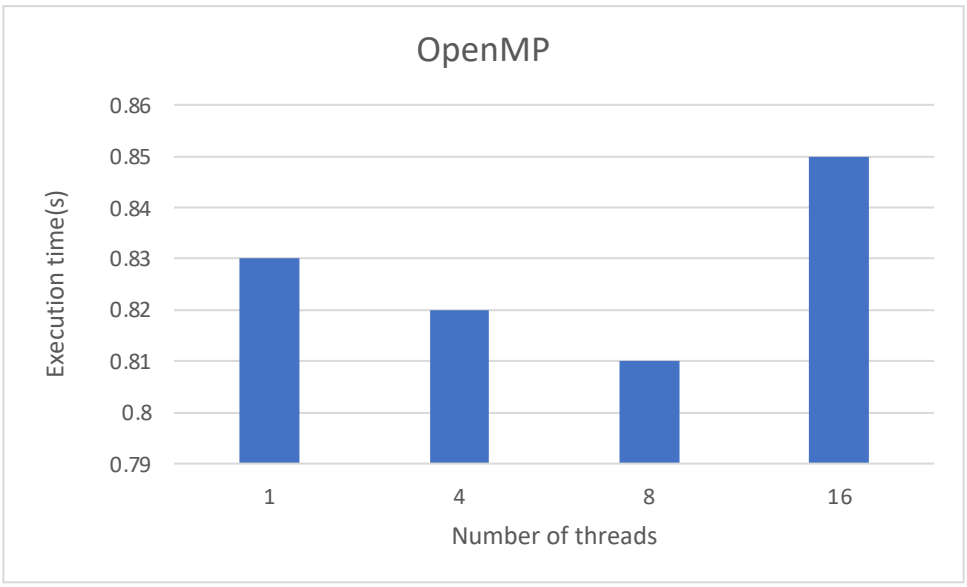
```
!./a.out 4 result.ppm
openmp ray tracing: 0.811877 sec.
```

- number of threads: 16

```
!./a.out 1 result.ppm
openmp ray tracing: 0.854301 sec.
```

(c). Explanation





Both CUDA and OpenMP implemented version of ray tracing showed much better result than the original CPU way, especially the CUDA version. In CUDA version, I assigned (1,1), (4,4), (8,8), (16,16) threads per block. As the number of threads per block increases, execution time reduced and plateaued at some point. Also in OpenMP version, execution time reduced as the number of threads increased, but when 16 threads were used, execution time increased due to overheads.