

# Problem1

---

## Environment

Os	Pop!_OS 22.04 LTS x86
CPU	Intel i7-8665U (8) @ 1.900GHz
Memory	16Gb
GCC version	14.0.0
GNU Make version	4.3
GPU	Google Colab

## Build

openmp\_ray.cpp

```
make
```

or

```
g++ -o openmp_ray ./openmp_ray.cpp -fopenmp
```

cuda\_ray.cu

```
nvcc ./test_cuda.cu -o cuda_ray
```

## Run

openmp\_ray

```
openmp_ray < number of threads >
```

cuda\_ray

```
cuda_ray
```

## Source code

### openmp\_ray.cpp

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <omp.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 no_threads;
    int x, y;
    unsigned char *bitmap;
    double start_time = 0;
    double end_time = 0;
    const char *output_file = "result.ppm";

    srand(time(NULL));

    if (argc != 2)
    {
        printf("> openmp_ray [option]\n");
        printf("[option] 1~16: OpenMP using 1~16 threads\n");
        printf("for example, '> openmp_ray 8' means executing OpenMP with 8
threads\n");
        return EXIT_FAILURE;
    }
    FILE *fp = fopen(output_file, "w");

    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);
start_time = omp_get_wtime();
#pragma omp parallel for schedule(dynamic) num_threads(no_threads)
for (y = 0; y < DIM; y++)
{
    for (x = 0; x < DIM; x++)
    {
        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 :%.3lf sec\n", no_threads,
(end_time - start_time));
printf("[%s] was generated\n", output_file);
return EXIT_SUCCESS;
}

```

## cuda\_ray.cu

```

#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;
    __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(Sphere *s, unsigned char *ptr)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    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++)
    {
        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[])
{
    unsigned char *bitmap;
    const char *filename = "result.ppm";
    FILE *fp = fopen(filename, "w");
    Sphere *temp_s = (Sphere *)malloc(sizeof(Sphere) * SPHERES);
    Sphere *d_s;
    unsigned char *device_bitmap;
    cudaEvent_t start;
    cudaEvent_t end;
    float elapsedTime;

    cudaEventCreate(&start);
    cudaEventCreate(&end);
    srand(time(NULL));
    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;
    }
    cudaMalloc((void **)&d_s, sizeof(Sphere) * SPHERES);
    cudaMalloc((void **)&device_bitmap, sizeof(unsigned char) * DIM * DIM *
4);
    cudaMemcpy(d_s, temp_s, sizeof(Sphere) * SPHERES,
cudaMemcpyHostToDevice);
    cudaEventRecord(start);
    kernel<<<dim3((DIM + 15) / 16, (DIM + 15) / 16), dim3(16, 16)>>>(d_s,
device_bitmap);
    cudaEventRecord(end);
    cudaEventSynchronize(end);
    cudaEventElapsedTime(&elapsedTime, start, end);
    bitmap = (unsigned char *)malloc(sizeof(unsigned char) * DIM * DIM *
4);
    cudaMemcpy(bitmap, device_bitmap, sizeof(unsigned char) * DIM * DIM *
4, cudaMemcpyDeviceToHost);
    printf("CUDA ray tracing: %.3lf sec\n", elapsedTime);
    ppm_write(bitmap, DIM, DIM, fp);
}

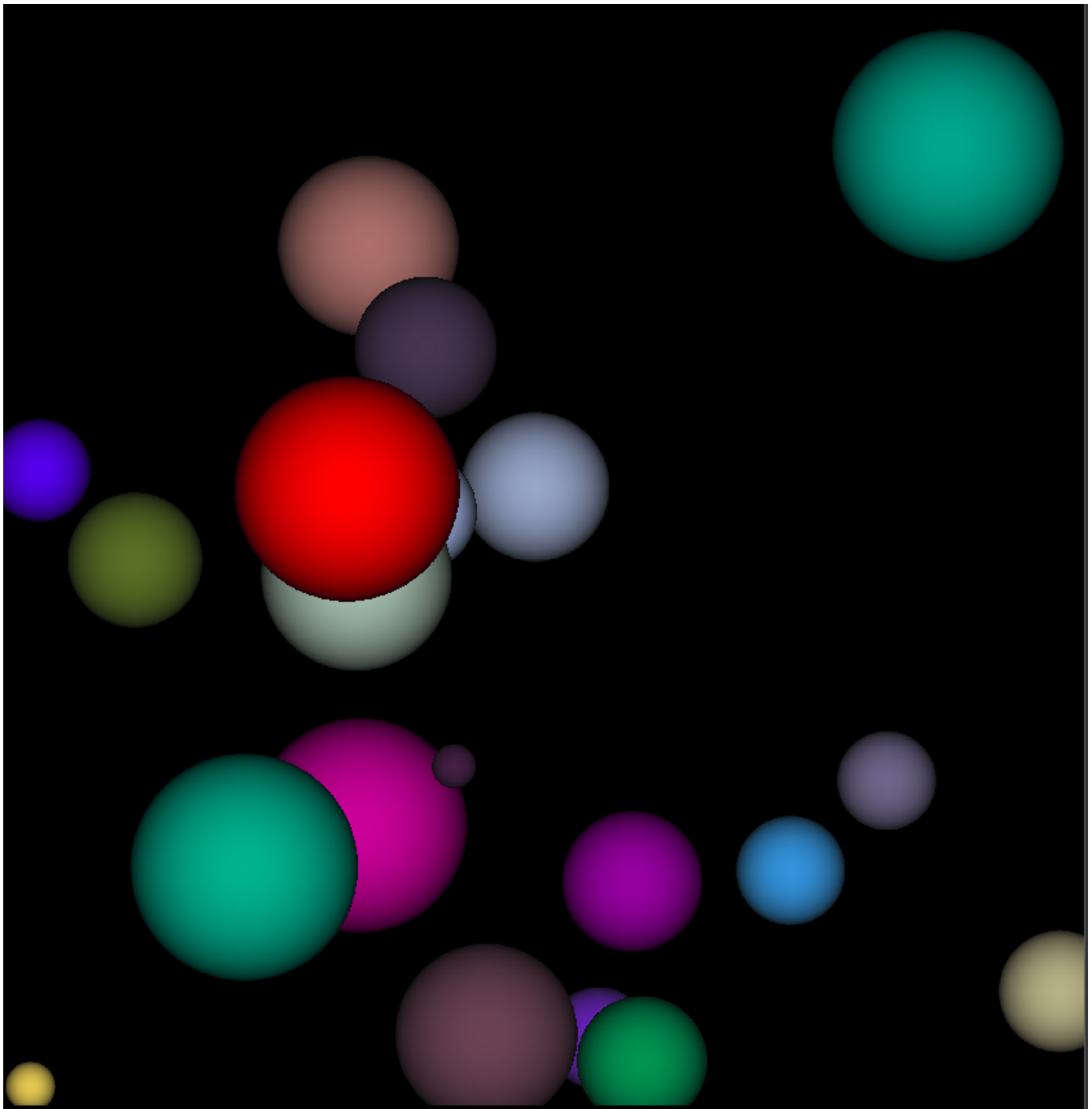
```

```
printf("[%s] was generated.", filename);  
fclose(fp);  
free(bitmap);  
free(temp_s);  
cudaFree(d_s);  
cudaEventDestroy(start);  
cudaEventDestroy(end);  
cudaFree(device_bitmap);  
return 0;  
}
```

## Outputs

### Open MP

```
~/C/M/CAU_Multicore_assignments > *... proj4/problem1 > ./openmp_ray 8  
OpenMP (8 threads) ray tracing :0.474 sec  
[result.ppm] was generated
```

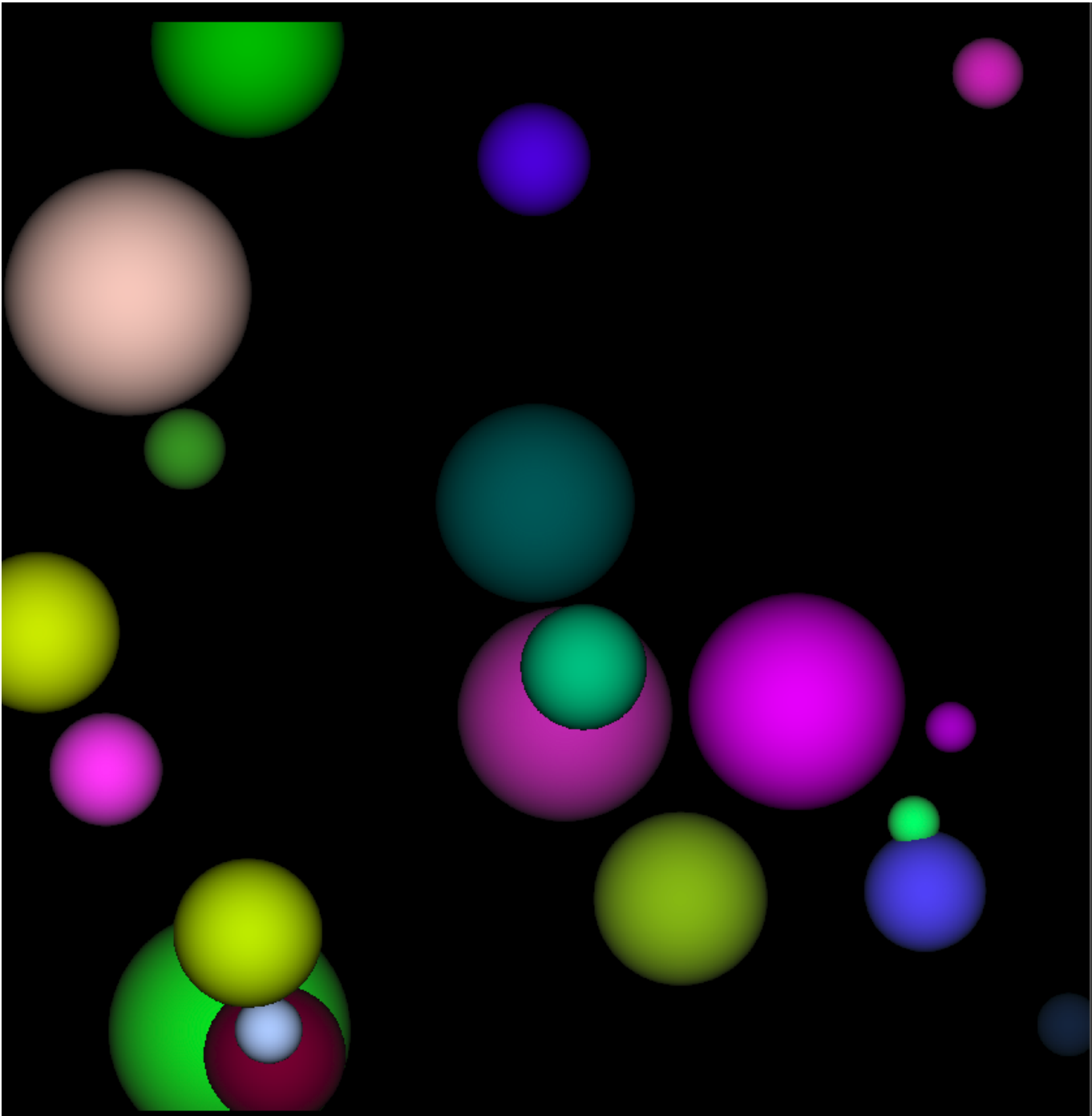


Cuda

```
[3] !./cuda_ray
```

```
CUDA ray tracing: 0.001 sec  
[result.ppm] was generated.
```





# Results

Raw

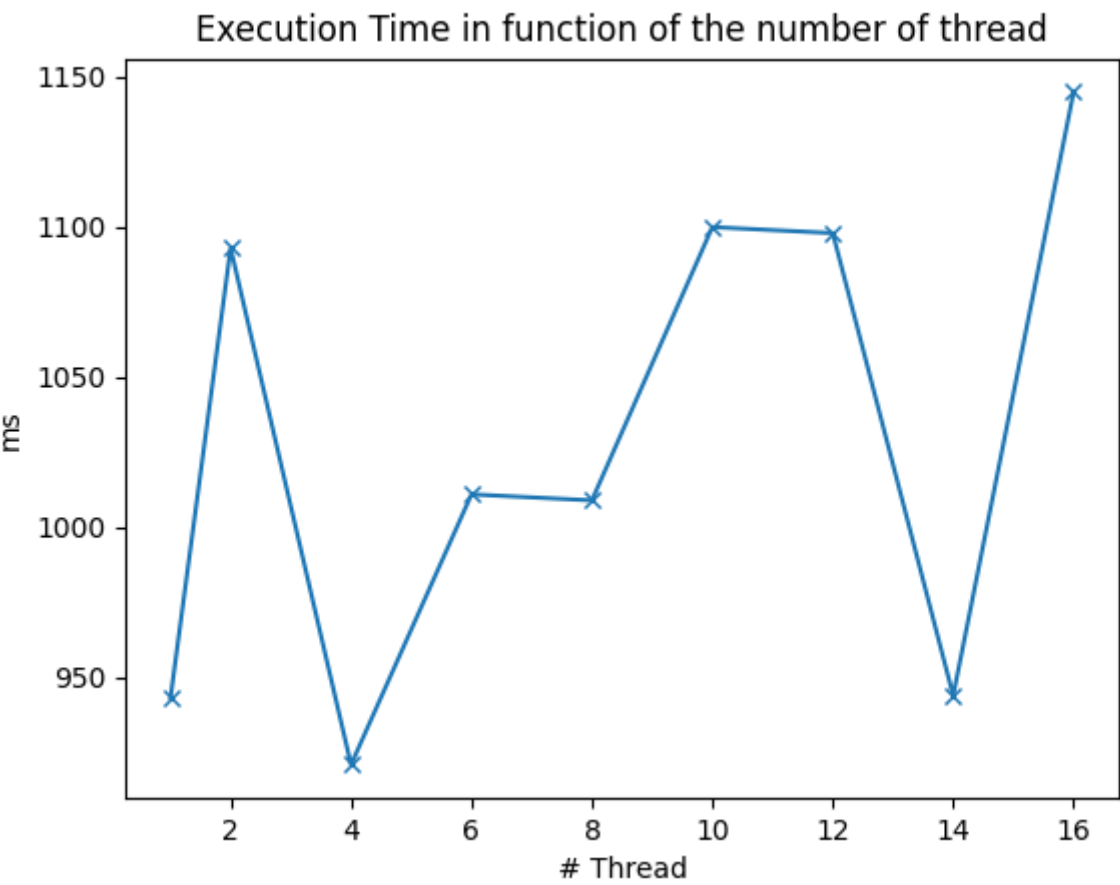
OpenMP

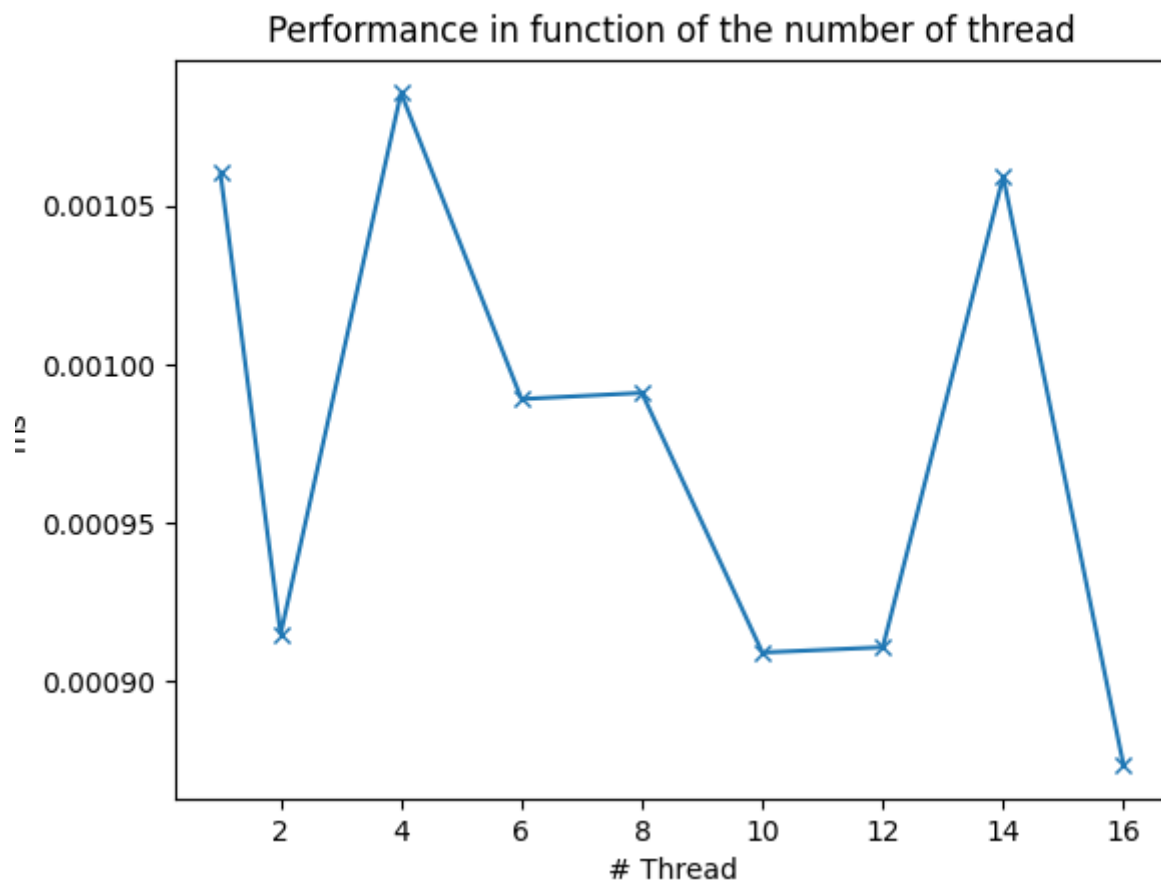
Number of thread	Execution time	Performance
1	943	0.0010604453870625664
2	1093	0.0009149130832570906
4	921	0.0010857763300760044
6	1011	0.0009891196834817012

Number of thread	Execution time	Performance
8	1009	0.0009910802775024777
10	1100	0.0009090909090909091
12	1098	0.0009107468123861566
14	944	0.001059322033898305
16	1145	0.0008733624454148472

Graphs

OpenMP





## Interpretation

We can see that for 4 threads and 16 threads it take the less time.