2024.1 [Multicore Computing] Project 4: OpenMP/CUDA/Thrust Practice
(Deadline : 11:59pm, June 14)

<u>Summision method : eClass</u>
- create a directory 'proj4' and create subdirectories 'problem1' and 'problem2' in directory 'proj4'
- In 'problem1', insert
    (i) source files (openmp_ray.cpp,cuda_ray.cu), (ii) report pdf file, and (iii) demo video file (.mp4)
- In 'problem2', insert
    (i) source file (thrust_ex.cu), (ii) report pdf file, and (iii) demo video file (.mp4)
- compress the directory 'proj4' into proj4.zip and submit it to eclass.
- <u>Important Notice</u>: there are two ways to edit/compile/execute CUDA/THRUST codes.
    (i) use a PC equipped with NVIDIA graphics card
    (ii) use Google Colab (see our class webpage for details.)
    You may choose one of above methods and do this project. The source files you submit should be compilable so that I can test execution of your code.

Problem 1. Implementing two versions of Ray-Tracing (openmp_ray.c, and cuda_ray.cu) that utilizes OpenMP and CUDA library, respectively.

(i) Write OpenMP and CUDA programs using C or C++ language.
- <u>Look at the code [raytracing.cpp] for ray-tracing of random spheres, which is available on our class webpage, and modify the code for your purpose.</u>
- You may assume the simplest form of Ray tracing that renders a scene with only spheres.
- Program input :
    (i) openmp_ray.cpp: [number of threads]
    (ii) cuda_ray.cu: no input
- Program output :
    (i) print ray-tracing processing time of your program using OpenMP or CUDA.
    (ii) generate image file (filename: result.ppm, format: .ppm or .bmp, image size: 2048X2048) that shows rendering result. In case you use google colab, you may have to copy (transfer) the resulting image file to your PC or your Google Drive.

Execution example of openmp_ray.cpp:
```
> openmp_ray.exe 8
OpenMP (8 threads) ray tracing: 0.418 sec
[result.ppm] was generated.
```

Execution example of cuda_ray.cu:
```
> cuda_ray.exe
CUDA ray tracing: 0.152 sec
[result.ppm] was generated.
```

(ii) write a report (pdf) that includes
- (a) execution environment (OS/CPU/GPU type or Colab?) (b) how to compile, (c) how to execute
- entire source code (openmp_ray.cpp and cuda_ray.cu) with appropriate comments
- program output results (i.e. screen capture) and ray-tracing result pictures
- experimental results : measuring the performance (execution time) of your OpenMP/CUDA implementation. Show the performance results with respect to various number of threads including tables, graphs and your interpretation/explanation. Also, include screen captures of output results.

(ii) create a demo video file (.mp4 format)
- create a demo video file (.mp4 format) that shows compilation and execution of your source files (openmp_ray.cpp and cuda_ray.cu). The size of the demo video file should be less than 30MB. Please include short audio explanation of what is going on during execution in the demo video.

Problem 2. Use thrust that is an open-source library

- Thrust is an open-source C++ template library for developing high-performance parallel applications, which is based on CUDA technology and brings a familiar abstraction layer to GPU.
- See our class webpage for more information on thrust library.

(i) Understand thrust library and write your C/C++ source code "thrust_ex.cu" that does following using the CUDA thrust library.

Approximate computation of the integral $\int_0^1 \frac{4.0}{(1+x^2)}dx$ by computing a sum of rectangles $\sum_{i=0}^{N} F(x_i)\Delta x$ where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i and $F(x) = \frac{4.0}{(1+x^2)}$. Choose N=1,000,000,000 that is a sufficiently large number for experimentation.

(ii) Write a report that includes (i) your source code, (ii) execution time table of sequential program using only one CPU thread [omp_pi_one.c (using N=1,000,000,000 and one thread) in class webpage] and your program using thrust library that utilizes GPU, and (iii) your explanation/ interpretation on the results. You also need to include screen capture image of execution results.

(iii) create a demo video file (.mp4 format)
  - create a demo video file (.mp4 format) that shows compilation and execution of your source file (thrust_ex.cu). The size of the demo video file should be less than 30MB. Please include short audio explanation of what is going on during execution in the demo video.