2021.1 [Multicore Computing] Project 4: OpenMP/CUDA/Thrust Practice

(Deadline : 11:59pm, June 13th)

Summision method : eClass
- create a directory 'proj4' and create subdirectory 'problem1' and 'problem2' in the directory 'proj4'
- In 'problem1', insert (i) source code files (openmp_ray.c, cuda_ray.cu), and (ii) report pdf file
- In 'problem2', insert (i) source code file (thrust_ex.cu) and executable file, and (ii) report pdf file
- compress the directory 'proj4' into proj4.zip and submit it to eclass.
- Important Notice: 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 the 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.
- 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.
- You may assume the simplest form of Ray tracing that renders a scene with only spheres.
- Program input :
  (i) openmp_ray.c: [number of threads]
  (ii) cuda_ray.cu: no input argument
- 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.c) > 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
- (i) execution environment (OS/CPU/GPU type or Colab?) (ii) how to compile, (iii) how to execute
- entire source code (openmp_ray.c 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 and the given single threaded CPU implementation. Show the performance results with respect to various number of threads. Include screen captures of output results.

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 CUDA/C++ source code "thrust_ex.cu" that does following.
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 CPU (omp_pi_serial.c in class webpage) and your program using thrust, and (iii) your explanation/ interpretation on the results.