Faculté des Sciences appliquées

INFO9012-A-a : Parallel Programming

From a sequential to a concurrent tiny ray tracer

Professeur :
Pascal FONTAINE

Groupe :
Romain LAMBERMONT
Arthur LOUIS

21 mai 2022

1 Hardware

We ran the tests on an 2022 ASUS Zenbook running a 4 cores Intel i7 11th generation CPU, which was already a pretty good starting base. Indeed by running the ray tracer sequentially, we acheived 8 frames per second.

2 Phase 1

For the first phase we simply used OpenMP by adding a simple line over the actual rendering loop to allow OpenMP to compute the frames of the ray tracer in parallel. This simple line allowed us to go from 8 to 18 frames per second. This line was :

```
#pragma omp parallel for
```

3 Phase 2

For the second phase, we moved from using OpenMP, to actually use the threads of the CPU with the `C++ <thread>` library. We used 1 thread to receive the inputs from the keyboard and to displays the frames while `N-1` threads computed the frames in parallel with `N` being the number of available threads in the system.

The computing threads are stocked in a vector and ran in a `while(boolWindow)` loop, which was always true as long as the window was open. When the window is closed, the threads are disabled using the `join` method. To ensure that there were no data race, we used the `mutex` library to lock and unlock the read and write of data correctly. To ensure that the images arrived in the good order to the displaying thread, we used a priority queue with a personalized structure adding an field `order` in the `sf::Image` object.

With the threads, we acheived a 28 frames per second, which is a pretty good improvement from the first phase.

4 Phase 3

For this phase, we reused our threads from the previous phase and simply modified the computation of the position and size of the spheres in the main function while also creating 2 methods to update these values in the tinyraytracer object to display them with the changing position and size. To ensure the period of oscillation of the spheres, we used the values $\frac{2}{3} * \text{fps}$ and $\frac{2}{5} * \text{fps}$ for respectively the red rubber ball and the mirror ball. With the mirror sphere changing in size (hardest thing to compute by the ray tracer), the ball was sometimes big or small, improving the average frames per second and leading us to aroung 37 frames per second in this phase.

5 Conclusion

We gathered our results in this small table :

Phase	Initial	OpenMP	Threads	Threads + Oscillating spheres
FPS	8	18	28	37

Table 1 – Frames per second relative to the phase