Parallelisation of a Raytracing algorithm

Alberto Taiuti
1300250
AG0803 – Architectures and Performance

Purpose of the application

- Render a 3D scene using a raytracing algorithm
- Make use of GPGPU to improve the performance and reduce the rendering time
- Explore the world of GPGPU computing

Application Design

- A base class represents a raytracer
- Child classes are instantiated to represent different categories of raytracer
  - CPU
  - GPU
- The main contains a pointer to the base class and by polymorphism instantiates the correct class depending on the user input
- Guarantees readability and consistent timing and buffers setup methods

Application Design

- Once the raytracer is created and the scene setup, the raytracer is initialised

- The raytracing algorithm is implemented by the child classes and is run in the main by the raytrace method

- Each child class can implement the raytracing algorithm as necessary
  - CPU runs is sequentially
  - GPU sets up an launches a kernel

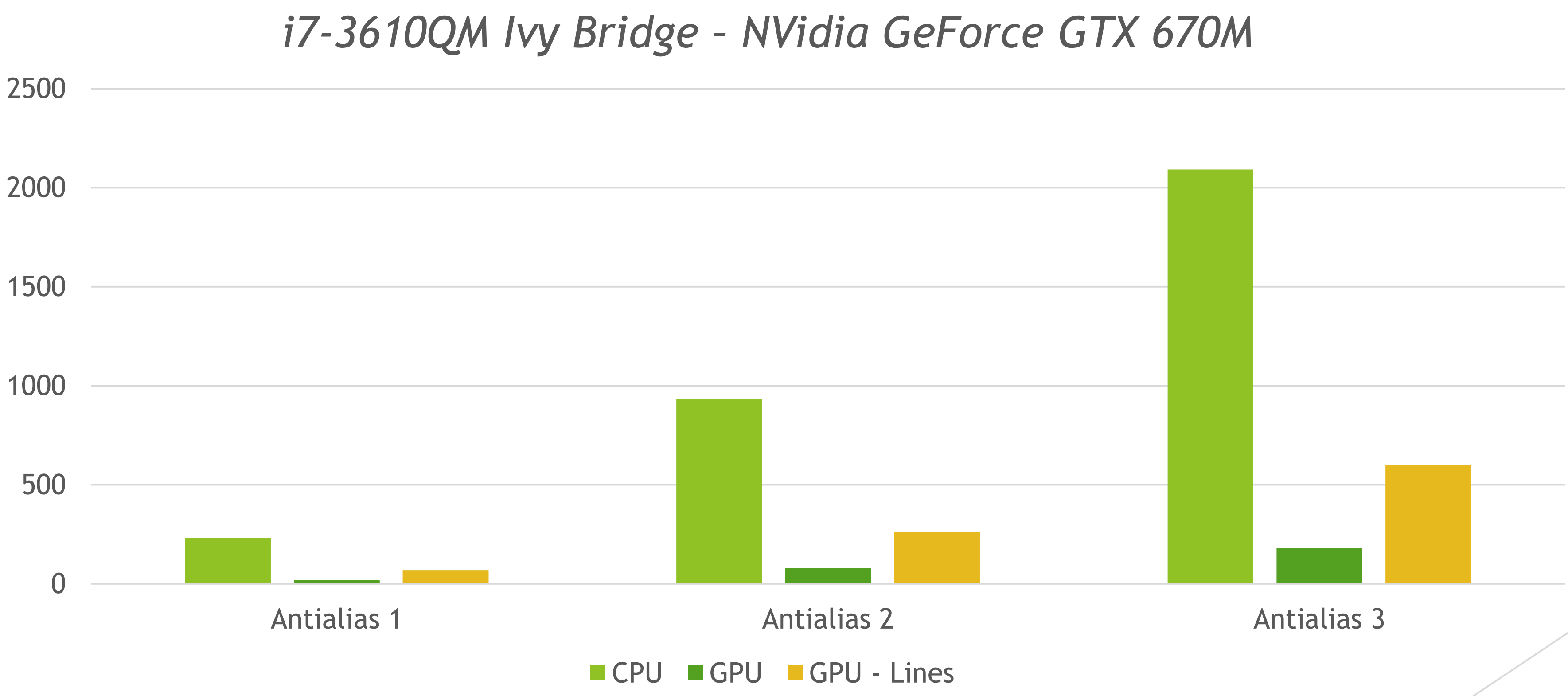
Threads synchronisation

- Demonstrated within a kernel
- Makes use of barriers when copying from global memory to local memory
- Ensures better performance
- Exploits the OpenCL memory hierarchy

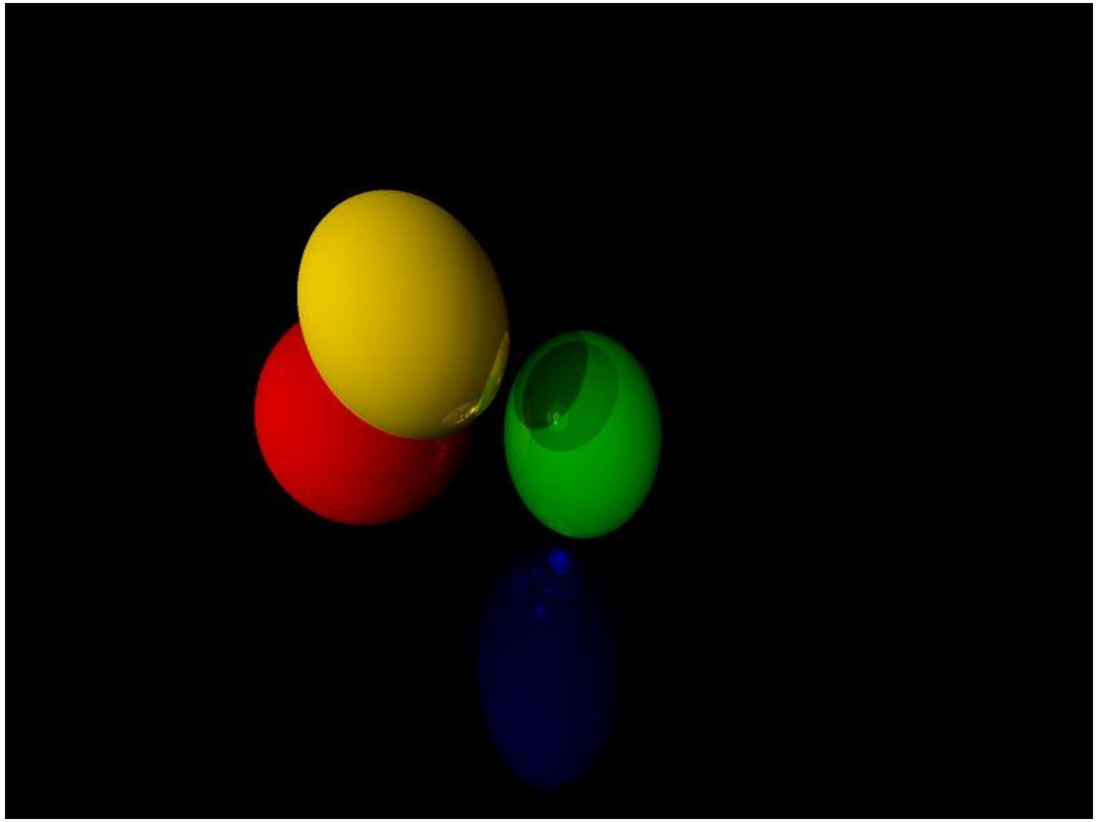
Signaling

- Demonstrated in the GPU raytracer
- The memory buffers are copied in memory and release event objects
- The kernel is enqueued only when the two events produced by the buffers have been completed and de-queued

Performance results



Result



Conclusion

- It was extremely interesting to explore the world of GPGPU
- Very rewarding to learn the raytracing algorithm
- The results obtained confirm initial hypothesis of performance improvement due to parallelism and specifically, GPGPU