An Investigation into the Speed-up of a Ray Tracer Application via the use of OpenMP and MPI

Sam Dixon
40056761@live.napier.ac.uk
SET10108 - Concurrent and Parallel Systems
School of Computing, Edinburgh Napier University, Edinburgh

The abstract goes here.

***Index Terms*—C++11, Ray Tracer, Parallel, OpenMP, MPI, Speed-up.**

I. Introduction

THE aim of this report is to document and analyse the results of an attempt to speed-up a C++11 Ray tracer application - via the use of concurrency and parallelisation techniques. The methods being tested in this project are `OpenMP` and `MPI`.

A. Ray Tracing

Ray Tracing a rendering technique that allows an image to be generated by tracing the path of a ray of light and simulating the effects that a virtual environment have upon it.

B. OpenMP

`OpenMP` (Open Multi-Processing) is an open source API that allows for the implementation of shared memory multiprocessing with minimal developmental effort. `OpenMP` makes use of the C++ `#pragma` directive and the pre-processor to allow developers to flag sectors of code to be parallelised. A number of different scheduling options can be implemented to alter the way in which `OpenMP` parallelises an application.

C. MPI

`MPI` (Message Passing Interface) is a standardised method of distributed parallelism that operates by having multiple processors communicate by sending and receiving signals from one another via communication channels.

II. Methodology

A. Profiling

Prior to implementing any methods that will save time, the sequential code must first be analysed. By using the Visual Studio Performance Profiler, it is possible to evaluate the sequential code and locate the functions or methods that use up the most CPU time. Once the potentially problematic areas have been identified, a suitable parallelisation method can be implemented to reduce the impact of those areas on the execution time. It should be noted that all code presented in the report was run without any form of compiler optimisation.

Project available at: github.com/neaop/SET10108Coursework_2

TABLE I: PC Specifications

CPU	i7-4790k 4 Core HT @ 4.00 ghz
RAM	16gb Dual Channel DDR3
GPU	Nvidia GeForce GTX 980
OS	Windows 7 64 Bit
Bandwidth	1 Gbit/s
Latency	$\sim$ 129947 ns

B. Data Collection

To ensure fair comparison and accurate results, each implementation was tested using the same parameters. Each solution was run till completion with a Sample per Pixel rate of forty and the execution time was recorded. This was then repeated one-hundred times for each application and the results were then averaged. All benchmarking was performed on the same device, the specifications of which are visible in Table I.

C. Evaluation

As well as the average execution time, speed-up and efficiency are calculated for each technique. Speed-up is defined as:

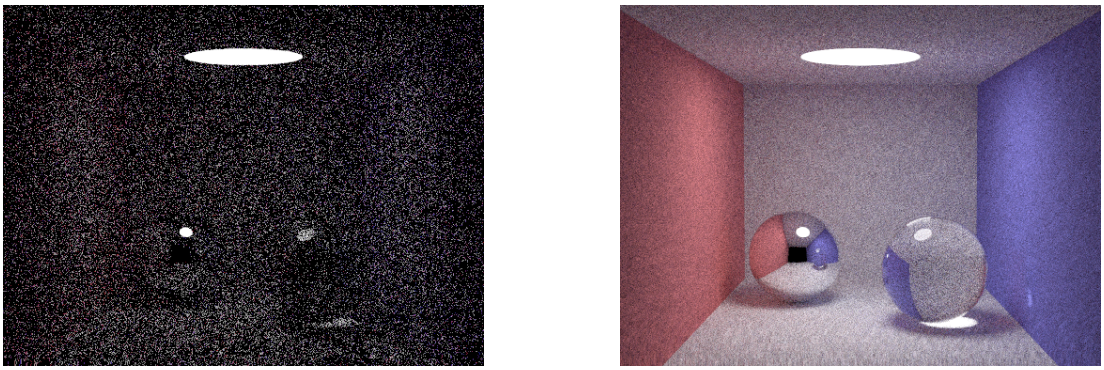
$$S = \frac{s_t}{p_t}$$

With s_t being sequential time and p_t being parallel time. Once the speed-up of a method has been calculated, the overall efficiency of the parallelisation can be measured as follows:

$$E = \frac{S}{P}$$

S being speed-up from the previous formula and P is the number of physical cores being utilized by the application.

The two equations listed above provide standardised metrics for each method or technology tested - allowing for a fair and simple comparison of the final results.



(a) Four Samples per Pixel (b) Four-hundred Samples per Pixel

Fig. 1: Two images produced by the Ray Tracer.

III. Results

IV. Conclusion

References

[1] A. Williams, *C++ concurrency in action: practical multithreading*. Shelter Island, NY: Manning Publ., 2012.