SET10108 Concurrent and Parallel Systems Coursework 1

***Index Terms*—I**

I. Introduction

Today learning how to parallelise is very important computer games can be some of the worst games and competent parallelization can allow great graphics to be run on an average pc. This allows people skilled in parallelization to be very successful in a variety of jobs very relevant today.

In this coursework the task was too parallelise a ray tracer to improve the performance using multi-threading and CPU-level parallelism. Then evaluate the performance and compare it to the serial performance. The ways that the algorithm was evaluated was through changing several aspects such as samples per pixel and size of image. These aspects were then measured through speed up and efficiency and recorded.

II. Inital Analysis

To perform the initial analysis of the program the program was run serval times changing several variables. These test showed that the algorithm performed well with small sample sizes however similarly to most serial programs the system struggled with large sample sizes showing a bottleneck exist as the time taken to complete the complete the picture increased exponentially the larger the sample per pixel this was also the case for changing the size of the image where a larger image would take a greater time. However while testing adding more objects to the scene it was noted that this had little effect on the overall time taken to complete which made sense as this did not add more pixels to the image but changed the picture itself.

A. Parallelizable Places

To look for places to parallelize the easiest places to look is at the for loops as threads allow the for loops to be run simultaneously. The largest for loop inside the algorithm was the multi for loop that calculate what colour each pixel is shown below. It was decided that this would be the part of the algorithm that would be parallelised.

```
1 for (size_t y = 0; y < dimension; ++y)
2 {
3     for (size_t x = 0; x < dimension; ++x)
4     {
5         for (size_t sy = 0, i = (dimension - y ↩
      - 1) * dimension + x; sy < 2; ++sy)
6         {
7             for (size_t sx = 0; sx < 2; ++sx)
8             {
9                 for (size_t s = 0; s < samples;↩
       ++s)
10                {
11                    //Calculation
12                }
13            }
14        }
15    }
16 }
```

Listing 1. The code that was be parallised

III. Methodology

To test the parallelisation of the ray cast algorithm three aspects of the code were tested; the number of samples per pixel, changing the size of the image and adding other objects to the scene. To keep the results consistent only one value was tested at a time. The other values were kept constant while the tests were being run. The time taken to complete the image was measured, this was repeated 10 time and an average was calculated. These test were time consuming so other computers were used however all the computers used had same graphics card and CPU to allow the tests to stay legitimate.

1) OpenMP

The first approach was to use OpenMP, OpenMP is a library to support shared memory concurrency allowing it to be portable and added easily to the program. The reason OpenMP was used over the first for loop as the first approach was because its simple to use which means it will also be rather robust as it will help the programmer tag parts of potential concurrency. It can also be used in conjunction with other approaches allowing more threading to be used.

2) Multi-Threading

3) Algorithimic Skeletons

4) CPU-Paralleism

IV. Results

A. Discussion

V. Conclusion

REFERENCES

[1] M. X, “Something great,” 2005.