SET10108 Concurrent and Parallel Systems Report for Coursework Part 1

Beej Persson, 40183743@live.napier.ac.uk
School of Computing, Edinburgh Napier University, Edinburgh

For part 1 of the coursework required for the SET10108 Concurrent and Parallel Systems module at Edinburgh Napier University a ray tracing algorithm's performance was to be evaluated and improved by utilising parallel techniques. This report documents one such investigation where the algorithm was parallelised and the difference in its performance was measured.

***Index Terms*—parallel, ray tracer, OpenMP, C++11, performance, speedup.**

I. Introduction and Background

THE aim of this report is to evaluate the performance of a ray tracing algorithm and attempt to improve this performance using parallel techniques. The ray tracer initially processes sequentially on a single core of the CPU, but by investigating different parallel techniques the algorithm was changed to run on multiple threads in an attempt to increase the performance.

A. Ray Tracer

Ray tracing is a technique used to render an image where "a ray is cast through every pixel of the image, tracing the light coming from that direction" [1]. The path is traced from an imaginary eye through each pixel on a virtual image plane and the colour of the object visible through it is calculated. It is typically capable of producing visually realistic images of high quality but at a greater computational cost compared to typical rendering techniques. Therefore it tends to be used when an image can be generated slowly ahead of time, but isn't so well suited to the real-time rendering requirements of video games.

II. Initial Analysis

Initial analysis of the base-line performance of the application and likely places that can be parallelised. [5]

The provided algorithm generates the image by iterating through each pixel using nested for loops. The ray tracer can also sample each pixel multiple times to produce a more accurate and detailed image but at the cost of processing time. Upon running the program a few times and changing the dimensions of the image produced and the number of samples per pixel an idea of its base-line performance was gathered. By its nature of currently operating sequentially, increasing either the dimensions of the image produced or the number of samples per pixel increases the time it takes to produce the image with an almost perfect positive correlation. That is to say: doubling the size of the image produced or doubling the samples per pixel doubles the time taken.

Submitted 5/11/17.

```
1 for (size_t y = 0; y < size; ++y)
2 {
3   for (size_t x = 0; x < size; ++x)
4   {
5     for (size_t sy = 0, i = (size − y − 1) ∗ size + 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      }
12    }
13  }
14 }
```

Listing 1. The many nested for loops of the ray tracer algorithm with the operations within the loops removed for clarity.

For smaller images at a low number of samples per pixel this results in reasonable times to produce an image, but when producing large images at a high number of samples per pixel the time to produce them was bottlenecked by only running sequentially on a single thread. Most of the time running the program is spent iterating through the nested for loops (seen in Listing 1) and so a clear solution to improving the performance of the algorithm is to run these for loops concurrently on multiple threads.

III. Methodology

Description and justification of the approach used and its overall suitability and rigour. [5]

TABLE I
Hardware Specifications

Processor	i7-4790K 4 Core HT @ 4.00GHz
RAM	16.0GB
OS	Windows 10 Pro 64-bit

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor

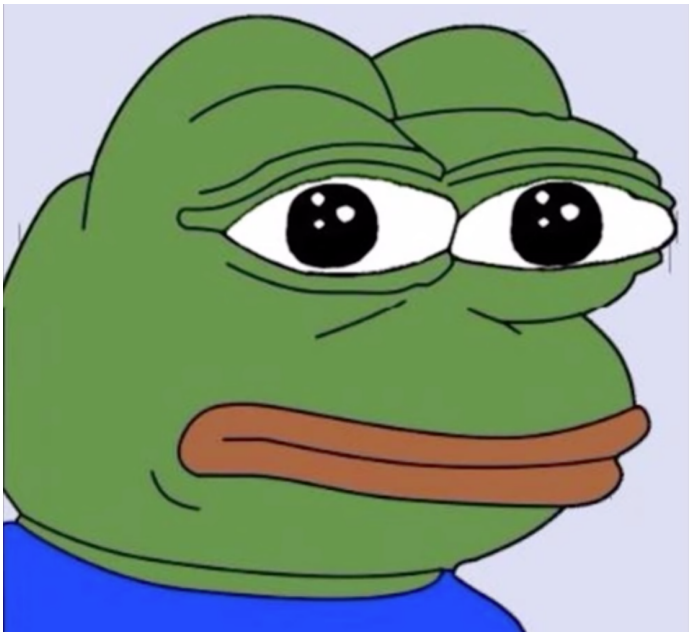

Fig. 1. A Fucking Sad Frog.

TABLE II
AN EXAMPLE OF A FUCKING TABLE

One	Two
Three	Fuck

gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

IV. Results and Discussion

Suitable performance analysis and testing documentation for the problem, including quality of presentation of the results. [10]

V. Conclusion

Level of discussion and appropriateness of the conclusions drawn based on the results gathered. [10]

VI. Conclusion

The fucking conclusion goes here.

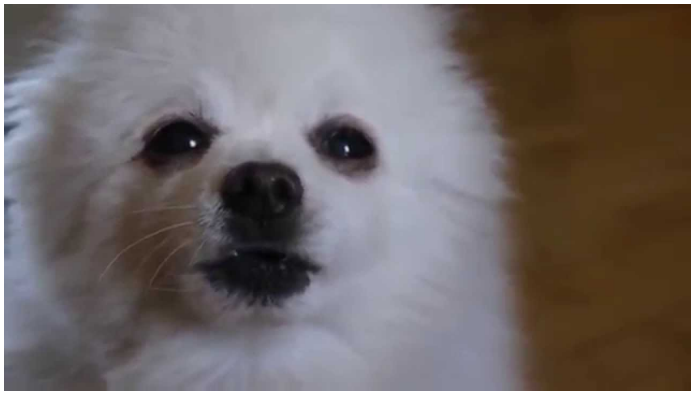

Fig. 2. A Fucking Pupper.

Fuck, this is a template for fucking IEEE transaction reports. THIS BIT OF TEXT IS FUCKING DIFFERENT. LaTeX is dead, I don't think anyone actually uses it, and we are all being strung along on a terrible joke.

APPENDIX A
PROOF OF FUCKING SOMETHING

Appendix one text goes fucking here.

```
1 #include <iostream>
2
3 int main(){
4     std::cout << "Hello World!" << std::endl;
5     return 0;
6 }
```

Listing 2. A fucking code listing.

APPENDIX B

Fucking appendix two text goes here.

ACKNOWLEDGEMENT

The author would like to fucking thank...

REFERENCES

[1] T. Nikodym, "Ray tracing algorithm for interactive applications," 2010. [Online]. Available: https://dip.felk.cvut.cz/browse/pdfcache/nikodtom_2010bach.pdf