Pavlo Bazilinskyy
Mikkeli University of Applied Sciences, Finland

Image rendering and the problem of performance

Rendering is a process of generating an image outputted to a computer screen based on a 3D model or a group of models (together commonly known as a scene) by means of computer programs. A scene normally contains geometry, viewpoint, texture, lighting, and shading information. The technical aspects of rendering methods are different, the general challenges to overcome in producing an image from a 3D representation stored in a scene are presented as the graphics pipeline along a rendering device, such as a GPU or a cluster of computers working in parallel.

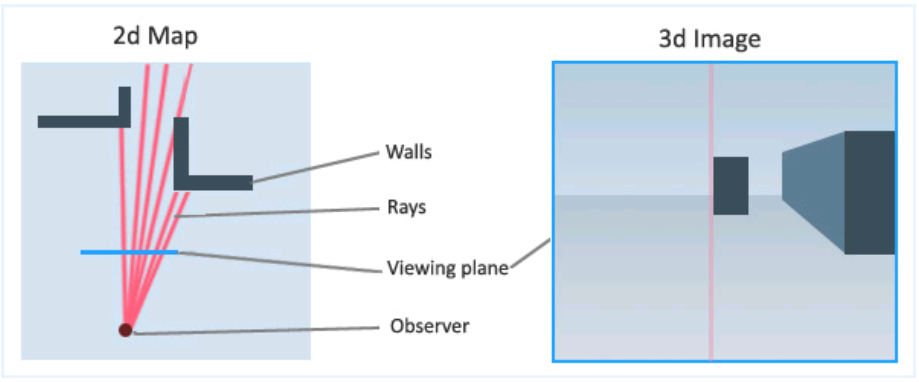

Figure 1. Raytracing explained by Anderson (2010).

Rendering is quite a challenging in terms of performance process. That is why such Message Passing Interface (MPI) libraries as OpenMPI are commonly used to overcome these challenges. According to Hursey, Mattox and Lumsdaine (2009) existing applications can easily be translated into High Performance Computing (HPC) programs using MPI level checkpoint/restart fault tolerance, which is appealing to HPC application developers that do not wish to rewrite their programme code.

In the scope of this article rendering of a simple set of spheres is taken into account; testing of possible solution implemented using a virtually running cluster of machines and a physically set-up cluster of computers is described and received performance data is evaluated.

Different ways of splitting up code to different nodes in the cluster exist in modern image rendering techniques. The first method that was discussed in the scope of this research was splitting up images on input to all nodes with ranks *[0; size)*, where *size* is a number of nodes. A problem of images being generated not entirely rose: images had unprocessed spots, the bigger number of nodes used, the smaller. As suggested by Anderson (2010) a different approach was tested instead where a node with *rank = 0* is allocated a task of memory management and is not participating in processing of the image. A formula for start and finish of each slice needs to be used first, this solution could be used: *finish = SIZE - ((rank) * (int)SIZE/(size-1)) + (int)SIZE/(size-1)* and *start = finish - (int)SIZE/(size-1)*, where *SIZE* is width of the image and *rank* is returned by OpenMPI indicating ID of the node that is working on the *[start; finish]* part of the problem. For testing 2 images, one black-and-white and one colourful are generated by the programme.

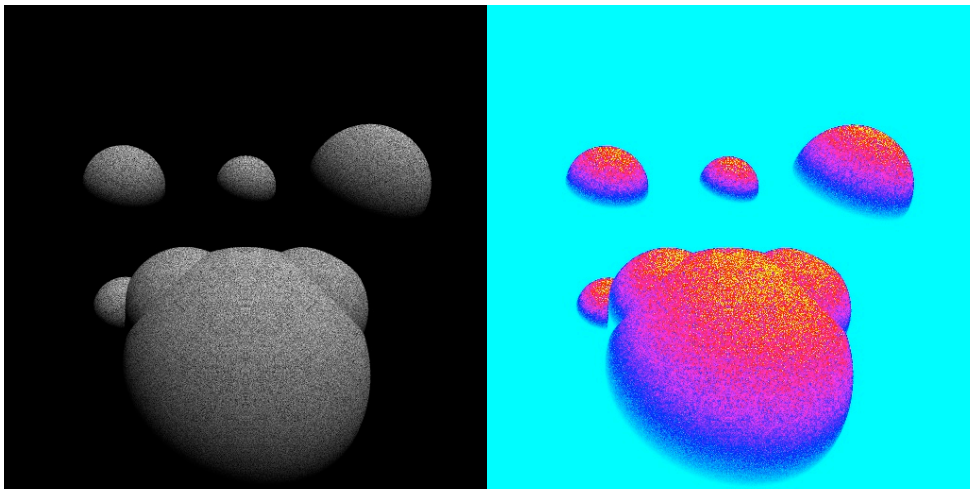

Figure 2. Rendered scene with 7 objects in PGM and PPM file formats.

In parallel computing Amdahl's law $\frac{1}{(1-P)+\frac{P}{N}}$ needs to be taken into account. It is assumed that 50% of raytracing can be given to parallel computations and therefore going over 32 nodes for executing this programme will be unreasonable. During testing of the sample programme was run on 2, 4, 8, 16, 32, 64, 128 nodes working together.

Code was tested on a cluster of 4 PCs and results were successful with an image generated on the master node and satisfactory timing data was received. Running raytracing on a single node (serially) is significantly slower than parallel computations. Results proved that calculations required for generating a black-and-white image of the scene and a colourful one do not differ significantly in terms of performance.

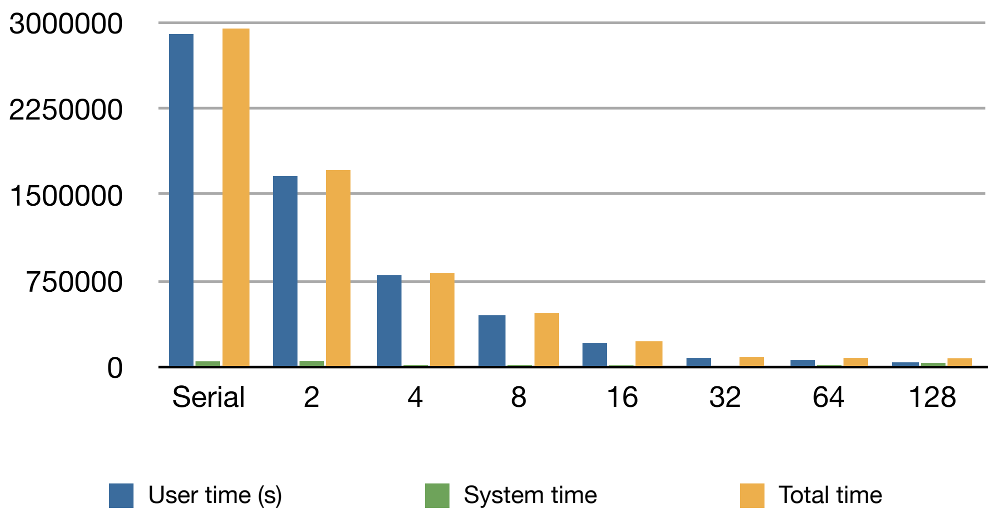

Figure 3. Timing data for MPI version of raycasting using virtual machine.

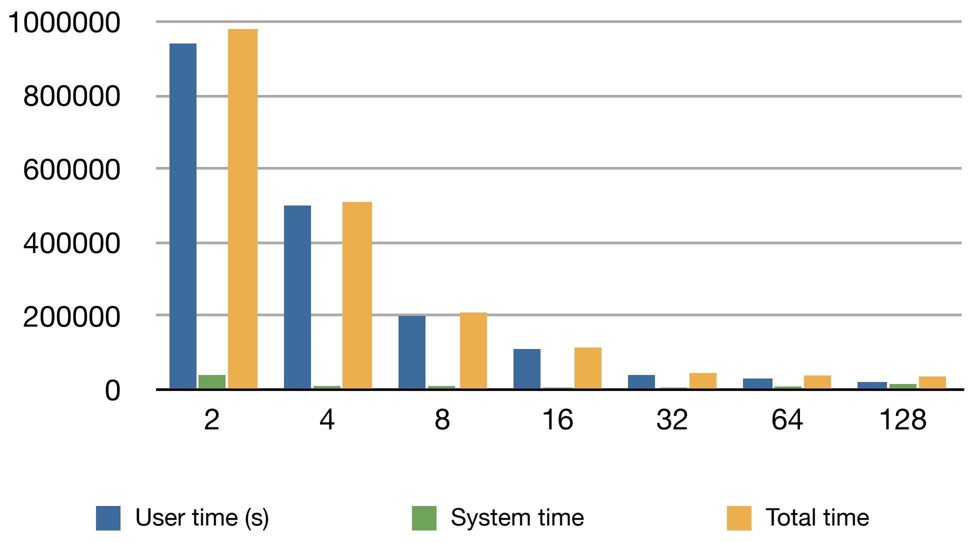

Figure 4. Timing data for MPI version of raycasting using a cluster of computers.

Results of latency, throughput and locality from testing on a virtual machine and on a real cluster were not very different. Pharr and Humphreys (2010) state that it is an expected result considering specifications of the Virtual Machine (VM) used and complexity of the programme; results of rendering of scenes used in professional development with VMs .

Piece of code for iterating through the image using OpenMPI

```
for (y=SIZE-1; y>=0; --y)
    // Iterate over every row in the image...
    for (x=0; x<SIZE; ++x) {
        double g = 0;
        for (dx=0; dx<ss; ++dx)
            for (dy=0; dy<ss; ++dy) {
            Vector d = {x+dx*1.0/ss-SIZE/2.0, y+dy*1.0/ss-SIZE/2.0, SIZE};
            Ray ray = {camera, normalise(d)};
            g += ray_trace(normalise(light_source), ray, *scene);
            }
            // Write a grayscale value to the PGM file.
            fputc((int)(0.5+255.0*g/(ss*ss)), fp);
     }
```

Literature:

1. **Joshua Hursey, Timothy I. Mattox, and Andrew Lumsdaine.** Interconnect Agnostic Checkpoint/Restart in Open MPI. In Proceedings of the Eighteenth International Symposium on High Performance Distributed Computing (HPDC 2009). ACM - June 2009.
2. **Matt Pharr, Greg Humphreys**. Physically Based Rendering. 2nd ed. Morgan Kaufmann - 2010.
3. **Greg Anderson.** What is ray casting, or a raycast engine? 2010. Referred 09.12.2011. Available in www-format: <http://www.tragos2d.com/ray-casting>.