LINUX CLUSTER POSSIBILITIES IN 3-D PHOTO QUALITY IMAGING AND ANIMATION

Arjun Jain [*] Himanshu Agrawal [†] Nalini Vasudevan [‡]

Abstract

In this paper we present the PC cluster built at R.V. College of Engineering (with great help from the Department of Computer Science and Engineering). The structure of the cluster is described and the performance is evaluated by rendering of complex 3D Persistance of Vision(POV) images by the Ray-Tracing algorithm. Here, we propose an unexampled method to render such images, distributedly on a low cost scalable **Linux Cluster Supercomputer**

Keywords: *PC cluster, parallel computations, Ray Tracing, Persistance of Vision, rendering.*

1 Introduction

Computational image rendering is widely used for generating images and graphics for the motion pictures industry and bio-modelling. From the mathematical point Ray-tracing is a rendering technique that calculates an image of a scene by shooting rays into the scene. The scene is built from shapes, light sources, a camera, materials, special features, etc. For every pixel in the final image one or more viewing rays are shot into the scene and tested for intersection with any of the objects in the scene. Viewing rays originate from the viewer, represented by the camera, and pass through the viewing window (representing the final image). Every time an object is hit, the color of the surface at that point is calculated. For this purpose the amount of light coming from any light source in the scene is determined to tell whether the surface point lies in shadow or not. If the surface is reflective or translucent new rays are set up and traced in order to determine the contribution of the reflected and refracted light to the final surface color, but in order to achieve results comparable with those generated by conventional techniques experimental we need supercomputers power or parallel computers cluster. In this way such tasks are solved during the last decade of 20th century.

Clusters based on PCs running Linux have become the cheapest supercomputers in the academic and commercial field. We created such a cluster at R.V. College(CSE, RVCE). The clusters performance have been tested by generating many such benchmark images.

2 Personal Computer Cluster

With the power and low prices of today s PCs and the availability of 100 Mb/s Ethernet interconnect, it makes possible to combine them to build High-Performance-Computing and Parallel Computing environment. Today, there is a wide range of switches available, ranging from 8 to over 100 ports, some with one or more Gigabit modules that lets you build large systems by interconnecting such switches or by using them 5 with a Gigabit switch. Switches have become inexpensive enough, so there is not much reason to build your network by using cheap hubs or by connecting the nodes directly in a hypercube network. The local area PC cluster was made in the RVCE Hostel with the help of Department of Computer Science, RVCE for image processing and rendering. PC cluster of 7 nodes:(as shown in figure)- all of them single AMD 1.8 G Hz processors; all have 128 MB RAM, a 40 GB disk drives.

Machines have been installed with the standard RedHat 9.0 and WinXP(parallel cluster runs in Linux OS at this time). All PCs are assumed to boot from a their own hard drive and have a fast Ethernet network (100 Mb/s) connection to a switch controlling the private cluster network. The suggested range of addresses for a private network is from 192.168.100.1 to 192.168.100.20. Nobody is able to connect directly to a compute node from outside this network any-

[*]Development Engineer, PI Corporation, Bangalore, India 560080 Tel: +91 99451 24241 Email: arjun@picorp.com

[†]R.V. College of Engineering, Electrical & Electronics Engineering, Bangalore, India 560059 Tel: +91 98862 19916 Email: himanshu.rvce@gmail.com

[‡]Software Developer, Yahoo Inc., Bangalore, India 560001 Tel: +91 94481 07482 Email: naliniv@yahoo-inc.com

Figure 1: The 'Linux Cluster Supercomputer'

way. This keeps normal traffic from interfering with inter-node communication and vice versa. Thus the total memory 896 MB; total disk space 280 GB. We use custom made PVM-Pov Cluster software - a high-performance parallel image rendering environment for workstation and PC clusters [3]. PVM-Pov uses the PVM 3.1 [1] high performance communication library - a dedicated communication library for cluster computing (it allows using many types of networks). Communication library used is faster than usual TCP protocol. In addition a trunking is possible (installation and use of up to 4 NICs). Under the PVM virtual machines users are not aware whether or not a system is a cluster of single/multi-processor computers or a cluster of clusters. PVM system has check point function. It allows restart function and improves a reliability of the system. Available compilers are gcc and g++ compilers C, C++ and HPF. The cluster differs from the network of workstations in security, application software, administration, booting and file systems. Application software uses underlying message passing system like Parallel Virtual Machine (PVM). There are many ways to express parallelism, but message passing is the more effective and more modern. For administering the cluster we have a login node with a keyboard, monitor and mouse(as shown in the figure). Other nodes can be headless (no keyboard, mouse, or monitor) but in our way there are all machines like login node, because our cluster is made on the basis all equal class. Any machine can be logged into from the login node using secure shell(ssh) and can be administered.

3 Design of Distributed Ray Tracing Cluster

Image generation using Ray-tracing is a is a rendering technique that calculates an image of a scene by shooting rays into the scene. The scene is built from shapes, light sources, a camera, materials, special features, etc. For every pixel in the final image one or more viewing rays are shot into the scene and tested for intersection with any of the objects in the scene. Viewing rays originate from the viewer, represented by the camera, and pass through the viewing window (representing the final image). Every time an object is hit, the color of the surface at that point is calculated. For this purpose the amount of light coming from any light source in the scene is determined to tell whether the surface point lies in shadow or not. If the surface is reflective or translucent new rays are set up and traced in order to determine the contribution of the reflected and refracted light to the final surface color.

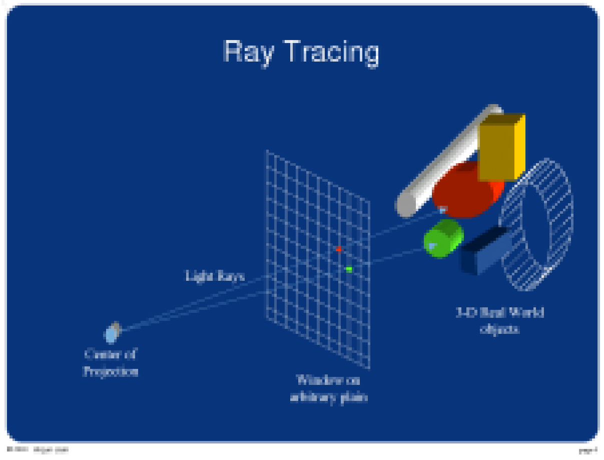

Figure 2: 'Ray Tracing'

Now this is a very heavy and time-taking job for the processor. For every pixel atleast three to four rays are shot and tested for intersection. Each ray has its equation and each equation has to be solved for intersection. Each pixel needs to be rendered and simple ray tracing can be done for a uniprocessor system using the following algorithm:
Select center of projection and window on viewplane

1. for(each scan line in image)

    (a) for(each pixel in scanline)

i. determine ray from center of projection through pixel;

ii. for(each object in the scene)

A. if(object is intersected and is closest considered thus record intersection and object name;

B. set pixel s color to that at closest object intersection;

Now, instead of rendering the entire image on the same system we here use a clustered system in the following way: the image is broken up into still smaller blocks. Each slave is to then render one or more of these blocks independently and send the rendered image back to the master where all the blocks are integrated to form the final image.

This can be explained using the following algorithm. This is an algorithm to add 75 integers on a clustered system with 3 slave nodes and one master node.

Algorithm for the master program:

```
initialize the array  items .

/* send data to the slaves */
for i = 0 to 3
     Send items[25*i] to
     items[25*(i+1)-1] to slave Pi
end for

/* collect the results from the slaves */
for i = 0 to 3
     Receive the result from
     slave Pi in result[i]
end for

/* calculate the final result */
sum = 0
for i = 0 to 3
     sum = sum + result[i]
end for
print sum
```

Algorithm for the slave programs:

```
Receive 25 elements from the master
in some array say  items

/* calculate intermediate result */
sum = 0
for i = 0 to 24
     sum = sum + items[i]
end for
send  sum  as the intermediate
result to the master
```

Here each slave is given the task of adding 1/3 of the total no of integers and the final result of these 25 addition is sent to the master 10 where it adds the 3 results from the slaves to get the final result of the addition of 75 integers. Thus the job is done in paralell.

4 Results

The algorithms described and developed were used to generate many benchmark 3-D images such as the famous skyvase.pov, chess.pov and blob.pov. All the experiments under consideration here were carried on clusters with 4, 3, 2, and single processor systems.

4.1 skyvase.pov

The benchmark SKYVASE rendered at 1024x768 resolution is as shown:

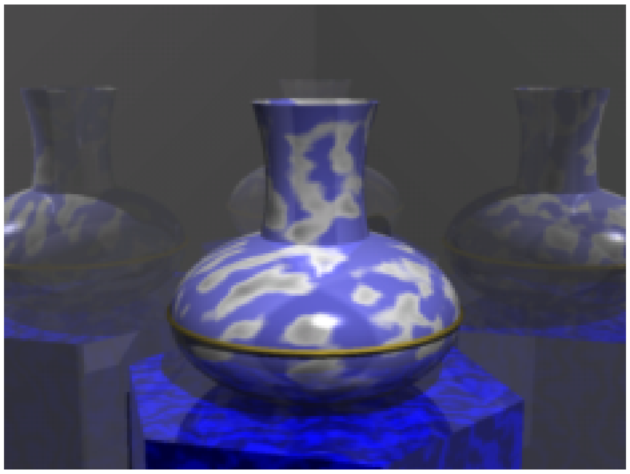

Figure 3: 'SKYVASE.POV'

This image takes approximately 14 seconds in a quard processor system and 17 in the tri-processor cluster system.

In a uniprocessor system the same takes 38 seconds and in a bi processor system 20 seconds thus with a degradation of only 2% to 14%.

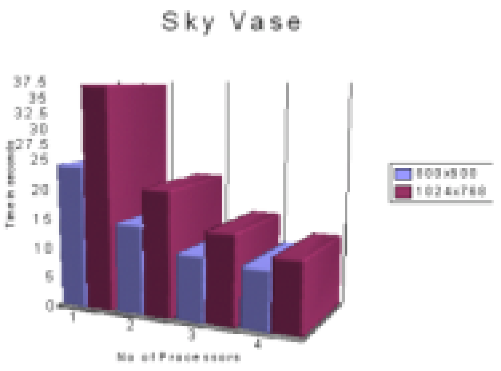

Figure 4: Results of rendering skyvase.pov on a cluster with varying number of processing slave nodes

4.2 CHESS.POV

This is a more complex nad heavier image than the SkyVase. The benchmark CHESS 2c rendered at 1024x768 resolution is shown bellow:

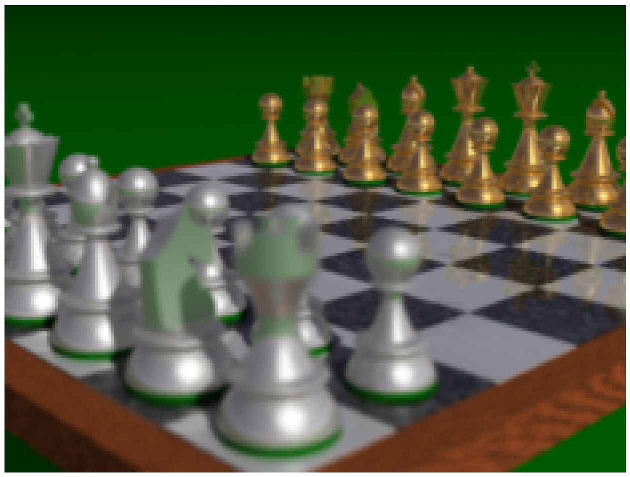

Figure 5: 'CHESS.POV'

This image takes approximately 150 seconds in a quard processor system and 210 in the tri processor system.

In a uniprocessor system the same takes 580 seconds and in a bi processor system 350 seconds thus with a degradation of only 12% to 14%, where degradation is given by

$$\frac{time-taken-on-n_i-processors}{time-taken-on-n_j-processors} \times \frac{n_i}{n_j}$$

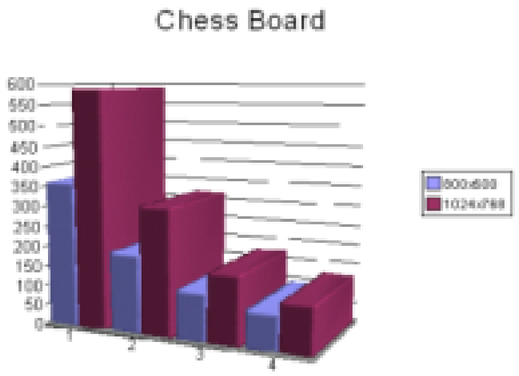

Figure 6: Results of rendering chess.pov on a cluster with varying number of processing slave nodes

References

[1] Al Geist, Adam Beguelin, Jack Dongarra, Robert Manchek, Weicheng Jiang and Vaidy Sunderam, *PVM: Parallel Virtual Machine - A User s Guide and Tutorial for Networked Parallel Computing,*, MIT Press. Available at http://www.netlib.org/

[2] Marc Snir, Steve Otto, Steven Huss-Lederman, David Waker and Jack Dongarra, *MPI: The Complete Reference*, MIT Press. Available at http://www.netlib.org/.

[3] Clay Breshears and Asim YarKhan, Joint Institute of Computational Science, University of Tennessee, USA. www-jics.cs.utk.edu/PVM/pvm/ guide.html, *A Beginner s Guide to PVM Parallel Virtual Machine.*

[4] Emily Angerer Crawford, O ce of Information Technology, High Performance Computing, www.hpc.gatech.edu/seminar/pvm.html, *PVM:An Introduction to Parallel Virtual Machine.*

[5] Arjun Jain, Harish G. Naik, "Rendering of 3D images using PVMPov", *http://linux-bangalore.org/2003/schedule.*

[6] http://clusters.top500.org/db