Ray Tracing in a Distributed Environment*

Kelvin Sung   Jason Loh Jen Shiuan   A. L. Ananda
Department of Information Systems and Computer Science
National University of Singapore
Email: {ksung|lohjensh|ananda}@iscs.nus.sg

Abstract

The results of parallelizing a classical ray tracer in a typical LAN-connected workstation environment is presented. The motivation of this work is to utilize the idle workstation cycles at night. Issues involved in distributing the processing of a ray tracer are discussed and our solutions are presented. Performance tests are carried out on a group of homogeneous and heterogeneous workstations. Some unexpected problems were encountered and our approaches are described. Initial results showed that the achievable speedup is comparable to some of the parallel approaches on dedicated MIMD systems. All 80 instructional workstations in our department were utilized without causing significant degradation in speedup.

Keywords: Distributed application, Ray tracing,
Asynchronous RPC, Load Balancing

Subject Descriptors: C.2.4 Distributed Applications
I.3.7 Raytracing
I.3.1 Parallel Processing

1 Introduction

It is well known that the ray tracing rendering approach has generated some of the highest quality images and that the computations involved are extremely demanding. Also well known is that tracing of the rays in the algorithm are independent of each other and that the algorithm can be easily parallelized[1]. There are numerous work done in investigating parallel ray tracing algorithms on systems such as the vector processors (e.g. [29, 12]), multiprocessors (e.g. [13, 11, 5, 30, 21, 6, 3]), or special purpose hardware (e.g. [16, 7, 19, 28]).

Workstations connected by a local area network (LAN) (e.g. the ethernet) are typical in many office working environments. It is observed that the utilization of these workstations are typically very low [23]. The LAN connected workstations can be perceived as a loosely-coupled multiprocessor system. In this way, it is obvious that one of the most cost effective ways to parallelizing a ray tracing system is to make use of the spare CPU cycles on the

*This work was supported by the National University of Singapore under grants RP900608 and RP930616.

[1]Ray tracing has been referred to as *embarrassingly parallelizable*.

idling workstations. It is interesting that there is a limited list of publications ([24, 34, 27, 4]) describing the work done in this area. We believe the reason is that the problem at hand is considered to be too simple, as a result very few researchers from the graphics community investigated in this area.

Although implementing a ray tracer in a LAN-connected environment may seem straightforward, as discussed in Section 3, there are some questions that we have to answer. Some of these questions are standard issues concerning a typical distributed application. In our case, these questions are important because understanding them allows us to utilize the idling workstations in our department efficiently, and more importantly, to be able to predict and to control the performance of the system.

In this paper, we summarize some of the interesting results of our investigation in parallelizing a *classical* ray tracer in a *typical* local area network environment [32]. In order to be objective in our investigation, we have adopted and parallelized the Rayshade system [22], a well received public domain ray tracer. Based on the survey in Section 2 and the evaluation Section 3, we have chosen the ASTRA asynchronous RPC system [1] as our network communication tool. We tested our distributed Rayshade System on all 80 of the instructional workstations in our department and speedups of more than 64 times were achieved for some environments. It is interesting that our results are comparable to some of the parallel approaches on dedicated MIMD systems (e.g. [21, 3, 5]), however our underlying hardware is essentially free.

This paper is organized as follows. Section 2 presents a survey of the past parallel ray tracing approaches in distributed environments. Section 3 discusses the issues involved in developing a distributed[2] ray tracing system. Based on the discussion, the requirements for a network communication tool is derived. The section following that describes the implementation of our system. Section 4 presents some of the interesting results from our study. Finally, Section 5 concludes the paper.

2 Survey of the Past Work Done

From the very beginning, when Whitted introduced ray tracing as a high quality image generation technique, he pointed out that the tracing of different rays are independent processes and that they could be executed concurrently [36]. This property has been widely exploited in the past decade. As pointed out, there are numerous work done in parallelizing ray tracing applications for parallel and special purpose hardware systems. We refer to these categories of approaches as the *tightly coupled parallel* approaches[3]. Please refer to [2, 17, 20] for some excellent surveys on the past parallel ray tracing work. Our work is different because we do not rely on any special parallel (or custom) hardware system. In this section, we describe the past approaches to tracing rays in distributed environments. The section is summarized with an example to illustrate the difference between a tightly coupled parallel system and a distributed system.

[2]In this paper, we use *distributed ray tracer* to refer to a parallel ray tracing system in a LAN-connected workstation environment. This should not be confused with Cook's [9] work.

[3]For example, the Hypercube is a tightly coupled distributed memory parallel MIMD system.

2.1 Previous Distributed Ray Tracing Systems

As in all parallel ray tracing work, the previous distributed ray tracing approaches can be classified, according to the granularity of parallelism, into two broad categories: *frame based*, and *image based*. In the *frame based* approach, each processing element is responsible for the generation of an entire image frame. In this way, all the processing elements cooperate to generate a series of images typically for animation purposes. On the other hand, *image based* approaches coordinate all the processing elements to cooperate in generating one image.

2.1.1 Frame Based Approaches

Leister et al. [24] used a network of 30 Sun workstations connected by the ethernet to generate a five-minute ray traced animation sequence. Their system has a *client* that is responsible for sending jobs to the servers (to compute one image) and collecting the results. It is reported that a total of three years of Sun CPU time was consumed in two months. Stober et al. [34] described the VERA system, where a similar approach is adopted for photorealistic animation with frame partitioning. They also addressed the issues involved in fault tolerance and recovery at the image level.

We observed that both of the approaches were designed specifically for generating animation sequences. These work does not take advantage of the ray level parallelism. The major drawback of frame-based distribution is that it cannot accelerate the time to generate one image. This is because the minimum task size for each processor is one image.

2.1.2 Image Based

Muuss [27] described the REMRT system where a central client dispatches workload, in 3-scanline units, to the servers for the actual computation. The underlying communication layer used is the package (PKG) protocol [26]. Muuss tested his system on 10 SUN-3/50 systems. Although the results are promising with little communication overhead, because of the small number of servers involved, it is difficult to draw significant conclusions from his work. For example, it is impossible to predict the number of machines that can execute in parallel without saturating the system. In addition, the fix three-scanline task unit does not address the question of how does varying the workload size affects the performance of the system. Lastly, the maximum number of servers that can be supported are limited to 64. This is because the BSD UNIX limits the number of opened files for a single process to 64 [15]. These files are required by the client for maintaining an active network connection to each of the server machines.

Bloomer [4] described an implementation of a networked ray tracer using the SUN RPC system. Bloomer's system also adopted the client/server model where the client dispatches workload (in blocks of scanlines) to the servers for the actual computation. The sun RPC is a synchronous system. This means that after assigning a job to a server, the client will be blocked until the results are returned. This severely limits the attainable parallelism. Bloomer resolved this problem by forking a child process for each server workload assignment. The forking of child processes is extremely costly and thus the overhead incurred for each call to the server is substantial. Another disadvantage is that there is usually a limit to the number of child processes that can be created simultaneously. This would limit the maximum number of servers that can be supported.

Bloomer used the previous processing time of a server to predict the complexity and thus accordingly adjust the size of the subsequent task to be assigned to the server. This approach only works if the *just-completed* task and the *next-to-be-assigned* task contain scanlines from the similar region of an image. In general, if the scanlines are assigned to different servers in a sequential order (e.g. 1^{st} server process 1^{st} scanline, 2^{nd} server process 2^{nd} scanline, etc.) and if there are a large number of servers, then this approach does not work well.

It is interesting that both implementations took advantage of the Network File System (NFS) and treated the remote disk storage as a form of shared global memory. In this way, the client and the servers do not need to perform explicit communications on behalf of the scene database. After the servers are done processing, to avoid simultaneous writing by different servers, the results are sent back to the client for image re-construction.

2.2 Discussion

More recently, Singh et. al. [33] discussed their work in porting a form factor calculation algorithm designed for a MIMD system directly to a LAN connected workstation environment. They reported that because of the much longer message latency, the performance of the distributed version is *worst* than that of one machine executing sequentially. This serves as an excellent example that: although a LAN connected workstation environment is, in theory, similar to a distributed memory parallel system, in reality, the algorithms should not be mapped directly. The two parallel approaches must be studied separately. Although it is important to understand the issues involved, some of the solution approaches may not be compatible.

3 Ray Tracing in a Distributed Environment

This section begins by examine the issues involved in implementing a typical distributed application [25] from the ray tracing point of view. We then identify the requirements of a communication tool in a LAN environment to efficiently support the implementation of a distributed ray tracer. It should be noted that the issues discussed are *not* orthogonal. For example, the granularity of parallelism adopted should depend on the efficiency of the underlying communication channel. Fine grain parallelism can only be considered if the communication channel is very efficient. Although the following issues are discussed separately, one should always remember to consider the other factors. After the background discussion, we present our choice for the distribution tool, and for the ray tracer. The section concludes with a description of our distributed ray tracer implementation.

3.1 Issues Involved in Designing a Distributed Ray Tracer

The following discussion assumes a group of workstations with different performance level connected by a typical local area network (e.g. the 10Mbps ethernet).

1. **Communication.** Due to the high processor speed to communication overhead ratio, when developing a parallel algorithm, it is important to minimize communication, especially exchanging large amount of data.

2. **Granularity of Parallelism.** For a ray tracing application, there are basically three levels of parallelism: frame based, image based, and ray based. The former

two are extensively discussed in in Section 2.1.1, and Section 2.1.2. A ray based approach is fine grain parallelism, where all the workstations cooperate to process one (or a few) rays. Typically, the long message latency prohibits systems developed in distributed environments to support a fine grain parallelism.

3. **Load Balancing and Size of Tasks.** A distributed environment supports a client/server model with demand driven load distribution naturally. In this case, one or more clients will manage the pool of tasks, and the servers will be responsible for performing the actual computation and for requesting new tasks whenever they become idle. The size of each individual task is very important because small task size may flood the communication channel, while large task size may cause unbalanced of load. In the worst case, the entire application's completion time may be limited by one server processing the last complicated task with all other servers left idling.

4. **Database Storage.** In a LAN environment, the Network Filesystem (NFS) caches the most recently referenced data [10]. By taking advantage of the NFS, not only each workstation will contain a local cache of the scene database, the implementation of the system will also be straightforward.

5. **Heterogeneous Environment.** In a LAN environment, the performance of the connected machines may span a wide range. There are two major implications:

   - The size of a task assigned to a workstation should be related to the performance of the machine. The task assigned to the faster machines should be more demanding than the ones assigned to the slower machines.
   - Synchronization among all the workstations should be avoided. This is because it is difficult to anticipate the completion time of the given tasks for all the workstations.

6. **Bottleneck Identification.** For a distributed environment, we can usually expect the slow communication channel to be the bottleneck.

7. **Programmability.** This is probably the most important factor when designing any software system. When considering the approaches for the above issues, it is very important to maintain the simple and yet elegant model of ray tracing programs.

3.2 Requirements of the Communication Tool

The following is a list of important functionalities that a communication tool should provide in order to support an efficient distributed ray tracing system.

1. **Asynchronous Communication.** One of the major shortcomings of Bloomer's system was the expensive forking of the child process [4]. This could be avoided if the underlying communication tool was asynchronous.

2. **Number of Servers.** The communication tool should not limit the number of servers that the client can simultaneously communicate with. This was one of the major shortcoming of Muuss's work [27].

3. **Simple Programming Paradigm.** As in all cases, it is important to present a simple parallel programming model for the system developer. For example, the Sun RPC uses the procedure calling paradigm which most programmers are familiar with.

4. **Multicast Broadcasting.** Often times, the client needs to send identical messages to all the servers (e.g. informing all the servers to initialize the ray acceleration structure). It is important for the underlying communication tool to support an efficient broadcasting protocol.

5. **Fault Tolerance.** As number of processing element increases, the probability of failure also increases. This is especially the case in a distributed environment where the workstations may be physically far apart. For example, a student may decide to reboot a workstation because his/her program is hanging. It is important to detect the failure of a server and to re-assign the task to other servers.

6. **Availability.** A LAN usually connects a heterogeneous platform of workstations. It is essential that the communication tool is available on all of the platforms.

3.3 Designing a Networked Distributed Ray Tracer

3.3.1 The Distribution Tool

The ASTRA asynchronous RPC was developed to achieve high-parallelism while retaining the simplicity and familiarity of the RPC abstraction [1]. A distributed ray tracer implemented with this system is expected to be efficient because of the low latency of the calls and the long computational time in ray tracing.

The following are distinct features of the ASTRA RPC that are important for our purposes:

- ASTRA RPC supports the familiar procedure calling programming paradigm.

- ASTRA allows the client applications to issue asynchronous calls to invoke the servers. The reception of the replies can be deferred until the client requires the results of the call.

- ASTRA is designed to be transport independent; it does not rely on any specific transport protocol. For inter-machine calls, two types of transport protocol are supported; virtual circuit and reliable datagram.

- ASTRA has been designed to treat intra-machine calls differently. It optimizes these calls by using the most efficient native IPC mechanism, thus avoiding overhead due to data conversions and network protocols.

- ASTRA is developed on top of the sockets which makes it easily portable. Currently, ASTRA is available on the SUNOS, System V Version 4 based UNIX Systems, PC/MSDOS using PC/TCP Toolkit Version 2.04, and the IRIX platform[4].

- ASTRA does not limit the number of servers that a client can simultaneously connect to.

[4] A version of the ASTRA system is available under anonymous ftp from ftp.nus.sg in the directory /pub/NUS/ISCS/SHILPA.

3.3.2 Choice of a Ray Tracing System

It is important that the study we conduct is unbiased. If we were to develop an original ray tracer, it would be difficult to avoid biased optimizations. Furthermore, as a secondary objective, we would like to demonstrate that the parallelizing effort is straightforward and can be achieved easily. For these reasons, we have adopted the Rayshade Version 4.0.6 by Craig E. Kolb [22][5]. Rayshade is designed to be reasonably fast, portable and easy to modify. It has been tested on many UNIX-based systems (e.g. the SGI 4D, IBM RS6000, Sun Sparcstation 1, etc.). A detailed description of the above features can be found in Rayshade User's Guide and Reference Manual by Craig E. Kolb in the Rayshade package.

3.3.3 Our Distributed Rayshade Implementation

Similar to Muuss [27] and Bloomer [4], we have implemented the image level parallelism, where all the workstations cooperate to generate one image. We have also adopted the client/server model, with a demand driven load distribution strategy. The original Rayshade program is separated into the client and the server portions at the outermost loops of the ray tracer. The client loops over all the pixels of an image, dispatching tasks (in scanline units) to the servers. The servers compute the colors for the assigned image pixels and return the results to the client. In this way, complicated image file locking mechanism can be avoided during image re-construction.

The demand driven load distribution model is implemented by having the client automatically dispatching the next task to the server that returned the pixel colors. After some experimentation[6], and to avoid complicated task assignment and image re-construction algorithms, the task size is restricted to a constant of one scanline. The first task for each server is assigned statically, mapping the first N tasks to the N number of servers. Subsequently, whichever server that finishes its task will be assigned the following scanline. In this way, the servers may not work on a contiguous block of scanlines.

All the servers construct the entire ray acceleration structure during the initialization stage. The scene database access relies entirely on the network transparent NFS. The ASTRA RPC system does not support multicast broadcasting. As a result, the only alternative to the NFS is for the client to send the entire scene database to each of the servers, one at a time.

4 Results

The workstations that are involved in our performance studies include the SGI Indigo R3000s and R4000s, the Sun ELC systems, and the Sun Sparc Classic (SSC) systems. The SPEC92 benchmarks [14] and the configurations of these machines are tabulated in Table 1. The network that connects all the machines is the 10Mbps ethernet. All the workstations are within the same segment. This segment is connected to a 100 Mbps FDDI backbone. To compare the performance of the original sequential Rayshade and that of our networked distributed version we have defined *Speedup* as:

$$Speedup = \frac{SequentialExecutionTime}{DistributedExecutionTime}$$

[5]This system is available under anonymous ftp from princeton.edu.

[6]Please refer to the discussions in Section 4.2 for the details of the experiments.

Workstation	Processor Speed (MHz)	SpecInt92	Specfp92	Main Memory (MegaBytes)	Local Disk
SGI Indigo R4000	50	57.6	60.3	32	1.2 GB
SGI Indigo R3000	33	22.4	24.2	16	diskless
SUN Sparc Classics	50	26.4	21.0	16	207 MB
Sun ELC	33	18.2	17.9	16	diskless

Table 1: The workstations and their configurations.

DataBase	Tetra	Sphere	Mount	Rings	Gears
Time (Sec)	1765.82	27612.79	12248.01	16480.32	40133.63

Table 2: Sequential Execution Time (On a Sun SSC).

The distributed execution time includes the reconstruction of the images from the return pixel values.

4.1 The Benchmark Testing Environments

The testing of our system was performed on the Standard Procedural Database (SPD) package as proposed by Haines [18]. We present the performance timing for five of the scene databases: the Tetrahedral Pyramid (Tetra - 4096 primitives), the Sphereflakes (Sphere - 7392 primitives), the Fractal Mountains (Mount - 8196 primitives), the Rings (8401 primitives), and the Gears (9345 primitives). All the images were generated at 512x512 with 16 samples per pixel. Table 2 summarizes the execution time of the original Rayshade system running on our Sun Sparc Classic (SSC) system. Please refer to [18] or [35] for detailed descriptions and analysis of the SPD databases.

4.2 Varying Task Size with Fixed Number of Servers

Since the size of a task does not reflect the amount of computation involved, an optimum task size should be determined to compromise between balance of load and communication overhead. Figure 1 shows a plot of the number of scanlines per task vs the execution time with 38 SSC servers. It is interesting that the results are consistent for all four test scenes: the optimum performance is achieved when the task size is one scanline. This implies that the computational time per task is dominating and that the communication overhead is relatively insignificant.

We observed that the curves in Figure 1 flatten out as the size of each task is reduced to one scanline. This suggests that further reducing the task size will not further decrease the execution time. The reason is because of the good load balance achieved. Figure 2 illustrates this situation for the Sphere environment. The '$\diamond$' markers in Figure 2 indicates the completion time of a task by the corresponding server. We observed that at one scanline per task, the load is evenly distributed among all the servers.

4.3 Speedup with Increasing Number of Servers

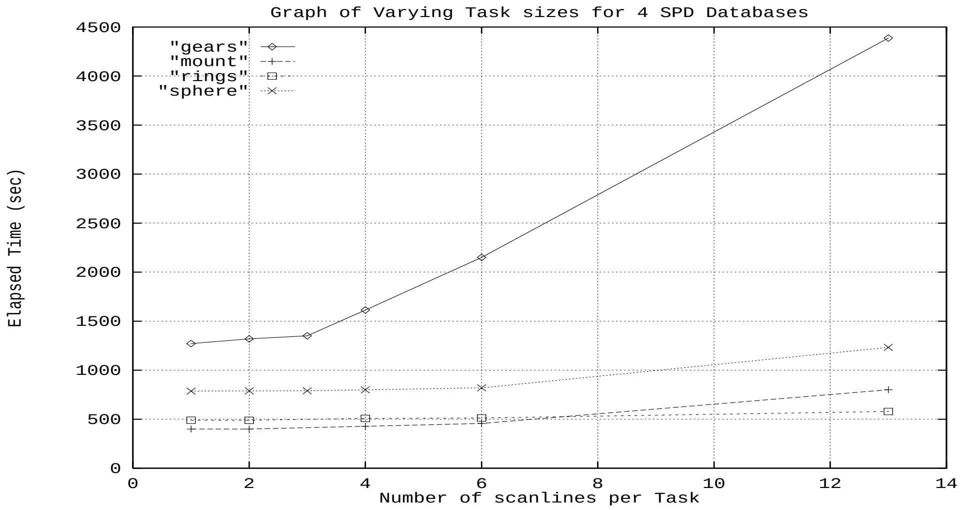

Figure 1: Optimum Task Size.

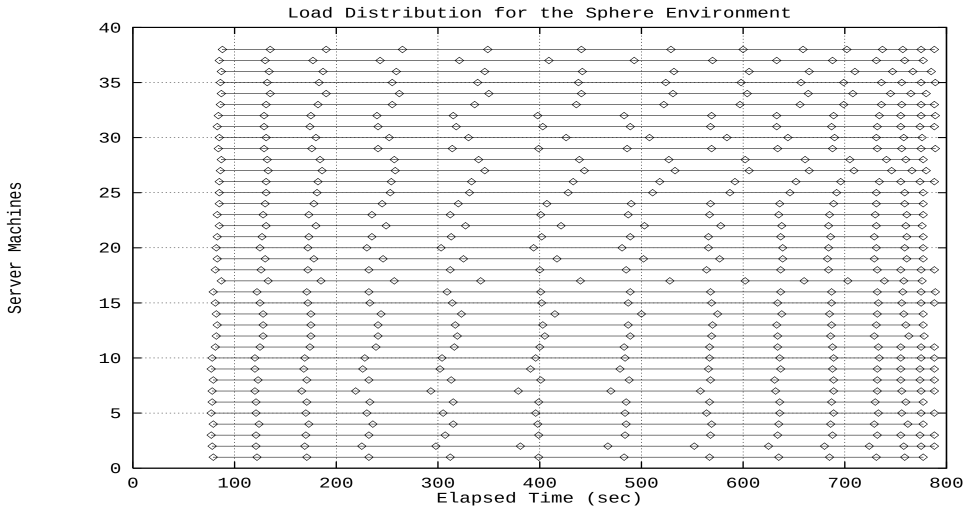

Figure 2: Load Distribution with 1 Line per Task

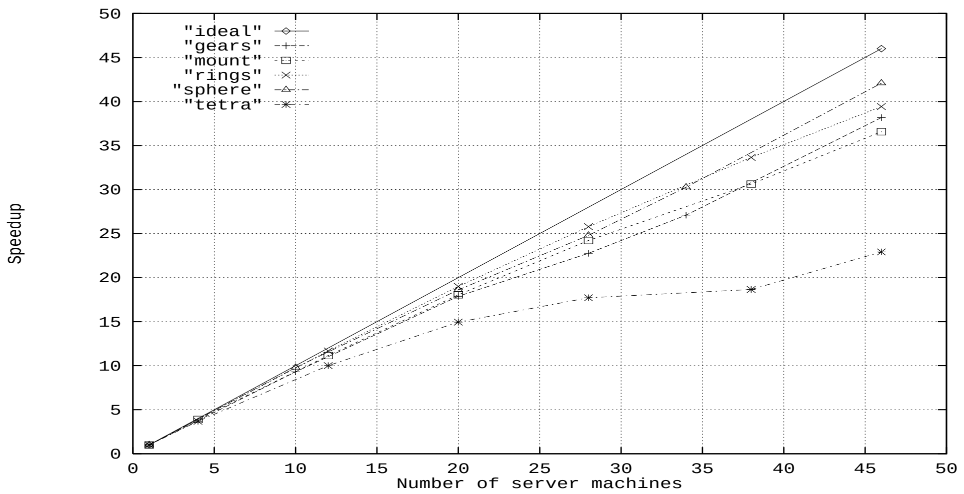

Figure 3: Performance with Homogeneous Servers.

Figure 3 shows the speedup plot of employing up to 46 SSCs as servers. The 45^o straight line depicts the perfect speedup. Except for the case of the Tetra environment, speedup observed ranges from 42.06 for the Sphere environment to 36.56 for the Mount environment. These numbers are very encouraging as they are close to linear speedup.

It is interesting that the maximum speedup achieved for the Tetra environment was only 22.91. The reason for the asymptotic curve is because of the constant time required by each server to initialize the ray acceleration structure. With 46 SSCs, the execution time for the Tetra environment is reduced from 1765.82 seconds (from Table 2) to 77.06 seconds, while the initialization time has remained a constant of 25 seconds. From the Amdahl's law, we know that as the execution time for an environment is reduced, this constant initialization time would become the limiting factor. In this case, even with infinite number of servers, the maximum achievable speedup would be limited to:

$$speedup = \frac{SequentialExecutionTime}{DistributedExecutionTime} = \frac{1765.82}{25} = 70.6\, times.$$

4.4 Speedup in a Heterogeneous Environment

In our department, there are 17 SGI Indigos (3 R4000s, 14 R3000s), 17 Sun ELCs, and 46 Sun SSCs for instructional purposes. It is rather difficult to compare an execution time obtained from this heterogeneous group of workstations with that of the original sequential Rayshade system. From our experience, the SPEC92 benchmark reflects the performance level of a particular machine fairly accurately. For the case of a ray tracer, because the computation involved is floating point intensive, the SPECfp92 benchmark could be used to *normalize* machines for comparison purposes. For example, an SGI Indigo R4000 with SPECfp92 of 60.3 (from Table 1) would be counted as 2.87 Sun SSCs with SPECfp92 of 21 ($\frac{60.3}{21} = 2.87$). In this way, we select a machine type as reference and normalized the rest of the machines. Since the majority of the systems are SSCs, we normalized all the other machines to the SSCs:

$$\text{No. of machine (normalized to SSC)} = \underset{\text{(R4000s)}}{3 \times 2.87} + \underset{\text{(R3000s)}}{14 \times 1.15} + \underset{\text{(ELCs)}}{17 \times 0.85} + \underset{\text{(SSCs)}}{46 \times 1} = 85 \text{ (approx)}$$

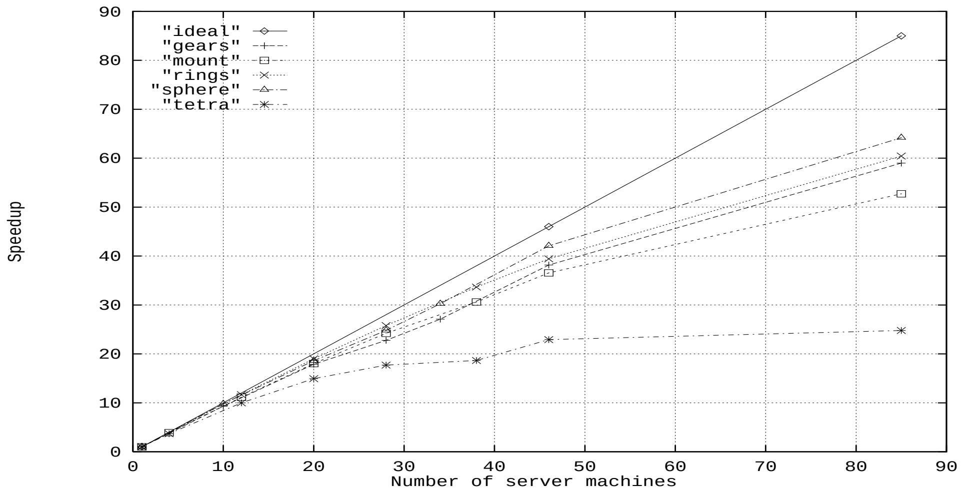

Figure 4: Performance with Heterogeneous Servers.

4.4.1 Constant Task Size

Figure 4 plots the speedup graph of employing all the 85 *normalized* SSCs as servers. In this case, the task size is always one scanline for each server request. Once again, expect for the Tetra environment, the speedup achieved ranges between 52.72 for the Mount environment and 64.19 for the Sphere environment. From the earlier discussion of the Tetra environment, the drop in efficiency is predictable. As more servers are involved the computation time decreases, thus making the constant initialization time more significant, resulting in the decrease in efficiency.

4.4.2 Varying Task Size As Machine Speed

In Section 3.1 we argued that in a heterogeneous environment, it is probably a good idea to scale the task size according to the performance of a target server workstation. In our case, we normalized the task size using the SPECfp92 figures (scanline sizes are round to the nearest integer) for different servers. It is interesting that the achieved performance was almost identical to that of Figure 4.

We use an example to illustrate the situation. Figure 5 shows the load distribution with constant task size (of one scanline for all server machine types) for the Sphere environment. The first 45 machines are the SSCs, followed by 17 ELCs, 14 Indigo R3000s, and finally 3 R4000s. Notice that a almost perfect load balancing is maintained until almost the end of the processing. Remember that the motivation behind varying the task size are to assist load balancing and to minimize communication overhead. In this case, the communication overhead is insignificant and load is well balanced. As a result, performance does not improve with changing task sizes. The four types of machines in our environment do not differ drastically in performances. We believe that the varying task size strategy will work well if servers with very different processor speeds are involved (e.g. 10 times or more).

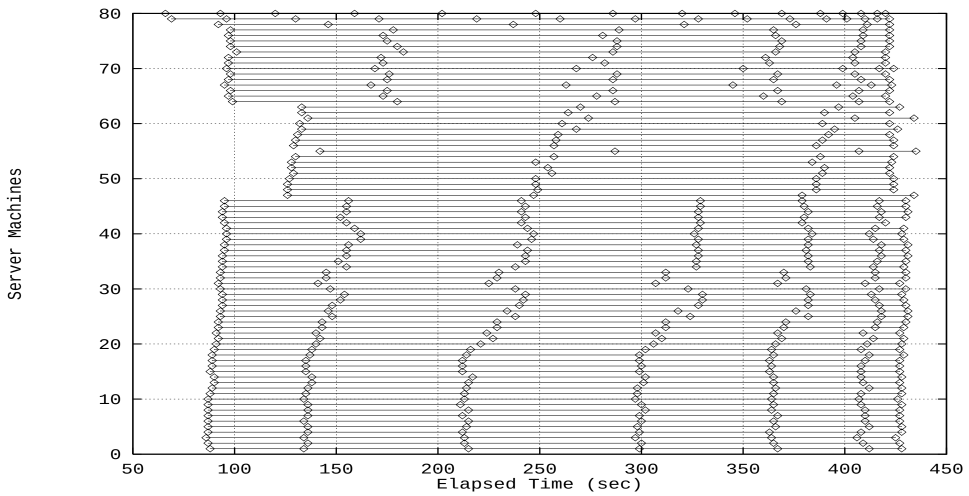

Figure 5: Load Distribution of the Sphere Environment.

4.5 Discussions

The initial design of our distributed Rayshade program was modeled after Muuss's [27] and Bloomer's [4] systems. However, when comparing to the Muuss's work, we have demonstrated that with an efficient underlying communication tool, three scanline task size is not the optimum. Also Muuss's system cannot support more than 64 servers simultaneously, we do not have this limitation. From the results presented in Figure 4, we see that limiting the number of simultaneously supported servers to 64, do restrict the potential achievable speedups.

When comparing to Bloomer's system, we observed that due to the underlying asynchronous communication tool, our system is more efficient. For example, Bloomer reported 10-15 times speedup with 22 servers while our results showed a better performance. We have also demonstrated that with the more efficient communication tool, the run time task size determination strategy is not necessary. With one a constant scanline task size, our system is well balanced for all of the test cases.

4.6 Limitations

Although the speedups observed are encouraging, our approach has some limitations:

- **Image Complexity Distribution.** Because of the different processing speed and image complexity, the servers usually do not complete their task in the same amount of time. If the most complex portion of the image is located near the end then the inter-server completion time (especially between the faster ones and the slower ones) may become the bottleneck of the system performance.
- **System Bottleneck.** Since all the servers build the entire ray acceleration structure, the maximum achievable speedup is limited by this initialization time. For this reason, our system can never achieve *real time* ray tracing [20].
- **Communication Channel.** Although the communication channel has not been a limiting factor in our study, it is important to remember that the channel is very

slow. Because of this inefficiency, our distributed model will not be able to support a large number of servers working in parallel. For example, Cleary et. al. [8] and Scherson et. al. [31] discussed algorithms that supports thousands of processors executing in parallel. Unless the bandwidth of the underlying communication channel is increased drastically, our model will never be able to support such a large number of servers.

5 Conclusion

Initially, we were searching for applications to demonstrate the capabilities and the usability of the ASTRA RPC system. Also, as a secondary goal, we wanted to keep all the departmental workstations busy at night. Ray tracing being *embarrassingly parallelizable*, was one the applications investigated. Somewhat to our surprise and frustration, a literature search of the previous work done did not help us in answering the question that: how many of the workstations should be allocated to this application? We began our investigation by allocating 30 Sun SSCs, more than any of the previous approaches, for our distributed ray tracing system. It soon became apparent that we were maintaining a linear speedup and that more workstations should be employed.

From our experience in parallelizing the Rayshade system, the ASTRA RPC do provide an easy environment for distributed application development. The communication overhead incurred is minimal, and hence we were able to maintain the one scanline task size for better load balancing. The performance testing results showed promising speedups for most of the environments. We have demonstrated that an asynchronous communication tool is essential for efficient distributed ray tracer implementation. Furthermore, because of the potential parallelism involved, the communication tool must not limit the number of machines that can work together. It is interesting that for most of the issues, the *simple* solution approaches provided the best performance results. From our experience, *small* constant task size with database accessing via the NFS gives the best performance in all the cases tested.

One of the most encouraging results of our study is that we have demonstrated that the achievable speedup of parallelizing a classical ray tracer in a typical LAN environment is, in many cases, comparable to the past parallel approaches on expensive dedicated MIMD systems. For example, Badouel and Priol described an algorithm which achieved a 55.71 times speedup when rendering the SPD Mountain environment on a 64-node hypercube system [3]. Of course, the total hardware cost of our networked workstations may exceed that of a 64-node hypercube. However the workstations are for general instructional purposes, and the development of our distributed Rayshade system is straightforward. The limitations of our system is discussed in Section 4.6.

From our experience, the following is a list of guidelines for developing a ray tracer in a distributed environment.

1. It is convenient to *normalize* the different types of servers. In this way, the environment can be considered as homogeneous. From our experience, normalization based on the SPECfp92 provides a good indication for the overall performance of the workstations.

2. To achieve a "good" load balancing, the ray tracing process should be subdivided into T number of tasks. From our experience, T should be about 5 times the number of the *normalized* servers.

3. From the experience with the Tetra environment, high efficiency can be expected only if the the following condition is true:

$$\frac{SequentialExecutionTime}{NumberOfServers} \gg InitializationTime$$

4. If the processing speed of the servers differs drastically, then the size of the task assigned to different servers should be scaled accordingly. From our experience, when the server speed difference is only a few times, maintaining a constant task size does not affect the system performance.

5. As the number of servers increases (e.g. more than 100), it may be a good idea to have more than one client.

Acknowledgments

Thanks to Chan Lee Lee and Koh Eng Kiat for their help in the early phase of the project; to Craig Klob for making the Rayshade system publicly available, and his prompt replies to our queries; to Eric Haines for making the SPD publicly available; to Greg Rogers for the discussion on the SPECmark; and to John McCallum for the SPECmark FAQ.

References

[1] A. L. Ananda, B. H. Tay, and E. K. Koh. Astra - an asynchronious remote procedure call facility. In *Proceedings of the IEEE's 11th International Conference on Distributed Computing Systms*, pages 172–179, 1991.

[2] Didier Badouel, Kadi Bouatouch, and Thierry Priol. Ray tracing on distributed memory parallel computers: strategies for distributing computations and data. *Parallel Algorithms and Architectures for 3D Image Generation*, pages 185–198, 1990. ACM Siggraph '90 Course Notes 28.

[3] Didier Badouel and Thierry Priol. An efficient parallel ray tracing scheme for highly parallel architectures. *The Fifth Eurographics Workshop on Graphics Hardware*, 1990.

[4] John Bloomer. An rpc case study: Networked ray tracing. In John Bloomer, editor, *Power Programming with RPC*, pages 385–427. O'Reilly and Assiciates, Inc., 103A Morris Street, Sebastopol CA 95472, 1992.

[5] K. Bouatouch and T. Priol. Parallel space tracing: An experience on an ipsc hypercube. In *Proceedins of the Computer Graphics International 1988*, pages 170–188, 1988.

[6] Michael B. Carter and Keith A. Teague. The hypercube ray tracer. In *Proceedins of the 5th Distributed Memory Computing Conference Vol. I*, pages 212–216, 1990. IEEE Computer Society Press.

[7] R. Caubet, Y. Duthen, and V. Gaildrat. Voxar: A tridimensional architecture for fast realistic image symthesis. In *Proceedins of the Computer Graphics International 1988*, pages 135–149, 1988.

[8] John G. Cleary, Brain Wyvill, G. M. Birtwistle, and Reddy Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5:3–12, 1986.

[9] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[10] George F. Coulouris and Jean Dollimore. *Distributed Systems - Concenpts and Design*. Addison-Wesley, Readings, MA, 1988.

[11] Franklin C. Crow, Gary Demos, Jim Hardy, John McLaughlin, and Karl Sims. 3d image synthesis on the conenection machine. In *Proceedings of the Conference on Parallel Processing for Computer Vision and Display*, January 1989. Unversity of Leeds, UK.

[12] Danile Evan Defend. Parallel ray tracing in a vector-multiprocessing environment. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, September 1987. Report No. UILU-ENG-87-8005, also CSRD Rpt. No. 677.

[13] Mark A. Z. Dippe and John Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *Computer Graphics*, 18(3):149–158, July 1984. ACM Siggraph '84 Conference Proceedings.

[14] Jeff Erilly and Reinhold Weicker. Answers to the faq about spec benchmarks. *Internet comp.benchmakrs newsgroup FAQ list*, 1994.

[15] fopen(). Bsd 4.3. Technical report, Unix man page, 1994.

[16] Severin Gaudet, Richard Hobson, Pradeep Chilka, and Thomas Calvert. Multiprocessor experiments for high-speed ray tracing. *ACM Transactions on Graphics*, 7(3):151–179, July 1988.

[17] Stuart Green. *Parallel Processing and Computer Graphics*. Mit Press, London, Pitman, Cambridge, 1991.

[18] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987.

[19] M-P Hebert, M.D.J. McNeil, B. Shah, R. L. Grimsdale, and P.F. Lister. Marti – a multiprocessor architecture for ray tracing images. In R. Grimsdale and A. Kaufman, editors, *Advances in Computer Graphics Hardware V*. Springer-Verlag, to be published. Proceedings of the 5th Eurographics Workshop on Graphics Hardware.

[20] Frederik W. Jansen and Alan Chalmers. Realism in real time? *The Fourth Eurographics Workshop on Rendering*, June 1993.

[21] David W. Jensen and Daniel A. Reed. Ray tracing on distributed memory parallel systems. Technical Report UIUCDCS-R-89-1551, Department Of Computer Science, University of Illinois, October 1989.

[22] Craig E. Klob. Rayshade user's guide and reference manual. Technical Report ftp site: princeton.edu, Rayshade.4.0.6.tar.Z, 1994.

[23] P. Krugeer and R. Chawal. The stealth distributed scedular. *Proceedings of the 11th International Conference on Distributed Compting Systems*, pages 336–343, May 1991.

[24] W. Leister, T. Maus, H. Muller, B. Neidecker, and A. Stosser. "occursus cum novo" - computer animation by ray tracing in a network. In N. Magnenat-Thalmann and D. Thalmann, editors, *New Trends in Computer Graphics (Proceedings of CG International '88)*, pages 83–92. Springer-Verlag, 1988.

[25] Sape Mullender. Introduction to distributed systems. In Sape Mullender, editor, *Distributed Systems*, pages 3–18. Addison Wesley, Readings, Massachusetts, 1989.

[26] Michael J. Muuss. Ballistic research laboratory cad package, release 1.21. *BRL Internal Publication*, June 1987. Aberdeen, MD.

[27] Michael J. Muuss. Workstations, networking, distributed graphics and parallel processing. In D. F. Rogers and R. A. Earnshaw, editors, *Computer Graphics Techniques, Theory and Practice,*, pages 409–472. Springer-Verlag, 1990.

[28] P. Pitot, B. Moisan, Y. Duthen, and R. Caubet. A transputer based implementation of the voxar project. In *EUROMICRO '90*, pages 347–354. Springer-Verlag, 1990.

[29] David J. Plunkett and Kichael J. Bailey. The vectorization of a ray-tracing algorithm for improved execution speed. *IEEE Computer Graphics and Applications*, pages 52–60, August 1985.

[30] Thierry Priol and Kadi Bouatouch. Static load balancing for a parallel ray tracing on a mimd hypercube. *The Visual Computer*, 5:109–119, 1989.

[31] Isaac D. Scherson and Elisha Caspary. Multiprocessing for ray tracing: a hierarchical self-balancing approach. *The Visual Computer*, 4:188–196, 1988.

[32] Loh Jen Shiuan. Distributed ray tracing using astra asynchronous rpc. Technical report, Department of Information Systems and Computer Sciences, 1994. Honours Year Project Report.

[33] Gautam B. Singh, Santosh G. Abraham, and Franklin H. Westervelt. Computing radiosity solution on a high performance workstation lan. *IEEE Conference Proceedings with Serial Number 0-8186-2970-3/92*, pages 248–257, 1992.

[34] A. Stober, A. Schmitt, B. Neidecker, W. Muller, T. Maus, and W. Leister. Tools for efficient photo-realistic computer animation. *Eurographics'88 Proceedings*, pages 31–41, 1988.

[35] Kelvin Sung. A dda octree traversal algorithm for ray tracing. In F. H. Post and W. Barth, editors, *Eurographics '91*, pages 73–85. North Holland, September 1991.

[36] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.