The Journal of Supercomputing, 13, 111–132 (1999)

Parallel Computing on an Ethernet Cluster of Workstations: Opportunities and Constraints

MOUNIR HAMDI hamdi@cs.ust.hk

Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

YI PAN

Department of Computer Science, University of Dayton, Dayton, OH 45469-2160, USA

B. HAMIDZADEH

Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, B. C., Canada V6T 1Z4

F. M. LIM

Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Editor: Hamid Arabnia

Abstract. Parallel computing on clusters of workstations is receiving much attention from the research community. Unfortunately, many aspects of parallel computing over this parallel computing engine is not very well understood. Some of these issues include the workstation architectures, the network protocols, the communication-to-computation ratio, the load balancing strategies, and the data partitioning schemes. The aim of this paper is to assess the strengths and limitations of a cluster of workstations by capturing the effects of the above issues. This has been achieved by evaluating the performance of this computing environment in the execution of a parallel ray tracing application through analytical modeling and extensive experimentation. We were successful in illustrating the effect of major factors on the performance and scalability of a cluster of workstations connected by an Ethernet network. Moreover, our analytical model was accurate enough to agree closely with the experimental results. Thus, we feel that such an investigation would be helpful in understanding the strengths and weaknesses of an Ethernet cluster of workstation in the execution of parallel applications.

Keywords: parallel computing, ray tracing, networked workstations, performance evaluation

1. Introduction

Present trends in parallel computer development emphasize expensive technologies and specialized architectural concepts. Currently, we observe a significant increase of workstation performance and communication bandwidth, together with a shift of market interest from mainframes to workstations. Networks of high performance workstations are becoming increasingly available in companies and research institutions [11]. These networks of workstations may be considered as less expensive

virtual parallel computers when used collectively to solve a single computationally intensive task.

To fully exploit and understand the parallel computing power of such a virtual parallel computer, many architectural and system issues have to be addressed and investigated. These issues include the workstation architectures, the network protocols, the communication-to-computation ratio, the task assignment strategies, and the data partitioning schemes. The aim of this paper is to take a step towards studying the strengths and limitations of a network of workstations as a function of the above factors on a computationally intensive application namely ray tracing.

Today's clusters of workstations are connected by various Local Area Networks (LANs). These LANs can be classified into two categories: point-to-point LANs and shared access LANs. In point-to-point LANs, such as Asynchronous Transfer Mode (ATM) LANs, various communication sessions could be taking place at the same time by various groups of workstations. This, in turn, can result in a high network throughput. On the other hand, in shared access LANs, only one communication session could take place at a single instant. The communication protocol characteristic can have a big effect on the performance of the whole network especially when many workstations want to transmit messages at the same time. Shared access LANs can be further classified into two categories: contention-based networks and token passing networks. In Contention-based networks, such as the Ethernet, when more than one workstation desire to access the network, message collision could occur, and retransmission of the messages is thus needed. This message collision is avoided in token-passing networks by proving a circulating network token. Each workstation desiring to access the network must wait until it captures the token. Thereafter, it can transmit its messages. Hence, the choice of a network and its associated communication protocol has a tremendous effect on the performance of a cluster of workstations [18].

The task assignment strategies employed in a cluster of workstations can affect the inter-workstation communication and load balancing of the system. Two major classes have been identified as static and dynamic task assignment strategies. In a static scheme, all tasks are assigned to the workstations once, prior to run time. During run time, the processors execute the assigned tasks. As a result, the overhead due to running these scheduling schemes does not affect the overall performance. They, however, fail to take advantage of valuable information about the system and the tasks that become available at run time. In a dynamic scheme, tasks are assigned to workstations at run time. Such schemes can benefit from on-line information about the status of the system. However, they have to be simple, since their running overhead can directly affect the system performance.

The data partitioning schemes in a cluster of workstations address the issue of how to distribute the data among the local memories of each workstation. A major consideration in data partitioning is the degree to which the data is replicated in each workstation's memory. Complete replication of data among workstation memories reduces inter-workstation communication since each workstation, in this scheme, has an entire copy of the data. This scheme, however, is inefficient in its use of memory. It is also not suitable for applications that frequently read and write into the data. In such applications all copies of the data have to be kept

consistent. Resolving consistency issues may lead to important performance considerations. Reducing the degree of data replication among workstation memories will lead to more efficient use of memory resources. Reducing replication, however, may increase the inter-workstation communication. In such a situation, a task assigned to one workstation may need to request data that reside on other workstations' memories. This can be costly, especially, when employing contention-based networks.

In this paper, we study and model the performance of a ray-tracing application using static and dynamic task assignment schemes on a cluster of workstations connected by an Ethernet network. To keep the model simple and more comprehensive, we assume a replicated data partitioning scheme. A replicated data partitioning scheme is viable, since this application is a read-only application. The model and experimental studies are aimed at demonstrating the strengths and limitations of these system choices for the parallel ray tracing application. As a result, they can shed some light on the suitability of a cluster of workstations in the execution of computationally intensive applications.

This paper is organized as follows. Section 2 gives a brief introduction about the ray tracing technique. Then, in Section 3, we specify our system model and discuss the parallel implementation of the ray tracing process. Section 4 details an analytical performance model of our parallel application on a cluster of workstations that can be used to investigate the scalability of this computing environment. In Section 5 we present extensive experimental results in order to assess various issues that directly affect the performance of a cluster of workstations. Finally, in Section 6, we give some concluding remarks.

2. Ray Tracing Application

In order to experimentally assess the potential of a network of workstations as a parallel computing engine and to be able to analyze the various factors that affect its performance, we have chosen a computationally intensive application from computer graphics. This application concerns the generation of life-like images, and is better known as *ray tracing* [4], [5]. By introducing color, shading and shadow into a picture, it can produce high-quality graphics for accurate 3-D space visualization. However, its large computation requirements make it inefficient on traditional single-processor systems even for generating simple pictures. In this paper, various parallel ray-tracing solutions are investigated and are implemented on a network of workstations connected by an Ethernet network.

When we are given a set of edges and/or surfaces, it is not difficult to create a 2D projection of these onto a screen. Figure 1(a) shows an example of this for a cube. However, to determine those parts of the edges and surfaces which would be visible if the objects were constructed from opaque material is a more complex problem as shown in Figure 1(b) and (c). This has been termed the *hidden* surface problem and techniques which provide solutions to this are known as visible surface algorithms [5]. This kind of algorithms can be classified into two categories: the *image space algorithms* and the *object space algorithms*. The image space

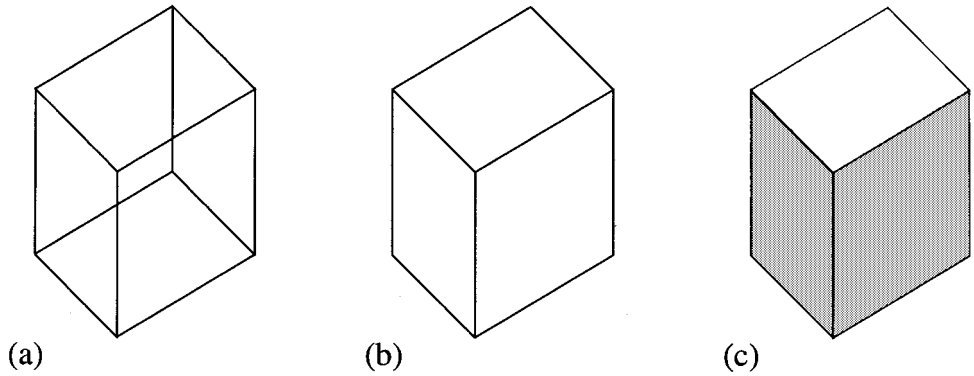

Figure 1. The hidden surface problem in image reconstruction.

algorithms work by first projecting the objects into the plane of the image, and then manipulating those projections. The object space algorithms perform no explicit projection, and each object is considered as a candidate for the visible surface at each pixel on the screen. Ray tracing is an example of this kind of algorithms. It applies the principles of geometric optics to determine how light rays of infinitesimal width interact within the environment [4].

In the ray tracing process, rays are traced from the view point through each pixel of the image screen into the object space as illustrated in Figure 2. As a ray enters the environment, it intersects with objects and only the closest intersecting point corresponds to the visible surface. The color of this point is mapped onto the image screen. This process is performed pixel by pixel and the whole image is thus generated. By extending the algorithm to incorporate reflections, refractions and shadows within a common shading model, high quality realistic pictures can be produced. This extension involves the trace of not only the primary ray, but also a secondary ray called *shadow* ray which is traced from the intersecting point towards each point-light source. If this ray intersects with any object, the original intersecting point is determined to be in shadow with respect to that light source. For reflective and/or refractive surfaces, secondary rays are traced in the direction of reflection and/or refraction. Similar to the case for shadow ray, the interaction of all these rays contribute to the final color of the original pixel.

Ray tracing is very much concerned with the determination of ray-object inter section. In fact, researchers have already found that more than 90% of computation is devoted to this task in most scenes [4]. To determine a single intersecting point may not be too complicated for a traditional serial computer. However, it always happens that the generation of an ordinary scene may involve several thousands of these calculations [3], [9], [14], [17]. To be concrete, let us consider a simple ex ample. The SUN IPX Sparc processor with a clock-speed of 20MHz can calculate a ray-sphere intersection in 466 processor cycles (worst case). For a scene containing 1000 spheres and a resolution requirement of 512×512 pixels, we roughly estimate the average number of rays traced per pixel as 6. This is

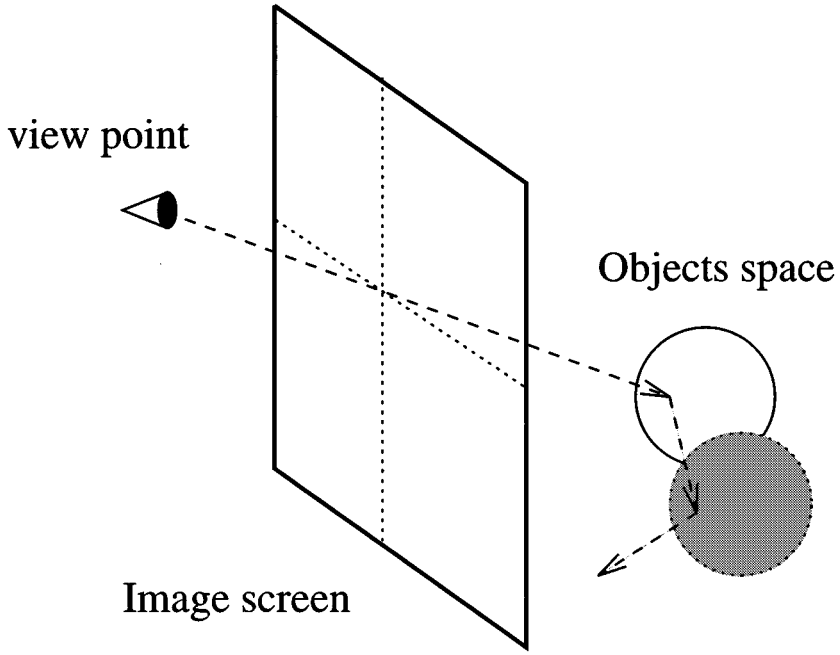

Figure 2. A ray fired into the object space.

contributed by the primary ray, the shadow ray, as well as all other secondary rays due to reflection and refraction. Just considering the cycles required to perform the floating point calculations for the ray-object intersections, the time taken for rendering that image is 512×512 pixels $\times$ 6 rays per pixel $\times$ 1000 spheres per ray $\times$ 466 cycles per sphere $\times$ 50 nsec per cycle = 36000 seconds = 10 hours [4].

This simple example illustrates how computationally intensive the ray tracing process can be. This is one major reason why we have chosen ray tracing for the evaluation of a network of workstations. The second major reason lies in the simplicity of parallelizing the ray tracing process. This simplicity facilitates qualitative and quantitative performance analysis of a cluster of workstations—a major goal of this paper. Next, we briefly discuss different techniques for task division in parallel ray tracing.

There are two main approaches to task subdivision in parallel ray tracing, namely object space subdivision and image space subdivision. Object space subdivision puts the focus on the object space which is being rendered and can be further classified into two categories, namely space subdivision and object subdivision. In space subdivision, the object space is divided into different subspaces, as illustrated in Figure 3. Then, they are assigned to different processors to process the ray tracing. As rays may pass through several subspaces during the tracing, there must be interprocessor communication for such transfer of ray information.

Object subdivision introduces the concept of bounding volume. This is a volume containing a complex object or a group of simple objects. A combination of these

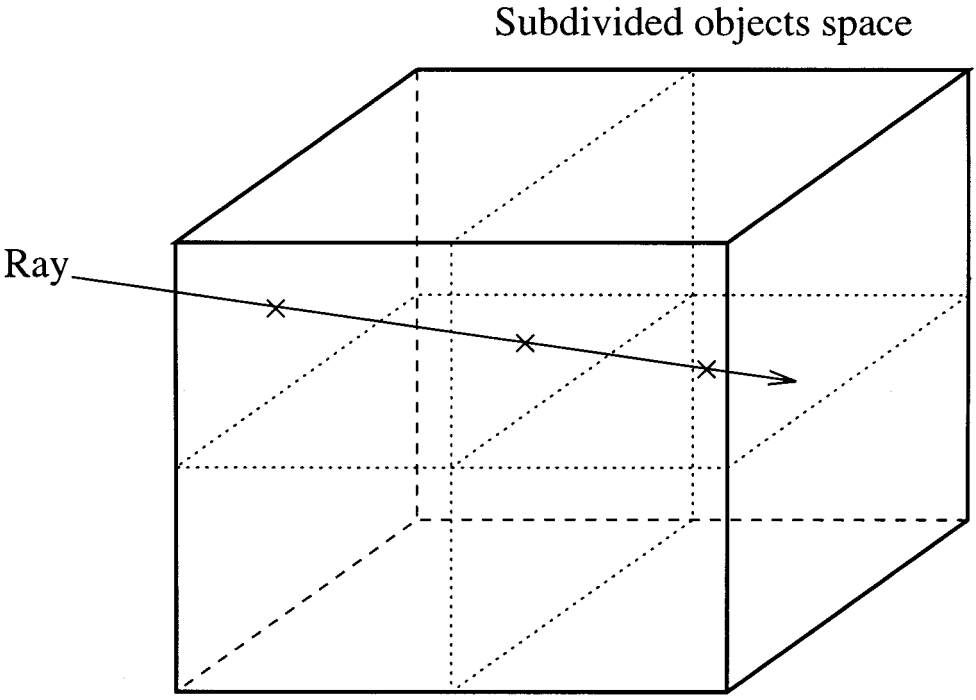

Figure 3. A ray passing through several subdivided spaces.

bounding volumes can form a hierarchy which can facilitate the determination of whether a ray will intersect with the objects that it contains. This avoids useless intersection calculations with complex objects as shown in Figure 4. In fact, this is an acceleration technique commonly applied in most ray tracers [4]. In a parallel processing implementation, distinct groups of bounding volumes are allocated to different processors and each processor is responsible for calculating the ray-object intersection in the particular volume. As a ray will travel through the hierarchy during ray tracing, information flow between processors is necessary.

In image space subdivision, the image plane is divided into several distinct regions, and the processing of a number of regions is allocated to a particular processor. It is just like partitioning the image into several small ones and letting a number of sequential ray tracers trace them separately at the same time. As each pixel on the image plane is independent from one another, and since each workstation stores a copy of the entire image in its local memory, no interprocessor communication is necessary in this approach during the computational process.

Each of the parallelization approaches illustrated above has its own advantages and disadvantages. These advantages and disadvantages are a function of the memory usage, computing/communication efficiency, ease of implementation, and fault-tolerance [2], [10], [12]. For the purposes of this paper, we have chosen the image space subdivision approach to implement the parallel ray tracer. This approach can fully illustrate the weaknesses and strengths of using a cluster of

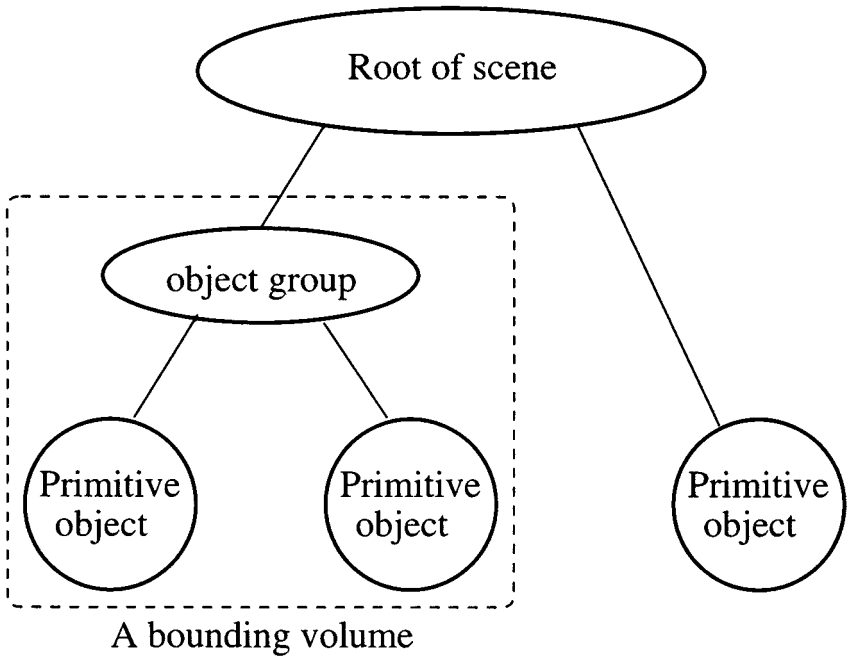

Figure 4. A hierarchical representation of a scene.

workstation as a parallel computing engine. Moreover, as can be easily seen, image space subdivision is better than the other approaches in terms of efficiency, ease of implementation and reliability when implemented on parallel systems.

3. System Model And Implementation

The parallel program for creating ray-traced images reads a text file containing a description of a scene to be rendered and produces a color image corresponding to the description. A simple host/node configuration on an Ethernet network has been adopted as our system model for developing a parallel ray tracing application on a cluster of workstations. According to this model, every workstation node is connected to the host workstation directly. A major characteristic of this model is that the access to the network is *mutually* exclusive. This is a simple and common model for a cluster of workstations. It may not, however, be the best solution in the sense that a bottleneck may be introduced between the nodes and the host. As all the nodes send back large blocks of messages to the host, a traffic jam may occur and the performance may be affected.

The basic architecture of the parallel ray tracer consists of a host program and a set of node programs. The role of the host program is to distribute the tasks among all the allocated processors. It then collects the finished subdivision of the image,

combining them to form the resultant picture. A node program is basically a sequential ray tracer which accepts jobs in the form of horizontal scanlines (a row of consecutive pixels).

To realize the message passing needed between the different processes in an Ethernet network, the EXPRESS parallel computing environment is employed. EXPRESS, developed by Parasoft [13], is a programming environment for writing parallel programs for MIMD multi-computers, including networks of workstations. It is simply a software layer which executes above the individual operating systems of a networked set of autonomous computers. EXPRESS works by setting up *demons* in a group of pre-assigned workstations. When the parallel process is initiated, the *demons* come into effect and handle the required interprocessor communication.

Static and dynamic strategies were implemented to investigate the performance tradeoffs of different task assignment strategies for parallel computing on a net work of workstations. A task in this application consists of processing a ray through a pixel. The static strategy pre-assigns all the tasks to the workstations in one invocation of the task assignment module. This strategy attempts to balance the load by assigning an equal number of tasks to each workstation. Two static assignment schemes, namely tiled subdivision and scattered subdivision were implemented. In tiled subdivision, groups of contiguous scanlines form subsets of the final image. Each node processor is assigned a particular subset. Because the view point and number of objects define the computational complexity for tracing each individual subset, this simple partitioning based on tilings of the screen often creates severe load imbalances. Figure 5 shows an example image generated by this method.

Scattered subdivision is another method of the static schemes which can im prove on the load imbalance introduced in the previous method. The idea is to map scanline i to processor $i \bmod p$, where p is the total number of processors. As neighboring scanlines should have similar complexity for tracing, the load can more or less be scattered evenly among all the processors though there still may exist some difference in the complexity of scanlines. As the node processors involved in the process are time-shared workstations, the computation ability for each machine depends very much on the number of users using it and how demanding the jobs running are. Therefore, all the processors still may not finish the given task at about the same time. Figure 6 shows an example image generated by this method.

The dynamic task assignment strategy that we adopted is a known parallelization strategy and is known as "farming" [1]. In this case, a node is termed as a farmer. Related methods have been proposed and are referred to in the literature as self scheduling [19], [15], [8], [20], [16]. In this scheme, the host node distributes a chunk of tasks to workstations on a "first come first served" basis. A new task will only be assigned to a particular workstation node when that workstation has finished its previous tasks. As a result, those faster workstations will be given more scanlines to process. This dynamic scheme achieves better balance on processor utilization.

The size of each chunk of tasks to be assigned to nodes, every time these nodes become idle, is an important parameter which can affect the overall performance. A small chunk size can benefit from a more balanced load, however, will result in

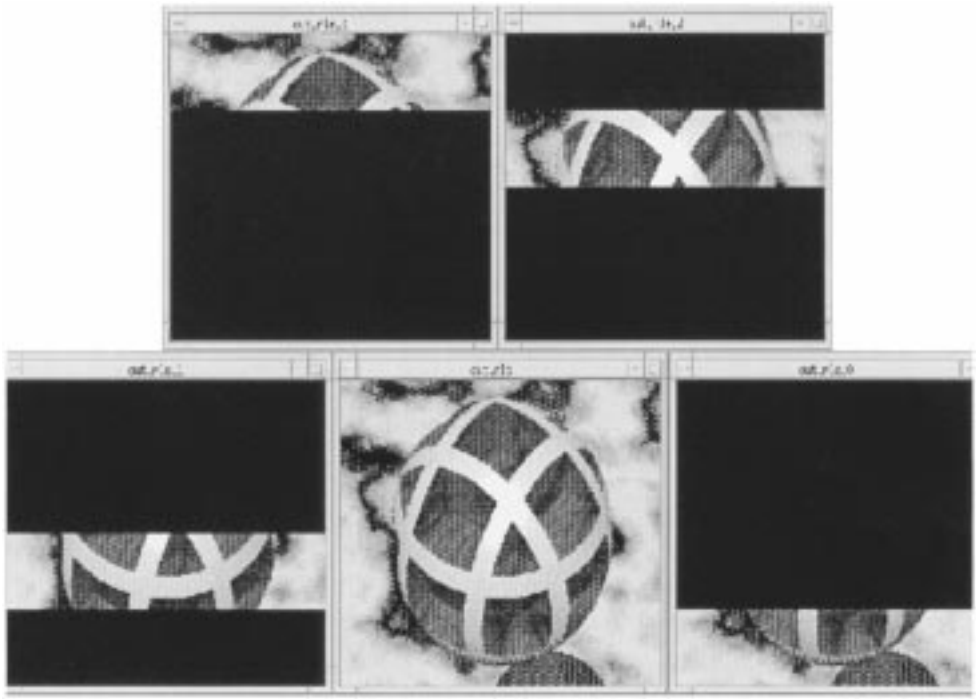

Figure 5. Image generated from tilted subdivision.

higher traffic between the host and the nodes and will result in making the host a synchronization bottleneck, as well. Larger chunk sizes result in a poorly balanced load, but they reduce host-node communication and synchronization overhead at the host. Many different schemes for choosing the appropriate chunk size have been proposed [15], [8], [20]. We selected a chunk size of one which consists of one scan line in our application. The reason for this was to achieve well-balanced loads, so as to make communication costs and network/protocol characteristics the focus of our study.

Figure 7 shows an example image generated by the self-scheduling method. Table I summarizes the comparison of these schemes.

4. Performance Model

Most researchers agree that a cluster of workstations have a good potential for becoming a viable high-performance computing engine. Unfortunately, many aspects of parallel processing via a cluster of workstations are not well understood. Some examples include determining the best computational granularity, decomposition of tasks and data, number and types of workstations, significance of network protocols, and load balancing techniques. In this section, we attempt to partially

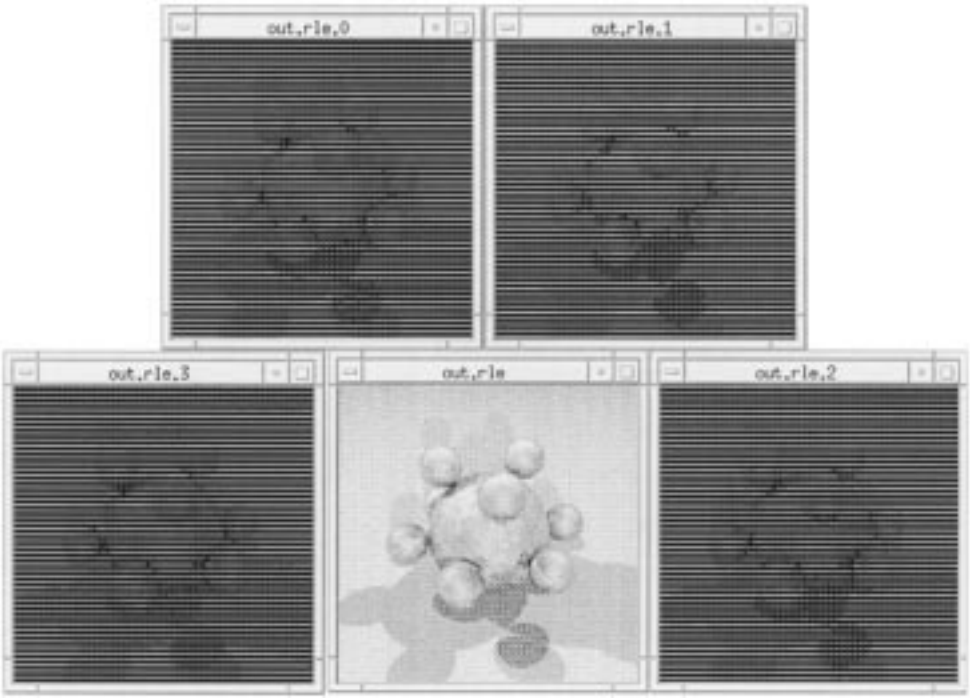

Figure 6. Image generated from scattered subdivision.

answer/address some of these questions and issues by presenting an analytical performance model for a cluster of workstations connected by an Ethernet network and executing a parallel ray tracing application.

One of the most important indicators of the strengths and weaknesses of a parallel computing engine is its scalability with respect to the parallel applications it is executing. Scalability is the study of the relationship between the number of node workstations and the efficiency of the parallel process (i.e., speedup) [6], [7]. As a result, we attempt to understand the scalability of a cluster of workstations connected to an Ethernet network with respect to its execution of the ray tracing process in order to pin-point its strengths and weaknesses. We expect a reasonable degree of speedup to result from running our parallel ray-tracing application on a number of workstations connected via Ethernet. We, however, suspect that this speed up will continue for up to a certain number of workstations and will deteriorate for larger numbers of workstations. Accurately predicting the optimal number of workstations that can perform a parallel job can be beneficial in allocating the resources to do that job. Moreover, it would give us an indication on the "goodness" of an Ethernet cluster of workstations. Consequently, we develop a model for analyzing the scalability of our application on a cluster of workstations connected via an Ethernet network. We validate this model in later sections by

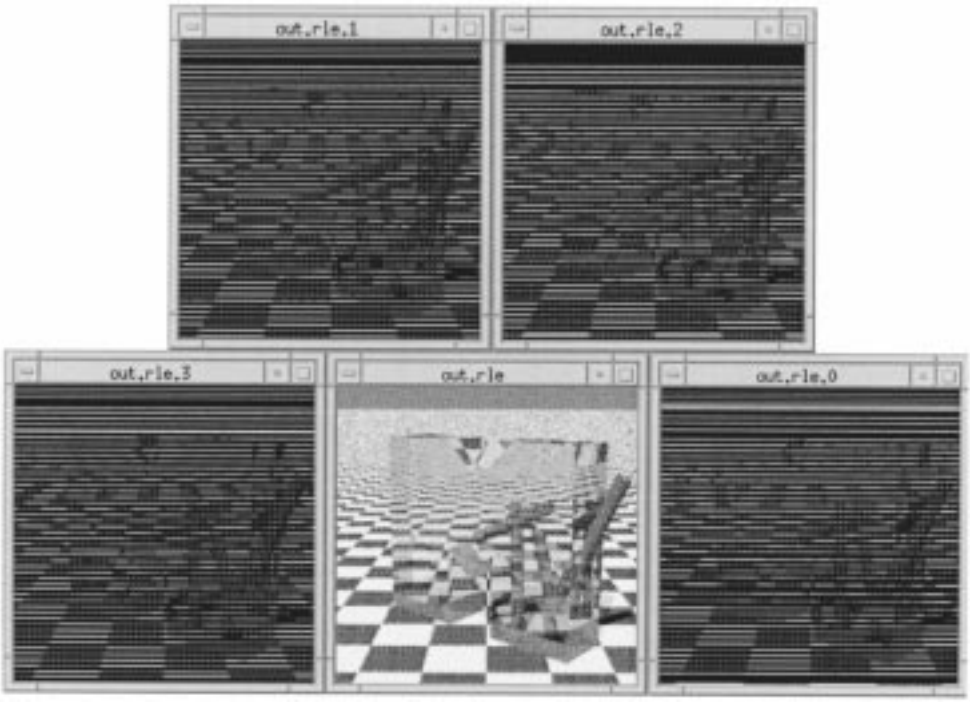

Figure 7. Image generated from dynamic subdivision.

comparing its prediction with actual data that we obtained from running our parallel ray-tracing program on a cluster of workstations.

As mentioned previously, there are numerous factors that directly affect the performance of network of workstations. The inclusion of *all* these factors into a single analytical model is a very complicated task if not an impossible one. As a result, we simply concentrate on the effect of a subset of these factors. The effect of the other factors is one of our future research directions, and will be presented in our future publications. In this present model, we attribute four majors factors

Table I. Comparison of the different ray partitioning schemes.

	Task Partitioning Schemes		
	Static		
	Tiled	Scattered	Dynamic
Implementation difficulty	Trivial	Simple	Moderate
Host-node communication	Low	Low	High
Node-to-node communication	No	No	No
System overhead	None	None	Little
Load balance	Low	Acceptable	High

that affect the scalability of network of workstations, namely *saturation*, *desynchronization*, *protocol* overhead, and *network congestion* [7].

- **Saturation:** means that a problem is split into so many subtasks, that the parallelization overhead, mainly communication, outweighs the performance gained by the parallel computation. This phenomenon can only be influenced by choosing different algorithms or parallelization strategies.
- **Desynchronization:** occurs, if workstations have to wait for data overdue from other machines. This may happen if sudden activities on the network cause a specific machine to reduce the CPU share of the process which is part of the parallel computation. This is a general problem for applications with strict constraint on synchronization.
- **Protocol overhead:** even the best network technologies are not able to avoid protocol overhead which is part of the communication needed for setting up drivers and adjusting flow control parameters.
- **Network congestion:** occurs, if the communication load is close to the throughput of the network. This phenomenon is a physical limitation of the communication medium. As it is well known [18], this is more likely to be a problem with CSMA/CD type networks like the Ethernet than with token-passing techniques. With the random access technique of the Ethernet, message collisions might occur when many stations try to transmit a frame approximately the same time. This is the case during the communication periods of the parallel ray tracer and may lead to limited speedup. Communication technologies in the future, offering up to Gigabits per second bandwidth, might partially solve this problem, whereas increases in workstation performance make this problem more prominent.

In order to get a better understanding of those effects, we developed an analytic model for the parallel ray tracer. For the ray tracing process over an image of n pixels and complexity c, yields a sequential computation time of $t_s = n \times t(c)$ where $t(c)$ is the average time needed for generating a single pixel of an image consisting of c objects. The value of $t(c)$ is directly related to the complexity of the image. For example, $t(c) \approx 0.376$ msec for the 4-balls scene of Figure 8. To calculate the elapsed time for the parallel computation, we integrated the overhead due to communication, idling, and other work which is necessitated by the parallelization of the ray tracer and would not have been performed by the sequential ray tracer. This overhead is dominated by the total communication time, t_{com}, and thus would be integrated into our model. Thus, the elapsed time of the execution of the parallel ray tracer on p workstations, t_p, can be calculated as:

$$t_p = \frac{t_s}{p} + t_{com} \tag{1}$$

The communication time in our parallel ray tracer is mainly due to the transfer of the results from the worker nodes to the host node. For an image of n pixels, $32n$ bits have to be transferred between the worker nodes and the host node. The

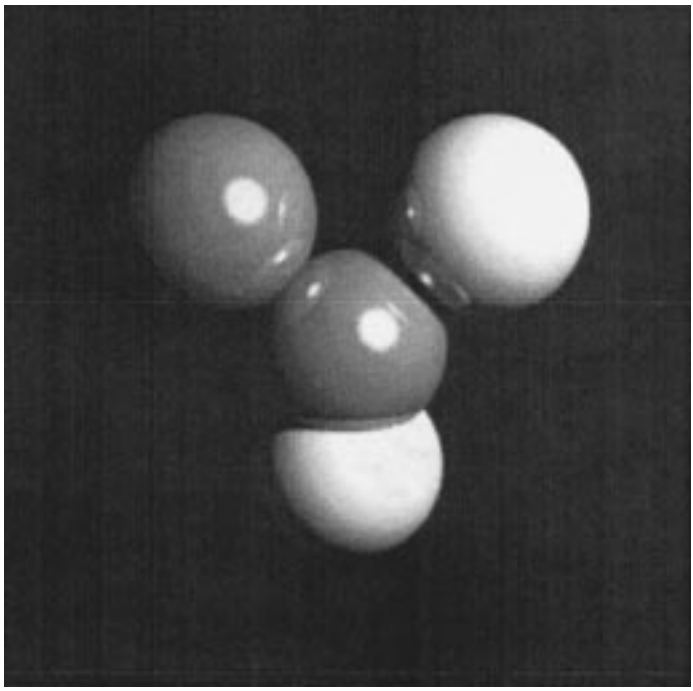

Figure 8. Test image consists of 4 balls with resolution 400 × 400 pixels.

constant 32 is the number of bits needed to store a single pixel. A pixel consists of four attributes: red, green, blue and an alpha channel. Each is of size 8 bits. Hence, a scanline contains $32n^{1/2}$ bits if we assume that the image is square (i.e. $n^{1/2} \times n^{1/2}$). If the image is of size 300 × 300 pixels, a scanline contains 9600 bits. An Ethernet frame contains 18 bytes for the header, and a maximum of 376 image pixels. This is because of the restriction on the maximum frame size in the Ethernet standard which is set to 1534 bytes. The frame length f is the quotient of the number of bits of the frame and the cable transmission rate, which is 10 Mbits/sec in a standard Ethernet. Thus, $f = 1.504$ msec for our 376 image pixels.

The modeling of the Ethernet in [18] offers the following formula for the normalized transfer time, γ, of a single frame, assuming constant frame length f:

$$\gamma = \rho \times \frac{1 + (4e + 2)a \times 5a^2 + 4e(2e - 1)a^2}{2(1 - \rho(1 + (2e + 1)a))} + 1 + 2ea + \frac{a}{2} - \frac{(1 - e^{-2\rho a})\left(\frac{2}{\rho} + 2ae^{-1} - 6a\right)}{2(e^{-\rho(a+1)-1} - 1 + e^{-2\rho a})} \tag{2}$$

where e is the base of the natural logarithm. The variable $a = \tau/f$ depends on the architecture of the Ethernet, where τ denotes the end-to-end propagation delay, which we estimated to be 7.6 μsec in our LAN. The variable $\rho = \lambda f$ denotes the traffic intensity, where λ is the total average traffic in frames per second. Our LAN load monitor suggested the approximation that packet submission is equidistributed during the average communication time t_{com}, because the mutual access to the Ethernet causes desynchronization of the tasks. This will be valid for communication phases with moderate number of collisions. However, for traffic with heavy collisions, the assumption will merely produce an optimistic approximation. During one communication phase, $32n^{1/2}/1516$ frames (i.e., one scanline, 1534 (frame size) − 18 (header) = 1516 (data information)) are transferred. Therefore, we obtain $\lambda = 32n^{1/2}/(1516 \times t_{com})$ and for the traffic intensity

$$\rho(t_{com}) = \frac{32n^{1/2} \times f}{1516 \times t_{com}} \tag{3}$$

Given frames of constant length f (in units of time), $f\gamma$ is the average transfer time of one frame. We therefore get an equation for t_{com} expressing the nonlinearly behavior of the network.

$$t_{com} = \frac{32n^{1/2}}{1516} \times f\gamma(\rho) \tag{4}$$

Substituting Equation (4) into Equation (1), we get the value of the execution of the parallel ray tracer, t_p. Dividing t_s by t_p, we get the speedup/scalability of our model. This analytical model can prove very useful in predicting the speedups. The predicted speedup can be used to project the maximum number of workstations that can be used while preserving a positive speedup. Hence, it can give us a good measure to determine the best computational granularity. As the model predicts, and as we will show in our experiments in later sections, using a large number of workstations can sometimes be counter-productive, that is, the speedup that they achieve is lower than that of a smaller number of workstations. The main reason for this is the low bandwidth of the Ethernet coupled with its collision behavior and the high granularity of certain applications. Hence, this analytical model can save us a lot of time running different experiments to find out the number of workstations that achieves the highest performance for our parallel ray tracer for a given image with a certain size and complexity. Moreover, it gives us a direct measure of the scalability of an Ethernet cluster of workstations.

5. Experimental Evaluation

5.1. Variables to be considered

In order to accurately assess the potential of a cluster of workstations connected by an Ethernet network, we experimented with the effect of various factors such as the computational/communication load, task granularity, data partitioning strate-

gies, and load balancing schemes. All these factors have been embedded into our parallel ray tracing application.

- **Image complexity:** This factor depends on both the size or resolution of the image and the number of objects in the image. If the size is increased, the number of pixels for the picture is larger. As the ray tracing process is to trace rays passing through each pixel, the number of rays to be traced will be larger and the computational complexity is thus increased. When there are more objects in an image, more number of ray-objects intersection has to be calculated. Also, more secondary rays may be generated which introduces more rays to be traced.
- **Amount of data flow between host and nodes:** This factor depends on the size of the image. The data flowing are mainly the informations about the colors of each pixel. As the image size increases, the amount of data will also increase.
- **Number of node processors involved:** This is the number of workers which share the given task. As this number increases, the communication overhead will become larger and the actual computation time becomes smaller.
- **Data partitioning:** This addresses the issue of how to distribute the image data among the local memories of each workstation. In the image space subdivision scheme, we assume that each workstation maintains the entire scene of an image in its local memory.

To provide variation on these factors, test images of different sizes and different number of *balls* are used. Figure 8 shows a typical test image with 4 balls and having a resolution of 400×400 pixels.

5.2. Comparing the three types of image partitioning schemes

To compare the three types of image partitioning schemes, a test image containing 4 balls and with resolution of 400×400 pixels is used. This scene is traced with our parallel ray tracer using different number of SUN IPX workstations connected by an Ethernet network. The graph for the execution time is plotted against the number of nodes (e.g. workstations) in Figure 9. Notice that 0 nodes means that there is no node processor involved in the process. There is only one single serial ray tracer running and no network communication is needed. For the case of 1 node, there is also one single serial ray tracer running but it is running on a remote machine. So, network communication is involved with the host node. Here are some important points which should be noticed from the graph.

- The dynamic subdivision seems to have the best performance no matter how many node processors are involved in the process. It also provides a very stable performance compared with the others. In general, the dynamic subdivision scheme is the best while the scattered subdivision scheme is better than the tiled subdivision scheme. These results completely agree with our expectation (refer to the previous sections discussing the nature of the three methods). As a result, in

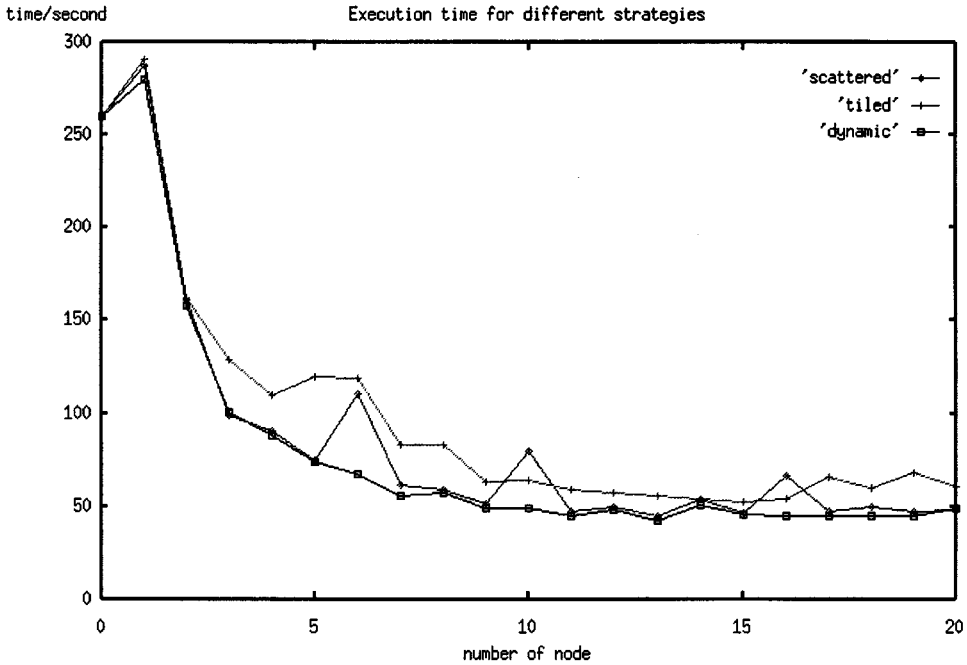

Figure 9. The execution time of the three image partitioning schemes as a function of the number of nodes.

order to take advantage of a cluster of workstations, dynamic load balancing seem to be a must especially given the fluctuation nature of computation and communication of a network of workstations.

- The increase in execution time for only one node processor (having a host and a single node) shows the overhead in sending and receiving information between the host node and the worker node. This is particularly costly for an Ethernet network.
- The execution time generally decreases as the number of nodes increases up to a certain point. This is what parallel processing intended to achieve. However, the curve seems to be not very smooth as the load for the time-sharing workstations do fluctuate quite a lot.
- The change in execution time decreases as the number of nodes increases. There is a saturation point when execution time stops to decrease as more nodes are added. As the number of nodes increases, overhead due to communication and other steps which are not involved in the original serial ray tracer will dominate the process. This saturation point is not that large for an Ethernet cluster of workstations. As a result, our expectations have to be very modest when employing this parallel computing engine.
- Large fluctuation (e.g., when the number of nodes = 6) on the graph occurs when particular node processors are shared by too many user's tasks. This is one of the disadvantages of using a network of workstations as opposed to using

dedicated parallel machines. We can notice that the dynamic subdivision scheme is not affected by this because the heavily loaded workstations will be given fewer jobs to process.

Having determined that the dynamic subdivision scheme is most efficient, later experiments are all performed using this method.

5.3. *Effect of image complexity*

Three test images of size 200×200 pixels are used in this test. The three images contain 3, 6 and 9 balls respectively. This represents different degrees of complexity. Table II lists the execution times in seconds for the different images as a function of the number of workstations employed. Figure 10 shows the variation in speedup (with respect to the serial ray tracer) against the number of node processors for different image complexities. It is easy to notice that the speedup is generally higher for more complicated images. Hence, a higher image complexity will introduce better performance for the parallel ray tracer. As mentioned in the previous section, the increase in efficiency will become saturated when the number of node processors involved reaches some point. From the graph, we can observe that this saturation point seems to depend on the complexity of the image. For the 3-balls image, the speedup begins to fall when the number of processors reaches 5. For the 6-balls image, the graph of speedup seems to flattened when the number of processors reaches 8. For the 9-balls image, the speedup still keeps increasing for all number of nodes. Fortunately, this is what we want since high complexity scenes need all the performance they can get. Moreover, the speedup achieved with a network of workstations is reasonably good especially given the fact that they are interconnected by an Ethernet. Consequently, we can expect a good performance from an Ethernet cluster of workstation mainly when the granularity of the task is coarse *and* the computational complexity of each subtask is high.

5.4. *Effect of image size*

Again, three test images containing 4 balls are used in this experiment. They are of size 100×100, 200×200 and 300×300 pixels. The variation in size not only

Table II. Execution times (secs) for different image complexities.

Number of Nodes	0	1	2	3	4	5	6	7	8	9	10
3 balls											
Execution time	69	89	59	37	32	26	29	32	30	32	34
6 balls											
Execution time	156	154	94	65	47	45	39	36	35	35	33
9 balls											
Execution time	215	238	131	103	65	58	49	47	42	40	39

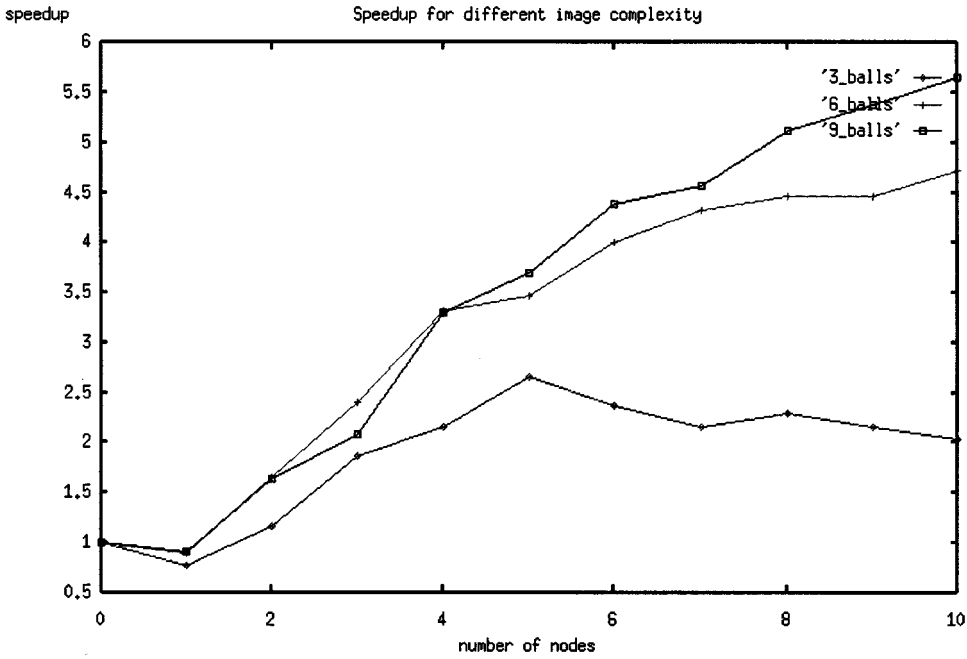

Figure 10. Effect of image complexity on the speedup of the parallel tracer.

affects the complexity, but also the amount of network data transmission (i.e., communication cost). Table III lists the execution times in seconds for different image complexities as a function of the number of workstations employed. Figure 11 shows the variation in speedup (with respect to the serial ray tracer) against the number of node processors for different image sizes. We can have similar observation as that in the previous section: as the image size increases the speedup increases. However, if the amount of data transferred between the host node and worker nodes is too large, the communication time might dominate the whole process. The reason for this result is simple. The execution time of the whole process can be separated into two parts. One is the time spent on actual computation (i.e., the real ray tracing process) and the other is the time spent on

Table II. Execution times (secs) for different image sizes and different number of nodes.

Number of Nodes	0	1	2	3	4	5	6	7	8	9	10
100×100											
Execution time	27	37	26	19	18	15	17	18	18	17	20
200×200											
Execution time	114	119	67	44	38	33	28	30	28	28	30
300×300											
Execution time	223	226	121	87	66	59	50	48	45	41	41

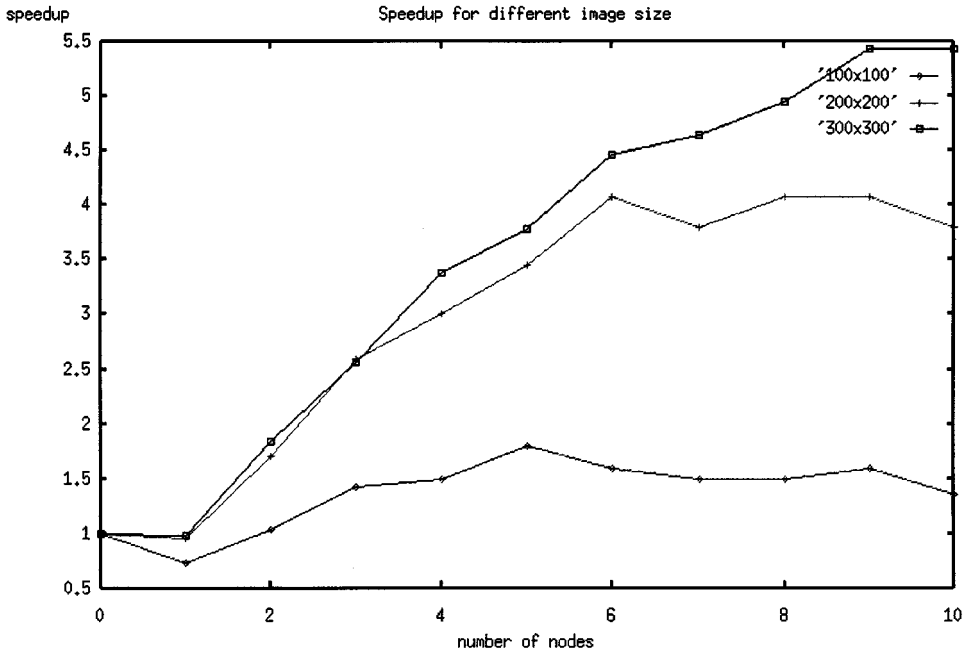

Figure 11. Effect of image size on the speedup of the parallel ray tracer.

communication between the host and nodes. The first part just depends on the complexity of the image. It is controlled by both the number of objects in the scene and the size of the image. However, the other part grows as the number of nodes increases. Each node has to spend some time to set up for the parallel process and to transfer the data back to the host. When the number of nodes gets large, this communication overhead dominates the process and the performance is affected. Thus, we can conclude that the ratio of communication time to the computational time is extremely vital in assessing the suitability of the cluster of workstation in the execution of a parallel application.

5.5. Validation of the Analytical Model

In this subsection, we validate our analytical performance model presented in the previous section with results obtained by experiments. Our model was implemented to study the scalability of an Ethernet cluster of workstations. As mentioned previously, our model does not capture all the potential affects that could be encountered in a cluster of workstations. However, we attempted to emphasize the major effects such as task granularity, computational complexity, communication load, and network protocol characteristics. In this connection, we used our analytical model to predict the speedup that an Ethernet cluster of workstations can give

us when executing a parallel ray tracing process of an image of size 300×300 pixels and 4-balls complexity. Figure 12 illustrates this result and compares it with that obtained experimentally. Thus, we can see that it is quite close to the experimental speedup. The slight difference between the two is due to the fact that our analytical model did not take into account *all* factors affecting the performance of an Ethernet cluster of workstations such as the workstation's heterogeneity and the background load. These effects will be captured in our future work in this direction. Moreover, this analytical model is not only used to verify the experimental results but also to analyze the scalability of an Ethernet cluster of workstations. This scalability can be used to project the maximum number of workstations that can be used while preserving a positive speedup. Using a large number of workstations can be sometimes counter productive, that is, the speedup that they achieve is lower than that of a smaller number of workstations. The main reasons for that are the low bandwidth of the Ethernet coupled with its collision behavior and the high granularity of certain applications. Hence, this analytical model can save us a lot of time running different experiments to find out the number of workstations that achieves the highest performance for our parallel ray tracer for a given image with a certain size and complexity.

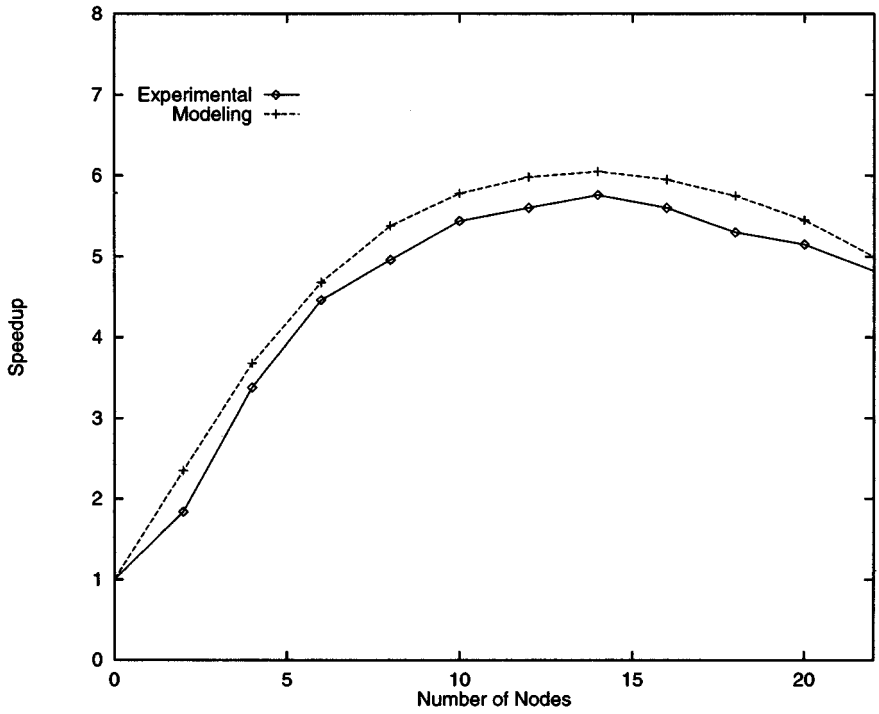

Figure 12. Scalability model of our parallel ray tracer for an image size of 300×300 pixels and 4-balls complexity.

6. Conclusion

In this paper, we addressed the issue of evaluating the performance of an Ethernet cluster of workstations in the execution of computationally intensive applications. There are many factors that directly affect the performance and scalability of this computing engine. Some of these factors are: the workstation architectures, the network protocols, the communication-to-computation ratio, the load balancing strategies, and the data partitioning schemes. We evaluated the performance of this computing environment in the execution of a parallel ray tracing application through analytical modeling and extensive experimentation. We also demonstrated the strengths and weaknesses of an Ethernet cluster of workstations with regards to the above factors. While all these factors have a direct effect on the scalability of an Ethernet network of workstations, it is concluded that the main limiting factor of this computing environment is the contention characteristic of the CSMA/CD medium access control (MAC) protocol of the Ethernet network.

Acknowledgments

This work is supported in part by the Hong Kong Research Grant Council under the grant RGC/HKUST 100/92E.

References

1. H. R. Arabnia and S. M. Bhandarkar, "Parallel stereo-correlation on a Reconfigurable Multi-Ring Network," *The Journal of Supercomputing*, Vol. 10, No. 3, pp. 243–270, 1996.
2. D. Badouel and T. Priol, "Ray tracing on distributed memory parallel computers: Strategies for distributing computations and data," In *ACM SIGGRAPH '90*, Course Note no. 28, pp. 185–198, 1990.
3. H. C. Delany, "Ray tracing on a Connection Machine," In *Int. Conf. on Supercomputing*, pp. 272–278, 1988.
4. A. S. Glassner, *An Introduction to ray tracing*. Academic Press, 1989.
5. S. Green, *Parallel Processing for Computer Graphics*, Addison Wesley, 1991.
6. A. Gupta and V. Kumar, "The scalability of FFT on parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 8, August 1993.
7. M. Hamdi and C. K. Lee, "Parallel image processing on a network of workstations computing environment," *Parallel Computing*, Vol. 21., pp. 137–160, Jan. 1995.
8. S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A practical and robust method for scheduling parallel loops," In *Proc. Supercomputing Conference*, pp. 610–619, 1991.
9. T. Y. Lee, C. S. Raghavendra, J. B. Nicholas, "A fully distributed parallel ray tracings Scheme on the Delta Touchstone machine," In *Proc. 2nd Int. Symp. High Performance Distributed Computing*, pp. 129–134, 1993.
10. T. Y. Lin and M. Slater, "Stochastic ray tracing using SIMD processor arrays," *The Visual Computer*, Vol. 7, pp. 187–199, 1991.
11. P. Krueger and R. Chawla, "The stealth distributed scheduler," in *Proc. 11th Int. Conf. Distributed Computing Systems*, pp. 336–343, 1991.
12. C. Montani, R. Perego and R. Scopigno, "Parallel rendering of volumetric data set on distributed–memory architectures," *Concurrency: Practice and Experience*, Vol. 5, pp. 153–167, 1993.

13. Parasoft Corporation. *An overview of the Express system*, 1992.
14. D. J. Plunkett and M. J. Bailey, "The vectorization of a ray tracing algorithm for improved execution speed," *IEEE Comput. Graph. Appl.*, Vol. 8, pp. 52–60, 1985.
15. C. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: A Practical Self-Scheduling Scheme for Parallel Supercomputers," *IEEE Transactions on Computers*, Vol. C-36, pp. 1425–1439, December 1987.
16. T. Priol and K. Bouatouch, "Static load balancing for a parallel ray tracing on a MIMD hypercube," *The Visual Computer*, Vol. 5, pp. 109–119, 1989.
17. J. Salmon and J. Goldsmith, "A hypercube ray tracer," In *Proc. Third Conf. on Hypercube Concurrent Computers and Applications*, pp. 1194–1206, 1988.
18. M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison Wesley: Reading, MA, 1987.
19. P. Tang and P. C. Yew, "Processor self-scheduling for multiple-nested parallel loops," In *Proc. 1986 International Conference on Parallel Processing*, pp. 528–535, August 1986.
20. T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 87–98, January 1993.