Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Comparisons of Load Balancing Strategies for Ray Tracing on Network Clustered Environments

Chungnan Lee, Tong-yee Lee*, and Tain-chi Lu

Institute of Computer and Information Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, ROC
email:cnlee@mail.nsysu.edu.tw
tclu@cie.nsysu.edu.tw
*Department of Information Management
Nantai Institute of Technology
Tainan, Taiwan, ROC
email: email:tlee@candy.ntc.edu.tw

Abstract

Load balancing has been a major issue in network computing for ray tracing and other applications. In this paper we present a new load balancing strategy, called global distributed control to load balancing in the network processors. The algorithm not only has the ability to dynamically adjust to the load, but also has the fault tolerance ability. In order to study the performance of the algorithm we design a common experimental protocol to evaluate the performance. The parameters used in the protocol include environment configuration, problem size, data type, number of workstations, and algorithm selection. We compare the algorithm with the traditional load balancing schemes such as a master slave, an interleaved, and a proposed hybrid algorithms. Experimental results show that the GDC performs the best among four algorithms.

1. Introduction

Ray tracing has produced the most impressive photorealism image in computer graphics which is widely applied to many areas, such as scientific visualization, computer animation, dynamic simulation, and virtual reality. A simple ray tracing model is shown in Figure 1. The color and intensity of a pixel in the image plane is obtained by tracing a ray from the view point backward into the scene. Then to find if there is a direct or an indirect path from object to the light source. In the past, research has focused on minimizing the cost or reducing the overall number of intersection calculations. Space partitioning structures, bound volume, shadow buffers and ray coherence techniques have all been proposed to reduce this cost. Many techniques have exploited different data structures to speed up the search for a closest intersection on a ray. These structures (such as Octree, BSP trees, K-D trees, and hierarchical bounding volume tree), allow us to search small percentage of the scene to determine the closest intersection. The survey of ray tracing acceleration techniques can be found in Arvo and Kirk's seminal article [1]. In our raytracer implementation, we adopt the hierarchical bounding volume tree structure proposed by Kay et al. [2] to speed up raytracing calculations.

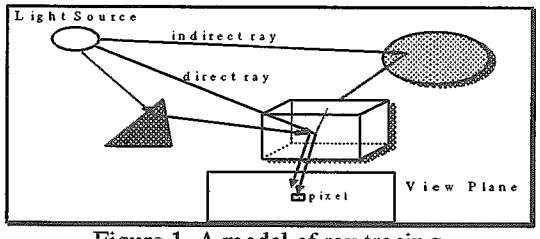


Figure 1. A model of ray tracing

In this paper, we present a new dynamic load balancing scheme, called global distributed control and compare it to a couple of traditional load balance schemes, such as the mater slave and the interleaved schemes. In order to evaluate the performance of these algorithm we design an experimental protocol to describe what performance can be characterized for performance evaluation of algorithms. The paper is organized in the following way. We first review a couple of traditional load balancing scheme and propose a hybrid algorithm in Section 2. Section 3 presents the global distributed control. In Section 4 we design an experimental protocol to carry out the performance evaluation of the algorithms. Section 5 presents the results and discussions. In the last section we give the conclusions.

2. Review of Typical Load Balancing Schemes

As mentioned in preceding section, in addition to inventing new accelerating techniques of ray tracing on sequential machines, a large number of researchers devote to parallelizing ray tracing on many experimental and commercial multiprocessor machines [8-11, 17, 18]. In this section, we review two schemes and propose a hybrid algorithm to parallel ray tracing on networked clusters of workstations.

The first one used is a static load balancing scheme called ``interleaved '' assignment. The idea is to assign scanline *I* to workstation *I mod N*, where *N* is the total number of workstations. As neighboring scanlines should have similar computational complexity for ray tracing, the computational load can be more or less scattered evenly among all the workstations. The communication mechanism for this ``interleaved '' scheme is described in Figure 2.

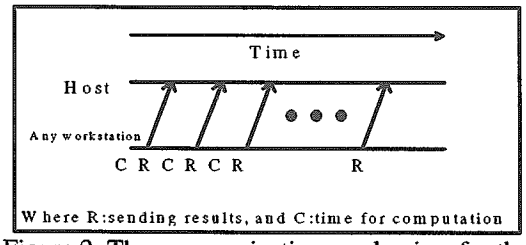


Figure 2. The communication mechanism for the interleaved scheme

In Figure 2, a host workstation is responsible to collect scanlines rendered at different workstations and display the image. As each scanline is finished, the rendered results are sent to the host by a non-blocking sending. Such overlapping between sending the previous results and computing the current scanline can hide communication cost and reduce the network contention to some extent.

The second one used is a dynamic load balancing scheme called ``master-slave '' approach, which is a popular strategy in network computing [1], uses a single master for task scheduling, results collections and image display, and uses multiple slaves to perform real computation. The communication mechanism for this ``master-slave '' scheme is described in Figure 3.

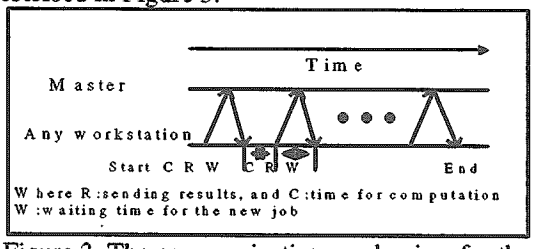


Figure 3. The communication mechanism for the master-slave scheme

In Figure 3, the slave requests a job(i.e., a scanline) from the master to compute. As the slave finishes the current scanline, it will send the rendered results back to the master and wait for the arrival of new job from the master. The master-slave scheme considers how the computing power of workstation is changing as the process is running. This scheme just distributes the task on a FCFS basis. As a result, those faster workstation will be given more jobs (scanlines) to compute (i.e., achieve better balancing on workstation utilization). As all workstations send back large blocks of messages to the master, a traffic jam may occur and thus effects the performance. In such situation, the waiting time (W) (i.e., the propagation time of both the sending results and the new job) will be lengthened. Additionally, the computing power of slave is wasted in the period of W in the master/slave configuration.

To take advantages of both ``interleaved'' and ``master-slave'' schemes we propose a hybrid scheme. There are two phases in our hybrid scheme where the first phase attempts to reduce the waiting time (W) in the master-slave scheme by a modified interleaved scheme, and the second phase attempts to dynamically adjust load imbalances incurred in simple ``interleaved'' scheme. Initially, the consecutive scanlines of image are cut into two parts K1 (0..C-1) and K2 (C.. L-1) at scanline, C, where L is the number of scanlines. In the first phase, done by the master, the modified ``interleaved'' scheme takes the current load of each workstation into account before assigning K1 among the workstations in an interleaved fashion. For example, there are three slave workstations in use and their relative speeds at initial stage are: w1=1,w2=2, w3=1. We will logically consider this configuration as: w1=1, w21=1,w3=1,w22=1, where w2 is logically considered as two different workstations , (w21, w22), with the same speed. Then, K1 part (0.. C-1) is logically assigned among these four workstations (w1, w21, w3, w22) in an interleaved fashion. After this assignment, the first phase performs the same operations as the original ``interleaved'' scheme does. In the second phase, the master will be responsible to dynamically schedule K2 part among the slaves. As time goes by, the remaining scanlines in each workstation becomes less, and after a threshold point (i.e., entering the second phase), an extra workload request, contained in the resulting message (i.e., scanline plus load request), is sent to the master. Under this arrangement, the slave can be processing the remaining scanlines while the new job (one of K2) is sent from the master simultaneously. In our implementation, the C is set to 400 and the threshold is set to 1 (i.e., one scanline is left). The communication mechanism for this ``hybrid '' scheme is described in Figure 4.

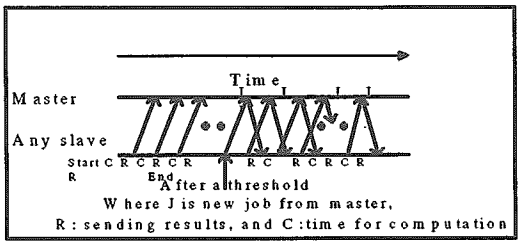


Figure 4. The communication mechanism for the hybrid scheme

3. Global Distributed Control

In this section, we present a load balancing method called "GDC" (Global Distributed Control). There are two phases in the GDC scheme. In the first phase, similar to the interleaved scheme, it uses the equal partition to assign scanline I to N workstations by I mod N. The master is responsible to collect the results which are sent from slaves and display the image. But the master process does not attempt to dynamically adjust imbalance load. The communication mechanism for this "GDC" scheme in the first phase is similar to the interleaved that is shown in Figure 2.

When a slave finishes the current scanline, it will send the rendered results back to the master without waiting for the arrival of the new job from the master. Because the computing power of workstations may be different and the load may be imbalance, the slaves in the high computing power or low load workstation will finish their job faster than those in the low computing power or heavy load workstation. In such situation, the GDC scheme will be able to dynamically adjust the load balancing by the second phase procedure. The second phase of GDC is as follows.

1. N processors are connected by using a ring structure as shown in Figure 5.

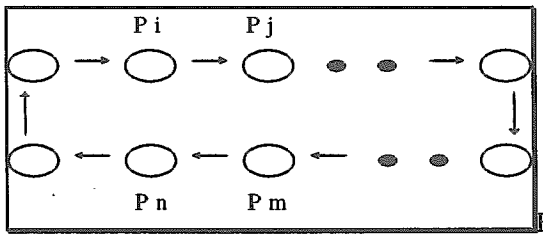


Figure 5. The ring structure of N processors

2. During rendering, when a processor P_i becomes idle, it sends a message to its next processor P_j to unfinished request scanlines. However, there are several cases to be considered.

   Case 1 : If P_i is not idle, P_j will send an extra task to P_i.

   Case 2 : If the processor P_j has finished its pre-assigned tasks (i.e., in the first phase), then P_i will use P_{i-1}, P_{i-2}, P_{i-3}, ..., P_k sequence to search an unvisited processor that is not idle. If P_i finds an unvisited process P_m that is not idle, P_i process will keep its integer task identifier. If P_i wants to request a new job, P_i will skip the processes between P_i and P_m and directly ask a new job from P_m.

   Case 3 : If P_i 's next processor P_j was idle, but P_j has found a processor P_m that is not idle. Hence, the processor P_i sends a message to the processor P_j to ask a new job, P_j will acknowledge P_i that P_i can ask a new job from P_m instead. Thus, the processor P_i does not have to waste time to repeat the same process.

3. If P_i has visited itself, then the processor P_i will terminate.

GDC method is a kind of decentralized parallel computation environment. In the traditional method "master-slave", the master process is the bottleneck and each slave must wait for the arrival of new job from the master. However, the master in GDC scheme does not dynamically adjust load balancing. Each slave must do real computation and control its own load balancing. The communication mechanism is shown in Figure 6.

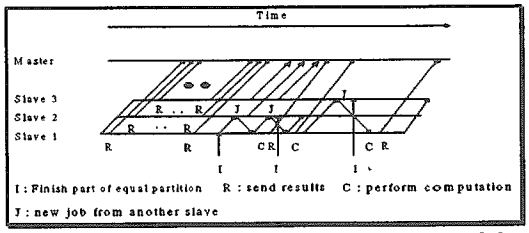


Figure 6. The communication mechanism of the GDC scheme

Network in PVM and fault tolerance

Network computing may consist of heterogeneous environment. In particular, PVM communication is based on TCP, UDP, and Unix-domain sockets. There are three connections to consider : pvmd-pvmd, pvmd and task, and tasks-tasks. In pvmd-pvmd mode, PVM daemons communicate with

another pvmd through UDP sockets. As we know, UDP is an unreliable delivery service which can lose, duplicate or reorder packets. However, it is a waste to lose a message in a virtual machine especially in GDC's environment. Because ring's construction is so weak that any lost message may incur a deadlock. Thus, it is important to incorporate the fault tolerance into the GDC's environment.

The process of fault tolerance is described as follows.

1. When a processor P_i finishes its job and is idle. Then P_i will visit P_{i+1} to ask for a new scanline to render. There are two cases to consider. One is when P_{i+1} is busy that can not response to the request of P_i. The other is when P_{i+1} is crashed.
2. In order to distinguish whether the processor P_{i+1} is crashed or busy, we must set the timeout. If P_{i+1} does not response the request more than three times, we can regard that P_{i+1} is crashed.
3. If P_{i+1} is crashed, P_i will send a message to the master and ask master for checking the global scanline table. Then master will notify P_i to take over the incomplete tasks that were owned by the process P_{i+1}. The information includes how many scanlines P_{i+1} has not finished and the processor identifier of P_{i+1}'s next processor. At present, P_i can replace P_{i+1} to connect with the P_{i+1}'s next processor. So it looks like a satellite to protect the planet when the planet cannot normally operate.

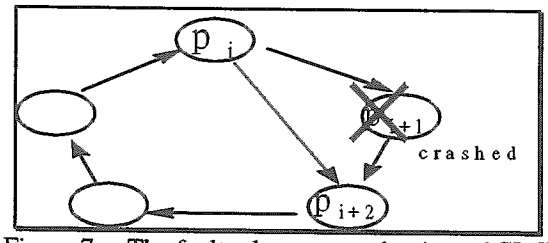


Figure 7. The fault tolerance mechanism of GDC

For this fault tolerance scheme, the crashed process can be successfully replaced by the preceding processor without paying too much overhead. The other processors can continue their jobs without spending the extra time to modify their environment's parameters.

4. Experimental Protocol

In order to evaluate the performance and understand the behavior of many applications on network computing environment systems, it is necessary to establish a common experimental protocol. In this protocol it should describe what is the system configuration, what application is run, what kinds of data types and volume are used, what parameters to be used, how to measure the performance, how change in parameters affecting the performance of the application, and how to make improvement. The purpose of this section is to design and apply the experimental to evaluate the performance of different algorithms on the network computing environments. In particular, it provides an explicit protocol to be followed in the general application.

4.1 Protocol

The performance evaluation of network computing and its applications for some parameters and algorithms have been scatterly reported in the literature [4-16, 20-24]. In [11], Cap and Strumpen presented the method of heterogeneous partitioning and compared it with the method of homogeneous partitioning. Strumpen and Casavant [9] considered the contention for LAN-based networks and latency for WAN-based networks. In this section, the parameters used in our protocol evaluate the performance of four load balancing schemes are described as follows.

1. Environment configuration
2. Problem size
3. Data type
4. Number of workstations
5. Algorithm selection

The main performance factor to be measured is speedup. The speedup is defined as the sequential execution time divided by the parallel execution time. The speedup currently is measured in the wall-clock time. We discuss each parameter in the following paragraphs.

Environment configuration

Network computing may consist of *homogeneous* or *heterogeneous* environment. For homogeneous environment where each computing node is of the same machine type, the measurement of some performance such as speedup tells what speed can be gained or saturated, if more workstations are added to the computing environment. However, for the heterogeneous environment where the computing power of each machine might vary significantly, it is hard to tell what workstation contributes how much computation to the speedup. In order to get a uniform measurement of performance for speed-up comparison, the computing power of each machine type should be measured first.

Parallel Computing Platform

A number of programming environments exist that

make distributed computing available to the application programmer. Among them are PVM, P4, Parform, Express, ...etc. Here, the public domain PVM is used as the parallel computing platform.

Problem Size

The system performance for a particular application will change for different problem size. The ideal case for the speedup versus the problem size is expected to be a linear relationship. Due to communication overhead and others factors, the relation may not be linear at both ends of the curve.

Data Type

While an unsophisticated parallel schemes give good performance on a scene with even distribution, it may give a poor performance when load distribution is highly non-uniform. For example, in graphics render problem, the performance of algorithms may have different behavior under different image data type, thus it needs to be taken into consideration.

Algorithm Selection and Implementation

There may exist several algorithms for each application problem. Some algorithms are inherently sequential and not parallelizable. Furthermore, some algorithms may have efficient speedup in dedicated multiprocessor system, such as Intel Hypercube, but cannot have the same performance on workstation clusters. There are several issues to consider during the implementation of the algorithms:

1. data partition: static, dynamic or fine-grained partition in order to reduce the communication overhead.
2. communication model: master-slave or single-program-multiple-data (SPMD) model depending on the selected algorithm.
3. load balancing: static or dynamic load balancing for purpose of distributing evenly the tasks onto the multi-user machines with different load.

Number of Workstations

The speedup usually increases in accordance with the number of workstations. It is an important parameter to show how much speedup can be improved by adding processors. But the speedup is not necessarily increasing without limitation by adding more processors as indicated in [8]. In heterogeneous environment, the workstation number is replaced by normalized computing node number by measuring the relative computing power of each composing node.

4.2 Experiments

Currently, we concentrate on some of many parameters proposed in the experimental protocol to illustrate how the performance characterization can be done. For each experiment the fifty data measurements are taken. The outliers are removed, before the mean and variance is calculated. The parameters will be set based on the following.

1. Environments configuration
   The workstations used for the experiments includes twelve HP 715/33, one Sun Sparc 2, and one Sun Sparc 20. The computing power for each type of workstation is measured by the data to be used in the experiment. The results are listed in Table 1.

Table 1. The relative power of different workstations

Host	Max. execution time	Min. execution time	Mean	Relative power
HP 715/33	101 secs	96 secs	97.86 secs	1
Sparc 2	225 secs	218 secs	221.42 secs	0.442
Sparc 10	110 secs	103 secs	105.3 secs	0.929
Sparc 20	96 secs	71 secs	72.65 secs	1.347

We use the same program of sequential version to test under the different workstations. And using the same image data as the testing input file. Thus, we define relative power of HP as one and compare it with the other workstation's mean execution time. As one can expect, Sparc 20 got the highest relative power and Sparc 2 the lowest relative power.

2. Data type and problem size
   To perform our experiments, we used a set of standard scenes from Eric Haines's database, called SPD[15], which can be found in the public domain. The geometric characteristics of three test scenes- gears, balls, and tetrahedral pyramid that are shown in Figure 8. The complexity of these test scene data is shown in Table 2. However, we find that the objects in these scenes are rather uniformly distributed, which tends to minimize potential load balancing problem. Hence, we adjust the view points of a test scene to produce more biased object distributions as shown in Figure 9. This change will result in different variance of pixel computations over this test scene. The large variance suggests that one scene should be more difficult for which to achieve good load balancing. Therefore, to evaluate the soundness of a load balancing scheme, we control the view points to allow different variance of pixel computations. Under the same testing environment, we define the time complexity of

balls scene as one. Therefore, when we use gears to be the testing data, it needs more time to get the rendering result.

Table 3. The complexity of three test scene image

	gears	balls	tetrahedral pyramid
Time Complexity	11.15	1	0.455
Size of Structure	1152 rectangles	91 spheres	16 triangles

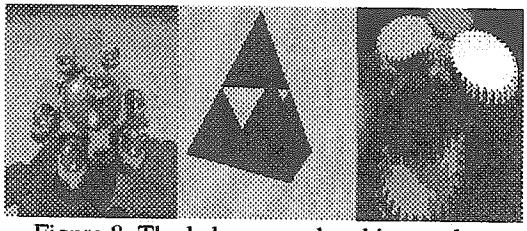


Figure 8. The balance rendered image data

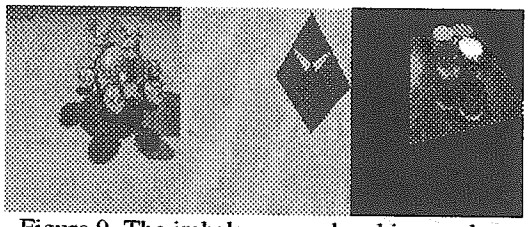
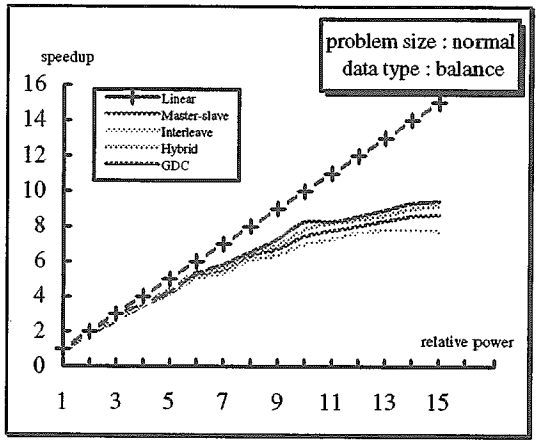


Figure 9. The imbalance rendered image data

3. The number of slaves
Since the UNIX based workstation uses time sharing for it tasks. The more slaves are running in the system, the larger portion of CPU resources is allocated to the slaves. We would to like to know how the number of slaves affects the speedup in one workstation.
4. No. of workstations: up to fourteen workstations. The HP 715/33 workstation is added one by one to twelve. Then, one Sun sparc 2 is added. Finally, one Sun sparc20 is added to make it fourteen.

4.3 Results and Discussions

Some results are shown in Figures 10-12. We calculate the speedup and compare it with the linear speedup. The heavy and balance testing scene can achieve the high speedup because it's easy to achieve the load balancing. But the imbalance and light testing scene got the worst speedup.



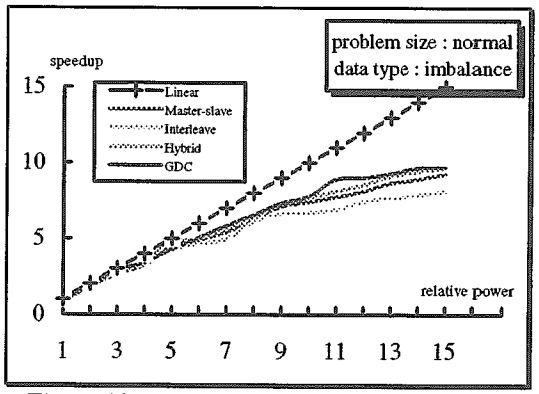
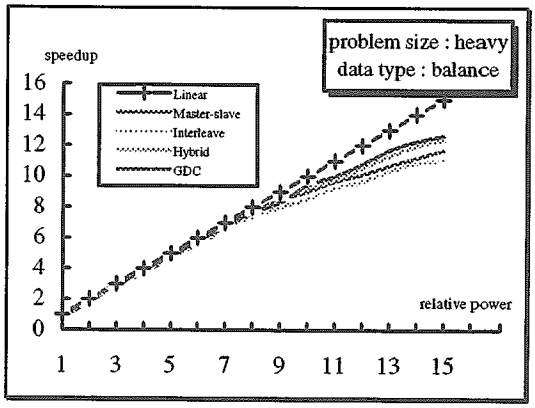


Figure 10. The experimental results for the ball image data under balance and imbalance



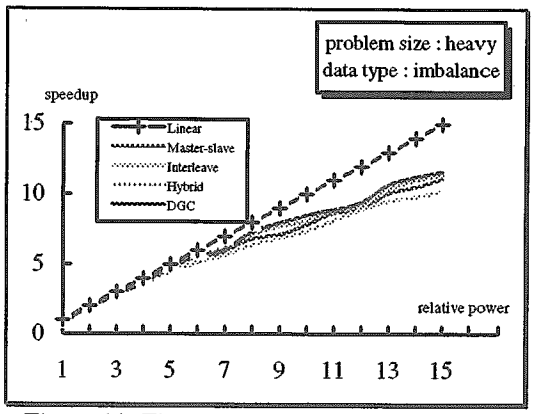


Figure 11. The experimental results for the gear image data under balance and imbalance

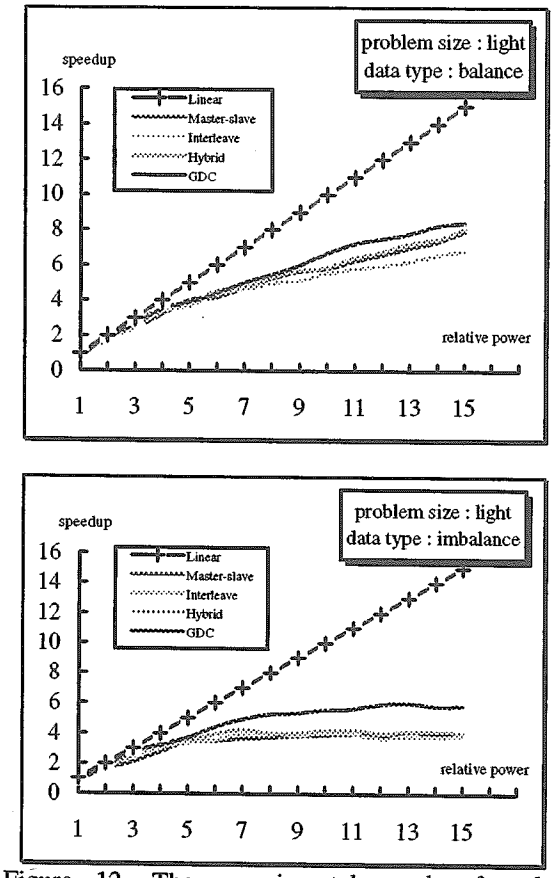


Figure 12. The experimental results for the tetrahedral pyramid image data under balance and imbalance

From the experimental protocol, we have the following observations.

1. The GDC algorithm gives the best performance for all types of data.
2. The hybrid scheme takes the advantages of both interleaved and master-slave schemes, it gives the second performance.
3. The interleaved algorithm gives the worst performance for all types of data.

   The static load scheme cannot adapt to the unequal computation power of workstation that causes the worst performance of the interleaved algorithm.
4. The higher the complexity of image data, the better the performance in speedup.
5. The performance deteriorates under imbalance image data.

   As expected the performance for the imbalance image is worse than for the balance image. They tend to saturate earlier.
6. The lower the complexity of image data is, the earlier the saturation occurs.
7. We will get the low speedup when we adopt light and imbalance scene as the testing data.

We further the study by doing the experiment for the number of slaves versus the execution time. The result is shown in Figure 13. When the number of slaves is increased, the execution time is decreased. But it is saturated very soon. That means to spawn a couple of slaves in the workstation will speed up the execution. Because the system will allocate the more cpu resources than a single slave in the workstation.

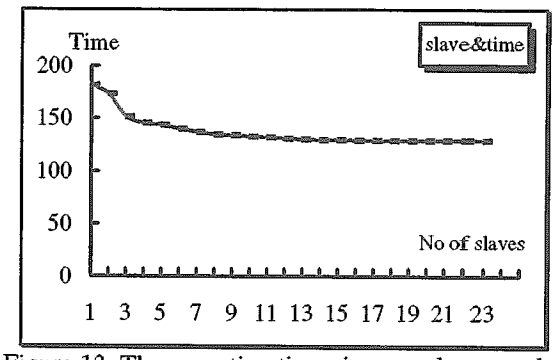


Figure 13. The execution times in second versus the number of slaves spawned in one workstation.

Fault tolerance

To test the fault tolerance ability, 9 workstations are added in the PVM's environment. After the master process gathered 300 scanlines from the slave processes, two workstations(i.e. two Sparc 20) are deleted from PVM console. In the master process, we adopted the blocking receiving within timeout. So when the given hosts were deleted, the master process can still receive the rendering result from other slave process. The ball image is used as the testing scene. The results in Table 3 show that the rendering process is still running after two workstations are deleted, thus the fault tolerance ability is proved.

Table 3. The performance of fault tolerance after deleting two Sparc 20 from PVM's console

	Sparc 2	Sparc 10	Sparc 20	Total relative power	Performance
No of workstations	3	2	4	-	-
Relative power	0.77	1	1.96	-	-
Before deleting hosts	3*0.77	2*1	4*1.96	12.15	14 secs
After deleting hosts	3*0.77	2*1	2*1.96	8.23	19 secs
Degradation	0	0	3.92	3.93	42.85 %

From the results shows that the experimental protocol proved to be effective in evaluating the performance and characteristics of different algorithms. The proposed GDC performs the best and has fault tolerance ability.

5. Conclusions

We have presented a new dynamic load balancing strategy, called global distributed control. We have compared the algorithm to a couple of traditional load balancing schemes. In order to characterize the performance of different algorithms, we designed an experimental protocol based on the parameters of problem size, data type, number of workstation, and environment configuration. Results show that the global distributed control gives the best performance for all type of data, the hybrid approach gives the second, the master-slave is the third, and the interleaved gives the worst performance. The global distributed control not only has the ability to dynamically adjust the load, but also has the fault tolerance ability.

Acknowledgment

This research was supported in part by the National Science Council of Taiwan, R.O.C., under contracts NSC-86-2213-E-218-011 and NSC-86-2213-E-110-028.

References

1. Acceleration Techniques,'' appeared in ``An Introduction to Ray Tracing,'' edited by A.S. Glassner, Academic Press, pp. 201-262, 1989.
2. T.L. kay and J.T. kajiya, ``Ray Tracing Complexity Scenes,'' ACM SIGRAPH, 20(4): 269-278, Aug. 1986.
3. V. S. Sunderam, "PVM: A Framework for Parallel J. Arvo and D. Kirk, ``A Survey of Ray Tracing Distributed Computing," Concurrency: Practice and Experience, Vol. 2 No. 4, pp. 315-339, Dec. 1990.
4. S. Ahuja, N. Carriero. and D. Gelernter, "Linda and Friends," IEEE Computer, Vol. 19, N0. 8, Aug. 1986.
5. J. Flower, A. Kolawa, "The Express Programming Environment," Parasoft Corporation Report, July 1990.
6. C. Giertsen, and J. Petersen, "Parallel Volume Rendering on a Network of Workstations," IEEE Computer Graphics and Applications, Nov. 93, pp. 16-23.
7. C. H. Cap and V. Strumpen, "Efficient Parallel Computing in Distributed Workstation Environments," Parallel Computing 19 (1993) 1221-1234.
8. White, A. Alund, and V. S. Sunderam, "Performance of the NAS Parallel Benchmarks on PVM Based Networks," in http://www.netlib.org/pvm3.
9. V. Strumpen and T. L. Casavant, "Exploiting Communication Latency Hiding for Parallel Network Computing: Model and Analysis," IEEE, 1994, pp. 622-627.
10. W. LEFER, ``An Efficient Parallel Ray Tracing Scheme for Distributed Memory Parallel Computers," Parallel Rendering Symp. 1993, San Jose, pp 77-80.
11. S. Whitman, ``A Task Adaptive Parallel Graphics Renderer," Parallel Rendering Symp. 1993, San Jose, pp 27-34.
12. U. Neuman, ``Communication Costs for Parallel Volume-Rendering Algorithms,"
13. H. Kobayashi, S. Nishimura, H. kubota, T. Nakamura, and Y. Shigei, `` Load Balancing Strategies for a Parallel Ray Tracing System based on Constant Subdivision,'' Visual Computer, 4(4): 197-209, Oct. 1990.
14. P. Dew, N. Holliman, D. Morris, and A. de. Pennington, ``Techniques for Rendering Solid Objects on a Processor Farm,'' in C. Askew, editor, Proceedings of the Ninth Occam User Group Conference, pp. 153-168, Southampton, 1988.
15. E. Haines, ``A Proposal for Standard Graphics Environment,'' IEEE Computer Graphics and Applications, 7(11): 3-5, Nov 1987.
16. M. Potmesil and E.M. Hoffert, ``A Parallel Image Computer with Distributed Frame Buffer: System Architecture and Programming,'' in: Eurographics 89, North Holland, Hamburg, 1989.
17. M.B. Carter and K. A. Teague, ``The Hypercube Ray Tracer,'' in Proc. the 5th Distributed memory Computing Conference, 1990.
18. F.C. Crow, G. Demos, J. Hardy, J. McLaughlin and K. Sims, ``3D Image Synthesis on the Connection Machine,'' in P.M. dew, T.R. Heywood, and R.A. Earnshaw, editors, Parallel Processing for Computer Vision and Display, Chapter 18, pp. 254-269, Addison Wesley, 1988.
19. M.B. Carter and K. A. Teague, ``The Hypercube Ray Tracer,'' in Proc. the 5th Distributed memory Computing Conference, 1990.
20. D. May, ``Toward general purpose parallel computers,'' MIT Press, Cambridge, 1989.
21. J. Packer, `` Exploiting Concurrency: a Ray Tracing Example,'' in: the transputer application notebook, Inmos, 1989.
22. S. Whitman, "Dynamic Load Balancing for Parallel Polygon Rendering," IEEE Computer Graphics and Applications, July, 94, P 41 ~ 48.
23. H. Nakanishi, V. Rego, and V. Sunderam, "On the Effectiveness of Superconcurrent Computations on Heterogeneous Networks," in http://www.netlib.org/pvm3.
24. B. K. Schmidt and V. S. Sunderam, "Empirical Analysis of Overheads in Cluster Environments," in http://www.netlib.org/pvm3.