Parallel Ray-Tracer on Shared Memory Multiprocessor Machines Connected via Internet

Vlastimil Havran
CTU, Fac. of Electrical Eng., Dept. of Computer Science and Engineering
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
e-mail: havran@cs.felk.cvut.cz

Abstract

This paper deals with parallelization of the ray-tracing algorithm using shared memory multiprocessor machines connected via Internet. The aim of our work is to show the advantages and the problems rising from the real implementation and dynamic load balancing via long response network. The computation is done in heterogeneous computational environment.

Key words: computer graphics, parallel ray-tracing, shared memory, load balancing

1 Introduction

Parallel implementation of sequential algorithms does not bring changes into the program structure only, but it forces programmers to move their interest toward specific hardware platform. Optimal sequential algorithms have to be mostly modified so that their efficiency is reduced by additional overhead needed for synchronization and communication between processors.

Our research team has been interested in the parallelization of ray-tracing for a several years. We started in 1993, with the implementation on a massively parallel system of T800 Inmos transputers. Later, the implementation was moved to loosely coupled network of workstations and *subdivision space scheme* [8]. Afterwards, we have aimed our research at the optimization of the ray-tracer on a shared memory machine. Currently, we are interested in utilization of shared memory machines connected via network with small throughput. Idea of parallel ray-tracer working on more shared memory multiprocessor machines is based on *image subdivision scheme* [1]. This approach is well suited to shared memory architecture, because the whole scene is stored in memory only once for one computer and it is accessed by more processors at the same time. Parts of the image are rendered by processors independently. Theoretically, the speedup of the ray-tracer on shared memory is almost linear, e.i., the efficiency is slightly below one. The speedup of the Internet ray-tracer (**inet-rt** in the next) is more difficult to define, because the performance of machines can be different. It depends on several factors – global load of shared memory multiprocessor machine, utilization of coherence, number of processes, number of shared memory clusters, the network throughput, and latency of the network during the computation.

A similar idea has been developed independently by Mental Images in Berlin [2] as well. As we have not found the details of its algorithm, the implementation, and the results satisfactory described anywhere, we have decided to parallelize and test our own ray-tracer.

2 Implemented techniques

In the first step, we have modified the sequential ray-tracer developed by our team for shared memory parallel implementation [5]. This ray-tracer was used as a reference for measuring the speedup of parallel solution. Although simple and well-known approach called ”process farming” [1] was used, the algorithm of the load balancing via Internet is not equal to original solution.

The implementation for shared memory machine is simpler, because we need to solve no problems rising from different processor performances, the different throughput, and the latency of network connecting the shared memory computers. The *inet-rt* can implement fewer load balancing techniques than the ray-tracer parallelized on shared memory.

Algorithm outline

The algorithm for *inet-rt* is composed of more different phases. At the beginning the data are distributed. The second phase is the distribution of computation load among the processors. In final phase the data are collected and saved to a file.

The basic terms

Let us define the terms used in this paper for better comprehension.

cluster ... the multiprocessor machine with shared memory.

main cluster ... the cluster, where the *inet-rt* process is invoked, where the scene data are distributed from, and where the image data are assembled.

remote cluster ... the cluster which is not main cluster, where the scene data from main cluster are received. It only serves as the computational unit for rendering and sends the image data back to the main cluster.

main master process ... the process in UNIX terminology running on main cluster. It schedules the load balancing both for local slaves and for processes running on remote clusters. This process is spawned by a user at the beginning of ray-tracing.

remote master process ... the process running on remote cluster. It serves for scheduling of the load between its local slaves.

local slave process ... the process running on main cluster. It performs computation using standard ray-tracing algorithm. The latency time for sending the message between main master process and local slave process is small.

remote slave process ... the process running on remote cluster, similarly as local slave process carries out ray-tracing algorithm. The remote slave process differs from local one by higher latency time needed for communication with main master process.

At the beginning phase, the main master process is spawned. It initializes both remote master processes and its local slave processes. The remote master processes spawn subsequently their slaves processes(remote slave processes).

The main master process asks the remote master processes about, whether they already have this description of the scene to be rendered, which is stored in an ordinary file (*scene file*). The remote master process can have it from ray-tracing done previously. If any remote master process does not have identical scene file, the main master process compresses current version of the scene file by LZW method and sends compressed data to corresponding remote master processes. They receive it and decompress it back to its original, then they save it to file.

After that, the main master process and remote master processes read the description of scene and convert it to an inner memory representation. They also build up spatial data structures required for BSP technique.

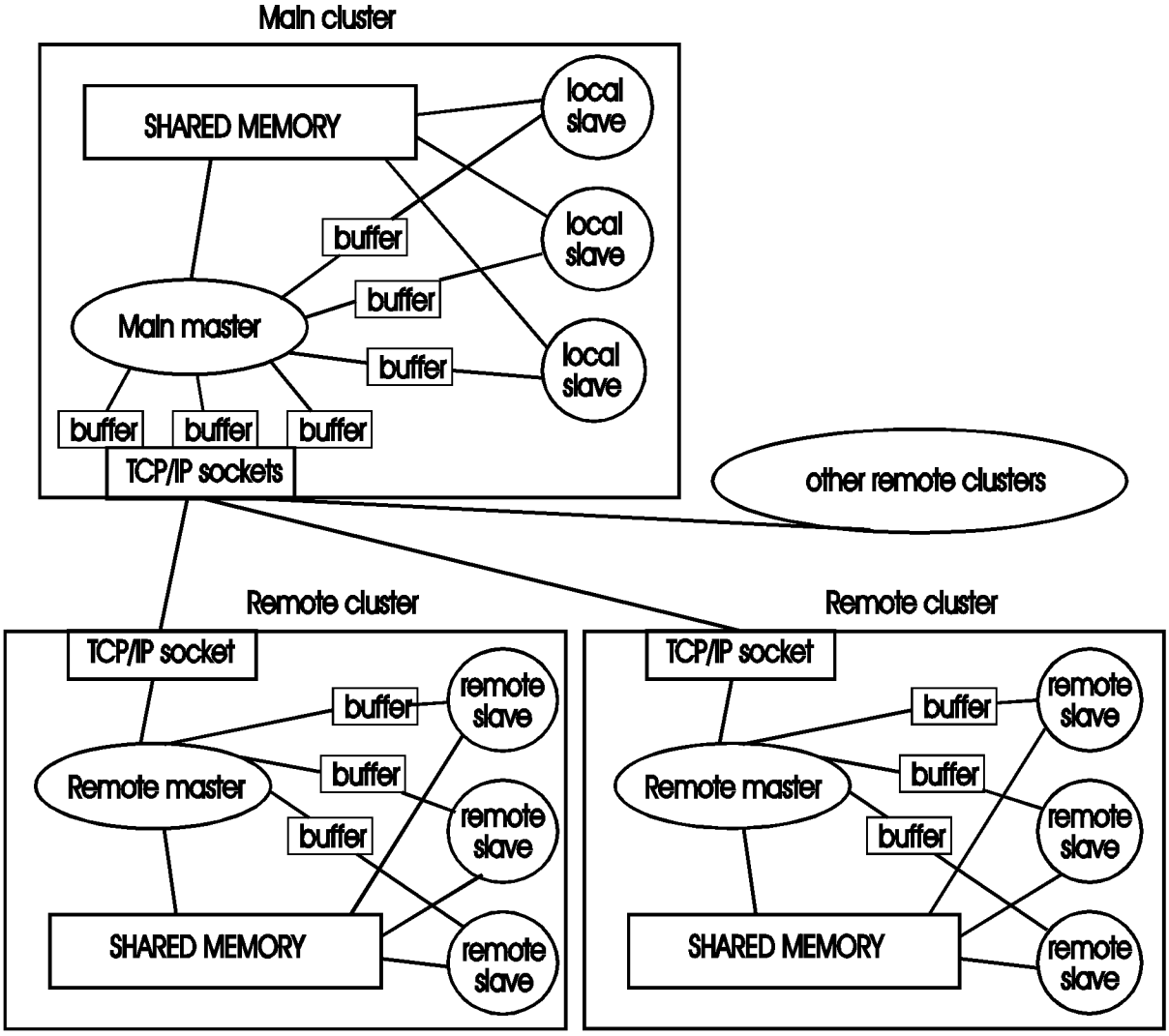

Figure 1: Architecture for Internet ray-tracing

In ray-tracing phase the main master process assigns the part of the image to local slave processes and to remote master processes. The remote master processes reassign these parts to its remote slaves processes. The whole image is assembled in the shared memory of main cluster. The strategy of load balancing is discussed in detail in the next section. RLE compression is used to transmit the image data from remote clusters back to the main cluster. The basic requirement made on the compression method for *inet-rt* in this application is its speed, although the compression ratio is important as well. The image is optionally displayed by auxiliary process running on main cluster during the ray-tracing phase.

At final phase, when the whole image is assembled in the memory of main cluster, the main master process saves it to the image file. As the last step, the network connections are closed and all processes exit.

Load balancing

In order to fully utilize all processors available the work load has to be distributed between them regularly during the whole computation time. Unfortunately, the characteristics of the ray-tracing algorithm does not allow to determine the complexity or the cost of computation for certain parts of an image beforehand. The second problem with load balancing is unequal 0 of the network connecting the supercomputer clusters (the latency [sec] and the throughput [KB/s]). The third

problem is the load of the machines by processes of other users running on the system.

We have tried to solve this situation by dynamic load balancing technique. The main master process controls the load of its local slave processes and the remote slave processes by different ways. The local slave processes guarantee their responses and there is no overhead for transmitting of the image data and the control commands between them and the main master process. The communication is done locally using the shared memory.

In order to decrease the latency of information exchange among the clusters and to use Internet effectively two-level load balancing strategy has been designed. The whole image is divided regularly into small blocks. The number of divisions is N_x and N_y in axis x and y respectively. Hence the number of all blocks to be computed is $N_X \times N_Y$. The block is the elementary unit of computation and cannot be further subdivided.

If the network has no latency and if the throughput of the network is very high, the algorithm for load balancing is simple. We can have one scheduler that assigns the blocks to be computed to all slaves. In Internet computational environment this method cannot be used any more with good efficiency. Let us describe designed scheduling mechanism.

The main master process finds out the number of its local slaves, the number of remote clusters, and the number of slaves running on them. The main master process controls its local slaves by simple mechanism, but the scheduling for remote clusters is more sophisticated. The master process sends to its slaves one block to each at the beginning of the computation. If the slave finishes the computation, then the master process registers this event and then sends to this slave a new block. If we want to decrease the latency and the slave's idle time, we can use **buffer** techniques well-known from many network protocols implementations (known also as sliding window strategy [9]). The buffer size determines, how many blocks are sent to the slaves or to the remote master processes regardless of any blocks have been computed.

In order to efficiently load balance ray-tracing there are the independent buffers for each local slave and for each remote master process. The buffers are not specified for remote slave processes.

The main master process determines the size of buffers for all. Simple strategy is used for local slaves, the size of the buffer is set to a constant (it can be set to one, then it works as without buffer). Load-balancing strategies for remote master were designed. The principle is based on measurements done on previously designed methods. The number of blocks for remote masters for first message is determined by the size of the buffer. The blocks definitions are sent in one message to save the time at the beginning of scheduling.

Strategies for remote masters

Following strategies declare the formula for computation of buffer size for remote masters:

C ... size of the buffer for remote master is set to a **constant** at the beginning of the computation. The buffer size remains unchanged during the computation.

P ... size of the buffer for remote master is **proportional** to the number of its slaves. It is calculated as $B = C \times N$, B denotes the buffer size, N is the number of remote slave processes in the remote cluster, and C is a constant. Size of the buffer is not changed during the computation process.

PD ... size of the buffer for remote master is determined as for P method. Besides the size for the remote buffers is **decreased**, when the number of blocks really computed and received by main master process exceeds a specific threshold. The threshold can be taken as a certain percentage from number of all blocks, for example 70%.

PDZ ... this strategy copies strategy PD. Besides, the buffer size is decreased to **zero**, when the number of received blocks exceeds a threshold(higher than for PD,e.g. 95%).

Let us describe the situation of a main master, when it has received the message and still a part of a picture is not computed. The master master process evaluates an expression f for local slave processes and for remote master processes. The expression f is computed as $f = B - BA - BC$, where BA denotes the number of blocks assigned by working process, BC is the number of blocks already computed by them and received by main master process, B is the size of buffer. The higher the f, the fewer blocks waits in buffer. The main master process chooses the buffer with the highest f and the next image block to be computed is sent to this process regardless of the sender of the last block received. The main master gives preference to its local slaves over remote masters, if f is equal for some of them.

The remote master works similarly over the buffers of its slave processes. It redistributes the blocks received from main master process among them. The two-level hierarchy decreases the latencies during load balancing process. Let us describe the strategy properties made from observations on measured times:

C ... it does not take into account the number of the processors on the remote system. Either the utilization of remote processors is small, or the remote masters are overloaded and local slave processes are idle.

P ... strategy considers the number of slaves in the remote clusters. The disadvantage of this control system is that at the end of the computation of an image the local slaves are likely idle. The main master process waits for image data sent by remote clusters. The utilization of the remote clusters is higher than for strategy C.

PD ... the remote clusters are not so loaded during the final phase of the rendering and it minimizes the risk of the overhead time. It is important, when and how the buffer size for remote master processes should be decreased.

PDZ ... strategy was done to improve PD. Our experiences based on measurements have shown, that at the end of the computation main master still waits for remote masters. One solution of this problem is to adjust the parameters mentioned for PD, but it can be never foreseen for different scenes. Therefore the computation of last blocks is left to the local slave processes. The main master process can decompress the image data received from remote masters clusters at the same time.

3 Measurements

Hardware platform

We have used all three super-computing centers equipped with shared memory multiprocessor machine located in Czech Republic (in Pilsen, in Prague, and in Brno). All three multiprocessor shared memory machines have for one processor nearly equal performance.

The first super-computing center in Pilsen uses the Alpha-Server 8400 System made by Digital, *KIRKE*. It is a 64-bit RISC machine with symmetric multiprocessing (SMP) extendibility. The system is equipped with eight 64-bit processors Alpha 21164 /300 MHz, 2 GBytes RAM memory, 4 GBytes swap space. The internal bandwidth of the data bus is 1.2 GB/sec and the processors are provided with a three-level cache. The internal cache on the chip is 8 KB, write back secondary cache is 96 KB and tertiary on board cache is 4 MB. Typical benchmarks for one processor are 7.43 SPECint95 and 12.4 SPECfp95. The operation system is Digital Unix.

The super-computing centers in Prague and in Brno are both equipped with Power Challenge XL, named (*HAL*) (*GROND*) respectively. They both have 12 CPU MIPS R10000 /195 MHz, 2048 MB RAM, and the swap space with 1x2GB, 5(2)x4GB SCSI disks. The internal bandwidth of data bus is 1.2 GB/sec. The internal cache is 32 KB and second level cache for each processor is 2 MB. Typical benchmark for one processor is 8.50 SPECint95 and 13.1 SPECfp95. Both systems

run on IRIX 6.2.

Parallel library

From programmer's point of view modified *P4* library [4] has been used. Latest release of P4 library uses for dynamic allocation of one shared memory variable simple first-fit strategy with computational complexity $O(n)$, where n denotes the number of variables already allocated. This strategy is inconvenient if the user process allocates and frees the data many times as during the initial phases of the ray-tracing. In such a case the preparation of data for parallel task can reach several hours in compare with a few seconds using standard C allocation library. Then the parallelism becomes meaningless. Therefore new allocation library with computational complexity $O(1)$ was implemented, but further discussion on this topic is out of the scope of this paper. The time requirements of implemented solution are comparable with standard C allocation library.

Description of measurement

Our goal was to measure rendering times for different strategies and to test, whether the current technology is well suited for *inet-rt* idea. Following parameters have to be balanced for successful work of *inet-rt* algorithm: the performance of processors, the latency, and the throughput of the network connecting the multiprocessor shared memory clusters. If the performance of the processors is higher than the throughput of Internet, the whole picture is computed quicker on one cluster before the scene description data are sent to remote clusters. If the throughput of Internet is extremely high and guaranteed, the *inet-rt* algorithm works as the system with virtual shared memory. The *inet-rt* lies somewhere between these two extremes.

The measurement of performance is more difficult than in case of single shared memory machine. The local times of clusters are not usually synchronized even it can be done relatively precisely using time servers. The next issue is the expression of **speedup** for heterogeneous computational environment. It is usually calculated as follows:

$$speedup = \frac{T(1)}{T(n)},$$

where $T(1)$ is sequential time and $T(n)$ is the time achieved by parallel algorithm with n processors. Optimal speedup is equal to n.

The speedup must be redefined for heterogeneous computational environment so that the definition is more correct. Let us have n machines and the time of the task on a given machine i is $T_i(1)$. For whole time $T(n)$ on N machines the speedup is computed as follows:

$$speedup = \frac{\sum_{i=1}^{n} T_i(1)}{n \cdot T(n)}$$

The speedup can be evaluated in both real time and user time of processes:

Real time is the time, which is important for the end user. It is a time interval from the beginning and to the end of the whole computational process, regardless of the number of processors involved. Real time depends on the load of the system, e.i., on the number of all user's processes during the rendering.

User time is bundled with a rendering process. It expresses a time, when the process is active, i.e., when a processor performs the code of the process. It is independent of the load of machine. This enables to compare the efficiency with sequential algorithm regardless of other processes running on the same machine, but it does not reflect the transmission of data via the network.

The set of user times is obtained, one for each process, on each cluster. The better the load balancing, the smaller the differences between the measured user times corresponding to ray-tracing. The maximal time of measurement from all processors is taken to compute speedup. To express the measure of imbalance and the properties of load balancing algorithm, the *change of the load* [1] for both real and user time has been used:

$$C = \frac{T_{last} - T_{first}}{T_{last}},$$

where T_{last} and T_{first} is the maximal and minimal time respectively measured by local slave and remote slave processes on all clusters.

If the change of the load is smaller, then the load balancing strategy is better. The goal of any load balancing strategy is to minimize the change of load measured by real time on main cluster and to maximize the speedup.

Scene characteristics

Two scenes have been used for testing of *inet-rt*. They both are standard PDF [3] ray-tracing benchmark data (*Balls, Tetra*). We have not intended to use more testing scenes because our previous experiments with shared memory machine have shown the speedup is roughly the same for different scenes. Therefore we have decided to test the algorithm on the data set for one scene with relatively small computation time and for one scene with the computation time about 10 times higher.

Name	# of elems.	# of light sources	# of inters.[10^6]	depth of rec.	HAL user[s]	KIRKE user[s]	HAL real[s]	KIRKE real[s]
Balls	7382	3	193.0	2	349.98	345.27	355.13	345.49
Tetra	4096	1	7.6	2	27.51	28.69	32.13	28.78

Both scenes were rendered with resolution 512×512 pixels, one ray per pixel. The binary space subdivision (BSP) technique was used for decreasing a number of intersection tests. The following table shows real and user times for rendering both scenes using one process only. The block size is 64×64 pixels (N_x=N_y=8).

4 Results

The following graphs show the measurements done on *HAL*, *GROND*, and *KIRKE*. We have tested all strategies C, P, PD, and PDZ. Strategy C and P have been shown too ineffective. Because of lack of space we only present the results of strategy PDZ in this paper, which have been evaluated as the best. Strategy PD is a little worse, but it can reach the same results in dependence on the load of network. The PDZ has been shown eliminating successfully this risk of latency at final phase of the computation.

The graphs in Fig. 2 show the speedup on *HAL*. The image subdivision $N = N_x = N_y$ is changed from 4 to 16 with step 4 and the number of processes is changed from 1 to 7 by 1. It actually corresponds the algorithm for shared memory.

The graphs in Fig. 3 show the speedup, if the number of local slave processes is a constant and the number of remote slave processes is changed. The measurements have been performed on *HAL* and on *GROND*. Unfortunately, the supercomputers and the network connecting them have been heavily loaded during measurements. The percentage threshold for decreasing the remote s's buffer size to 2 has been set to 60% and for zeroing to 90%.

Two following graphs in Fig. 4 show the change of the load upon the same conditions.

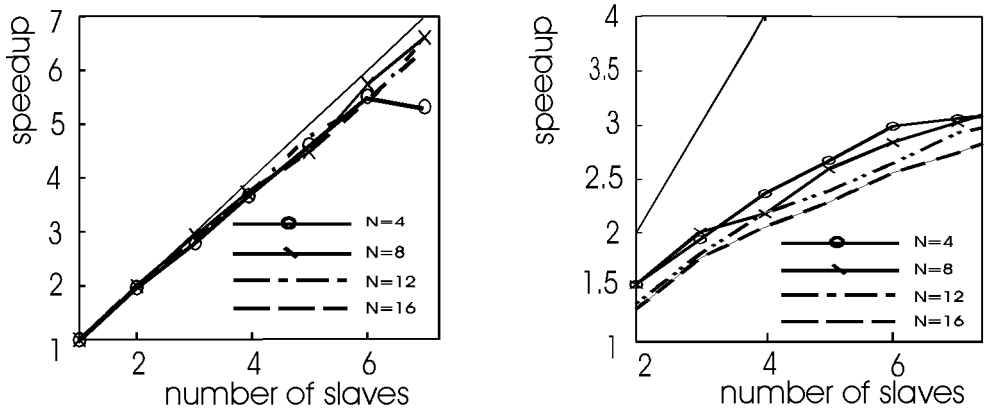

Figure 2: Speedup for balls in user(left) and real(right) time on *HAL*

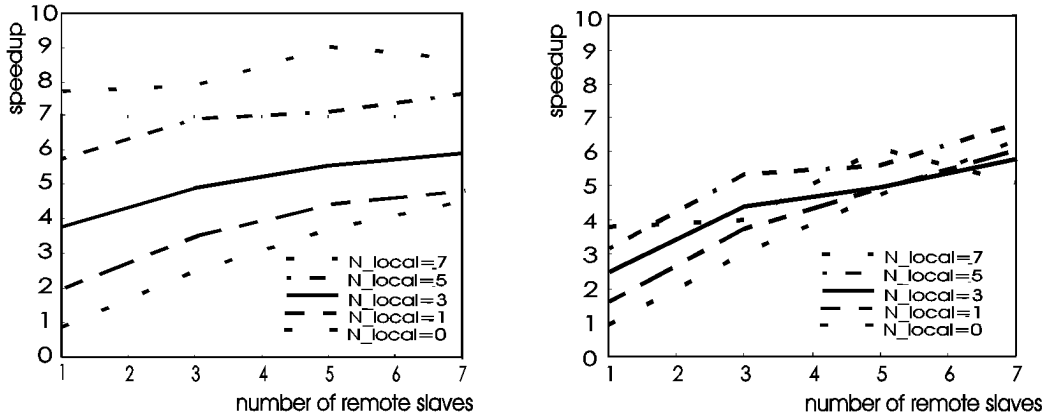

Figure 3: Speedup for balls in user(left) and real(right) time on *HAL* and *GROND*

We have also carried out the measurement on the combination *KIRKE* and *HAL*, the results are very similar and therefore they are not reported in this paper. Unfortunately, we have not succeeded to measure all the three supercomputers working together, because some problems with Internet address translation to *KIRKE* and with the use of parallel library on this system encountered. Another limitation concerning the number of processors has been caused by limitation of system resources. We have been able to test the number of the slaves up to the number seven per cluster, even if the number of processors is twelve for *HAL* and *GROND.*

The speedup for scene *balls* is increasing with the number of remote slave processes, but it is far from linear speedup. This outcome proves the idea of Internet ray-tracing is applicable upon condition that the network has sufficient throughput. The computation load of supercomputers

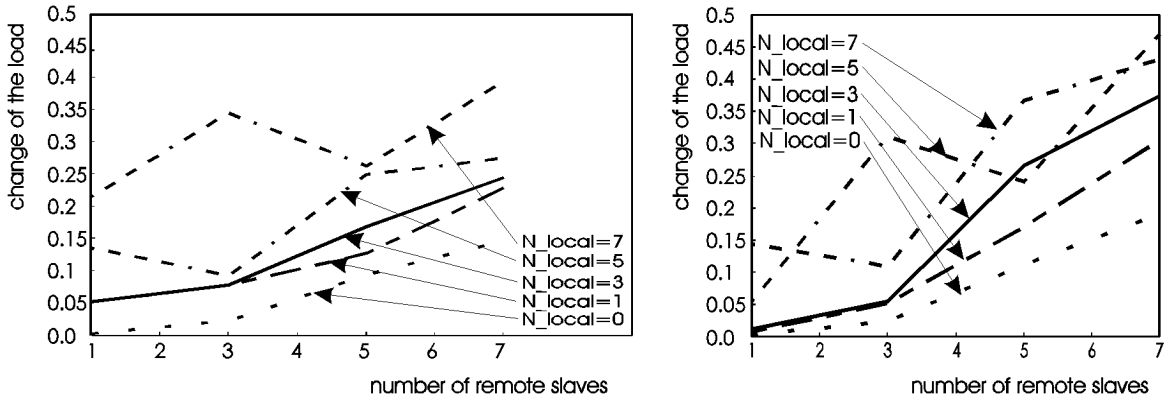

Figure 4: Change of the load for balls measured in user(left) and real(right) time on *HAL* and *GROND*

and the load of connecting network is changed during the rendering. Measurements performed for scene *tetra*, which have smaller sequential time required for rendering, have not brought any positive effect. The smallest time achieved is not below 7.5 seconds. The computation on three local slaves is below 7 seconds. The result is caused by small ratio of the sequential rendering time to the latency time of network.

5 Conclusions and future work

The idea of *inet-rt* has been implemented and tested in real environment. The results of measurement have shown, that current computer's technology and the state of the art of the software for ray-tracing enables to join the performance of supercomputers together via network with sufficient throughput. This collaborative computation scheduled by two-level hierarchy load balancing starts to be meaningful for images rendered sequentially longer than five minutes measured on configuration described in this paper.

In future, we are going to improve two-level hierarchical model of load balancing with analysis of scheduling by queuing theory and to design statistically controlled load balancing. We want to rework the parallel library to be more efficient for purposes of *inet-rt* and to improve the compression and decompression algorithm used.

References

[1] Green, S.: *Parallel Processing for Computer Graphics.* Research Monographs in Parallel and Distributed Computing, Pitman Publishing, London 1991.

[2] Rolf Herken, et al. : *High Quality Visualization of CAD Data.* ESPRIT Project 6173, Project Overview, July 1, 1992.

[3] Haines, E.: *A Proposal for Standard Graphics Environments.* IEEE Comp. Graph. & Apps., Vol. 7 (11), pp. 3–5, 1987.

[4] Ralph Butler and Ewing Lusk: *User's Guide to the p4 Parallel Programming System*. Argonne National Laboratory, October 1992, Revised in April 1994.

[5] Havran, V., Žára, J.: *Some Practical Aspects of Ray Tracing on Shared Memory Machine*, at Poster Section of First Eurographics Workshop on Parallel Graphics and Visualization, Bristol, UK, 26-27 September 1996.

[6] Horiguchi, S., Masayuki, K.: *Parallel Processing of Incremental Ray Tracing on a Shared Memory Multiprocessor.* The Visual Computer, Volume 9, pp. 371–380, Springer-Verlag, 1993.

[7] Keates M.J, Hubbold R.J.:*Interactive Ray Tracing on a Virtual Shared-Memory Parallel Computer*. Computer Graphics Forum, Volume 14, number 4, pp. 189–202, 1995.

[8] Menzel K. et al.: *Distributed Rendering Techniques using Virtual Walls.* Proceedings of First European PVM Users Group Meeting, Roma, Italy, October 9-11, 1994.

[9] Andrew S. Tanenbaum. Computer Networks. Prentice-Hall Software Series. Prentice-Hall International, Inc., Englewood Cliffs, N.J. 07632, 1981.