A Parallel Implementation of Bidirectional Ray Tracing on Transputer-based System

Toshiyuki Kawai, Masahiko Kami
Osaka Electro-Communication Univ.
Japan *

Tadamasa Teranishi
Plus One Inc.
Japan †

Jun-ichi Abeki, Hironobu Ohnishi
Mitsubishi Precision Co.,Ltd.
Japan ‡

Abstract

This paper describes a parallel processing scheme for bidirectional ray tracing which improves the reality of the image drastically. It is organized from two stages, i.e. light ray tracing and viewing ray tracing. In light ray tracing, pairs of a light source and an element of objects are distributed to each processor dynamically. In viewing ray tracing, we use a screen subdivision algorithm. Our system consists of 86 transputers. All of the scene data are placed on shared memories, and only the required parts of them can be transferred to the cache memory on each processor by means of DMA on-the-fly. We also show some experimental results.

1 Introduction

Ray tracing[21] is a simple technique which can generate realistic 3D image. Various extensions of this method improving the reality also have been proposed[1, 5, 17, 19, 18, 20, 11]. However, these methods require the great amount of computation. One of the successful solution to reduce the computing time is a parallel processing. Some parallel architectures and machines have been proposed[2, 3, 4, 6, 7, 9, 10, 13, 14, 15, 16, 22], but none of them takes account of those extensions.

We also have developed a multi-transputer CG system called MAGG (Mitsubishi Advanced Graphic Generator) and a parallel ray tracing renderer for this system[12]. We have devised fast communication between the large shared memory and the local memory on each processor by means of DMA transfer instead of serial link transfer. Our renderer is based on a screen subdivision algorithm and a simple dynamic load balancing method.

This time, we have implemented the bidirectional ray tracing renderer on this system. This renderer has two major paths. One is light ray tracing, another is viewing ray tracing. We use these terms rather than backward and forward ray tracing.

In this paper, at first we show the system overview and the parallel processing scheme, and then show the results of the performance evaluation.

2 Hardware Configuration

The CG system MAGG consists of an I/O Processing Unit(IOP), five cards of Graphic Processing Unit(GPU), two cards of HDTV Frame Buffer(FB) and Video Input/Output Units. These are connected to two busses called G-busses as shown in Figure 1. Each G-bus is 32 bit width and the transfer rate is up to 84MB/sec. Round robin bus arbitration is used. These two busses are equivalent and every unit is able to use

*18-8 Hatsu-cho, Neyagawa, Osaka 572 Japan kawai@kwlab.osakac.ac.jp
†8-10-603 Toyotsu-cho, Suita, Osaka 564 Japan
‡345 Uemachiya, Kamakura, Kanagawa 247 Japan

/hichever it is unused, so that the occurrence of the bus onflict will decrease.

IOP consists of 68020/68030, 68882(20MHz), 8MB ɔcal memory, and 8MB shared memory which can be ccessed by FPs described below. It also has two chanels of serial link adaptor, RS232C/422 ports, VME, CSI and floppy disk interfaces. So that it can comıunicate with FPs, VTR, other hosts, hard disks and ther peripherals.

GPU consists of a Fork Processor(FP) and 16 Node 'rocessors(NP). All of them are INMOS T800-20 transıuters. They are connected together with serial links nd DMA bus which is 64 bit width as shown in Figure . The FP and the outer 10 NPs are also connected to 'Ps or NPs on other GPUs with serial links as shown ı Figure 1. Up to 7 GPUs are able to connect to the ;-bus.

Each FP has 256KB local memory and 8MB shared ıemory. All of the FPs and IOP can access the shared ıemories and FB mapped onto the memory space of hem via G-bus.

Each NP has 512KB local memory. DMA transfer ; available between this local memory and any shared ıemories or any FBs through DMA bus and G-bus, /hich is under the control of FPs. In the case of DMA ransfer or shared memory access by FPs, if the target hared memory is inside of the GPU, G-bus will not be sed.

Host processor(HP) is also T800-20 transputer on the ɔoard in a host computer NEC/IBM PC, and has 2MB f main memory.

Each FB has 2048×1280 pixels and the display resoıution is 1920×1035. Up to 4 FBs are able to connect to he G-bus. Stored images in FBs can be recorded frame ɣ frame to VTR through Display Bus and Video Outıut Unit.

3 Bidirectional Ray Tracing

Backward ray tracing[1, 20] or bidirectional ray racing[5, 11] is two-pass algorithm for computing global lumination. The first pass computes view-independent omponents such as diffuse-to-diffuse and specular-to-iffuse light transport. Here we call this stage light ay tracing(LRT). The second path computes viewependent components such as diffuse-to-specular and pecular-to-specular light transport. This stage is same s the conventional ray tracing except ambient term nd diffuse component in the intensity calculation. The iffuse component computed in LRT is used instead of hem. We call this stage viewing ray tracing(VRT).

This time we modified and simplified the original alorithm for a parallel implementation, so that our alorithm has the following restrictions.

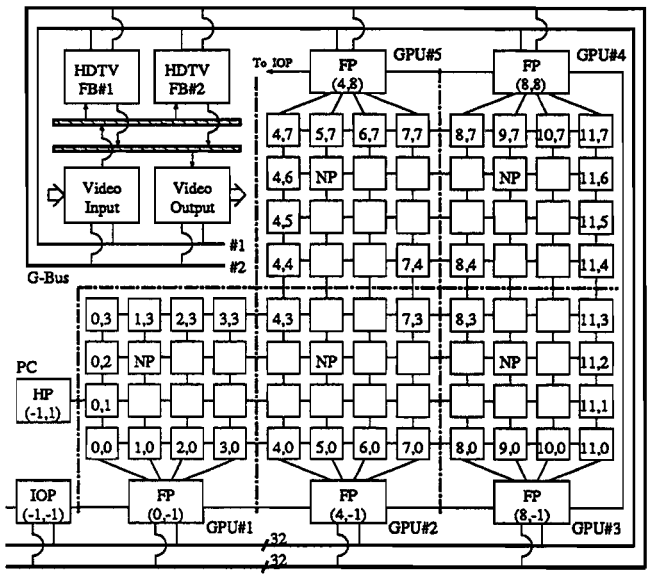

Figure 1: System configuration of MAGG.

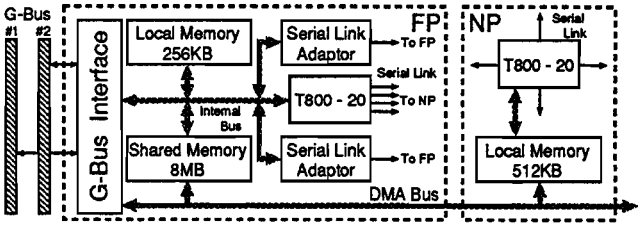

Figure 2: Hardware configuration of GPU.

- Using ideal point light sources only.
- Amount of the energy per unit solid angle from a light source is constant.
- Using triangle patches for geometry models.
- A triangle patch is subdivided into triangle elements for energy calculation.
- Omitting computation for diffuse-to-diffuse transport.

We use the space subdivision method[8] for the fast intersection seeking among rays and patches. We show our modified algorithm as follows.

3.1 Preprocessing

This stage is preprocessing for LRT and VRT, such as reading scene database from disks, geometry transformations and element generation.

3.2 Light Ray Tracing (LRT)

This stage is for distributing energy from the lights to the elements. Sampling rays are emitted from a light source toward an element. We call them primary light rays, and each ray has same energy. If some of them have intersections with the other elements, they will not reach the target element directly. So that the computation concerned with these rays will be discontinued. The other rays reaching the target element give the energy to it. It will be accumulated on it, and the secondary light rays will be emitted in the direction of reflection or refraction.

The secondary light rays which have no intersection with elements will be discarded. The others which have intersections with any elements will give the energy to them. This process will be repeated recursively.

The emission of primary light rays will be continued for all combinations of a light source and an element. Each iteration of this procedure can be executed individually. Finally, we can get the diffuse component by the direct light and the specular-to-diffuse component by the indirect light.

3.3 Energy Interpolation

This stage is for computing the energy at each vertex of the elements by interpolating the energy among the elements which share the vertex.

3.4 Viewing Ray Tracing (VRT)

This stage is for finding intersections and computing intensities by emitting rays from the eye toward each pixel to produce the final image. The diffuse component of the intensity at the intersection between a ray and an element is calculated by interpolating the energy at the vertices of the intersection element. The constant ambient term are also added to the intensity because of omitting the computation for diffuse-to-diffuse transport.

4 Parallel Processing Scheme

4.1 Process configuration

Two concurrent processes are running on a processor. One is the communication process written in OCCAM for data transfer between the processes on transputers. The other is rendering process written in C, which has two major stages(LRT,VRT) and two additional stages described above. The communication process has higher priority than the rendering process. This is because NPs have to relay the packet via serial links and then the communication time affects the efficiency of parallel processing greatly.

4.2 Communications

Processor communications can be done by not only serial link transfer but also DMA transfer between FP's shared memories and NP's local memory or between FBs and NP's local memory. These are under the control of the communication processes.

Serial link transfer is done by exchanging packets through the FIFO buffer which belongs to each communication process. The process picks up the packets for itself, or forwards them to neighbor processors via serial links. The size of normal packet containing control header and data is 1KB. The control header contains communication mode (broadcast or one-to-one), data size, and processor addresses of sender and receiver.

This time we use DMA transfer from FP to NP only. DMA transfer requires synchronization between FP and NP, so that the NP sends a request packet including the buffer address by serial link transfer to the FP on the same GPU card, and then waits for the completion of DMA. Once the FP receives the request packet, it invokes DMA transfer immediately after its preparation. If the shared memory is outside of the GPU in which the NP is located, the FP will get G-bus at first.

4.3 Preprocessing and Data Management

HP reads scene descriptions from the disk and transfers them to the FP located at $(0, -1)$. Geometric transformations and generating subspaces and elements are carried out on the FP located at $(0, -1)$.

Subspace data, patch data, element data and vertex energy data are stored in FPs' shared memories. Each kind of data is packed in 2KB blocks interleaving to five FPs. These blocks will be transferred to NPs by DMA in the following stages. NPs prepare the specified number of cache blocks (2KB/block) in the local memory for storing these data. If a NP requires some data, in case they are not on the cache, the NP will request them to the FP in the same GPU. If the NP does not have enough space to store them, it will free the existing data which has been unused for the longest time.

Then FPs transmit all other data such as light sources to the NPs in the same GPU by DMA. They are not expired from the NPs' local memories.

4.4 Light Ray Tracing

Each FP manages the element data on its own shared memory. The number of the elements are sufficiently greater than the number of NPs in a normal scene. These are assigned via serial links according to NPs' requests in order to balance the load of each NP (simple dynamic load balancing). A NP requests a pair of a light source and an target element to the FP in the

ame GPU. After receiving it from the FP, the NP will race the primary light rays toward the element. If the VP requires the other data not on cache in this procelure, it will also request them to the FP in the same GPU. Finishing all the trace, the NP will request a pair of a light and an element again.

In the case of accumulating the energy on an element luring the light ray tracing, the NP will send a packet or requesting the accumulation of the computed value o the FP which manages the element. This prevents rom two or more FPs might try to update the same lata at the same time. The energy of the element never e used in light ray tracing, so that there is no need to pdate the data in the cache.

If some NPs in a GPU have finished their jobs but all he pairs of a light and an element on the FP's shared nemory were already assigned, the FP will request to he other FPs which still have unassigned elements to et some of them.

4.5 Energy Interpolation

The FP located at $(0, -1)$ does this stage.

4.6 Viewing Ray Tracing

HP subdivides a screen area into a sufficiently greater number of subscreens than the number of NPs. These subscreens are assigned via serial links according to NPs' requests in order to balance the load of each NP simple dynamic load balancing).

NPs request a subscreen to the HP individually. Then NPs execute intersection seeking and intensity calculaion for each pixel in the subscreen assigned by HP. Computed intensity values in each scan line of the subcreen are packed and also passed to the FP via serial ink. Then NPs request a subscreen again.

FPs are ready to accept packets from the NPs. If a NP requests a data block, FP will transmit the applicable one to the NP by DMA, or if the packet is the esult of intensity calculation executed on the NP, then t will write the intensity values of pixels contained in he packet into the FB.

If some NPs have finished but the others have been till working when all the subscreens were already assigned, HP will request to the working NPs to return ome portion of the remaining area of the assigned subcreen (adaptive redivision of subscreens). HP will be ble to redistribute them to idle NPs.

5 Performance Evaluations

5.1 Conditions

We have made some experiments to evaluate our system. A scene for the experiments consists of 336 subspaces (336 memory blocks, 1block=2KB), 8 patches (1 block), 1664 elements (118 blocks), 1896 vertices (11 blocks) and 1 point light source as shown in Figure 3. The resolution of the image is 400×400.

The experiments are done on condition that reflection is limited to 1 time for saving the rendering time, varying the number of NPs and the cache size of the NPs. We have chosen 6 types of the number of NPs as 5, 10, 20, 40, 60 and 80, 4 sizes of the cache as 20, 40, 60, 80 blocks.

Unfortunately, we have some problems in the logic for the bus control so that we couldn't use two G-busses simultaneously.

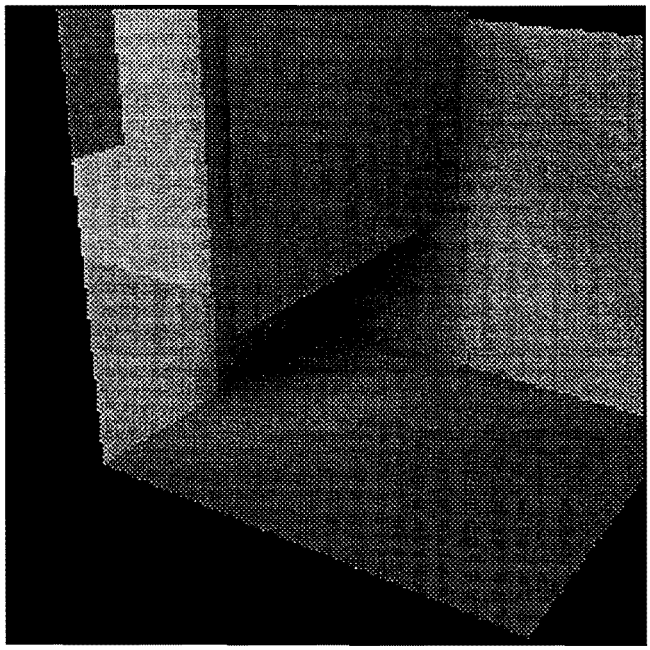

Figure 3: A scene for the experiments.

5.2 Rendering Time

The relationships between the rendering time of the image and the number of NPs are shown in Figure 4–7. Processing time of the each stage is also shown.

It can be seen from the results that the processing time for VRT is relatively longer than for LRT, and especially longer in the case that the cache is small such as 20 blocks. The processing time for LRT might be longer if the number of light sources or elements becomes large.

On the other hand, the time for energy interpolation is negligibly small.

It also can be seen from these figures that the rendering times are almost same when the cache size is larger than 40 blocks. This is because that the cache

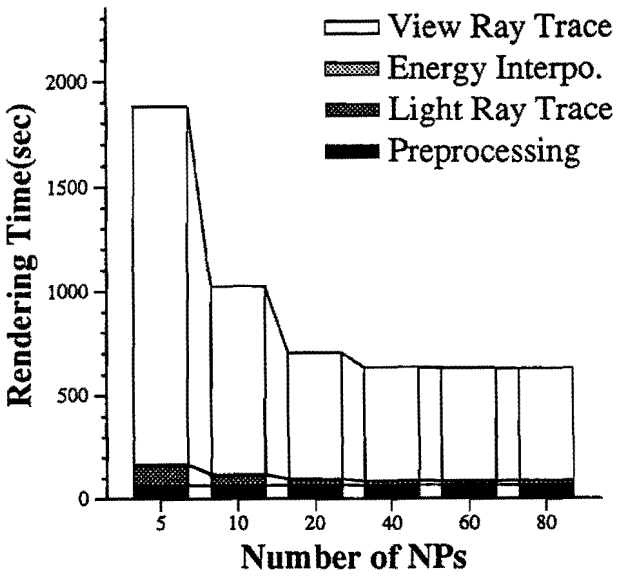

Figure 4: Rendering time (cache: 20 blocks).

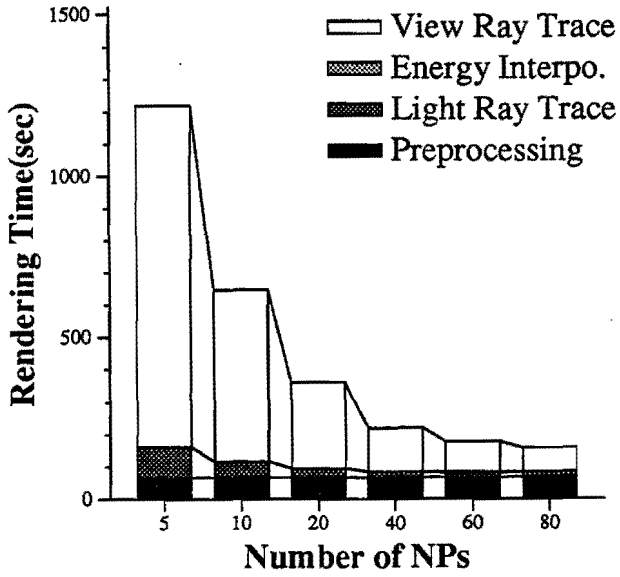

Figure 5: Rendering time (cache: 40 blocks).

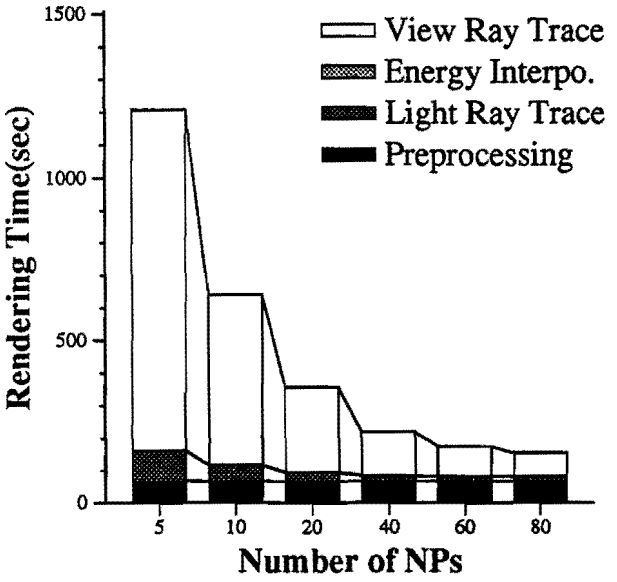

Figure 6: Rendering time (cache: 60 blocks).

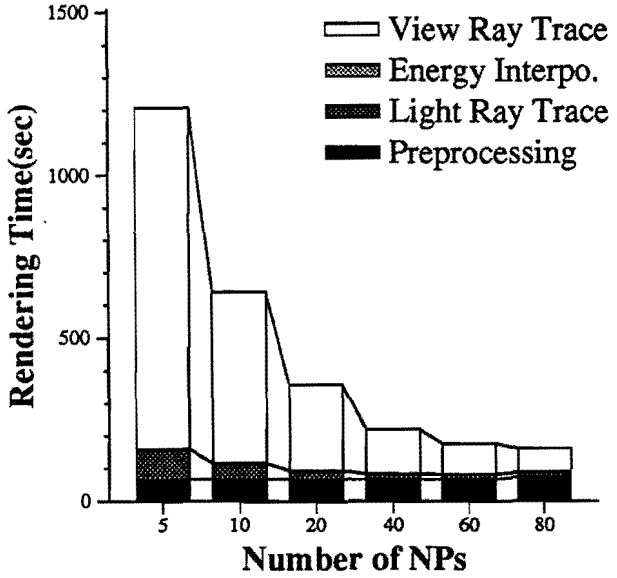

Figure 7: Rendering time (cache: 80 blocks).

hit ratio is nearly 100% in each case (see Figure 8). In other words, the rendering time and the performance described below must be mainly affected by the transfer time for subspace data. Whenever a ray enters a new subspace, in our algorithm, the data for the new subspace should be required.

5.3 Parallel Processing Performance

The relationships between the parallel processing performance and the number of NPs are shown in Figure 9–11. This performance is defined by $T(5)/T(n)$, here $T(n)$ is the rendering time of a whole image with n NPs.

It can be seen from the results that the total performance is not so good though the performance of VRT is linearly increasing when the cache size is larger than 40 blocks. This is because that the preprocessing time for generating subspaces and elements is relatively long. So that it should be improved. Now only 1 FP dose this, but 5 FPs should be used.

The performance for LRT is also saturated. This is because that each FP is saturated due to frequent requests from NPs and due to problems on bus control.

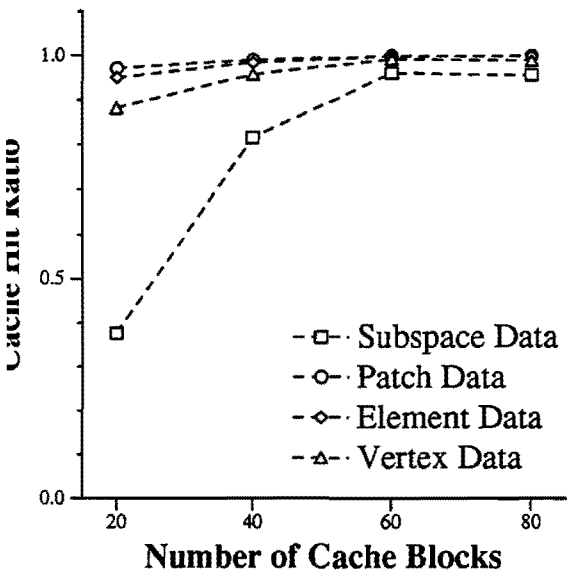

Figure 8: Cache hit ratio with 80 NPs.

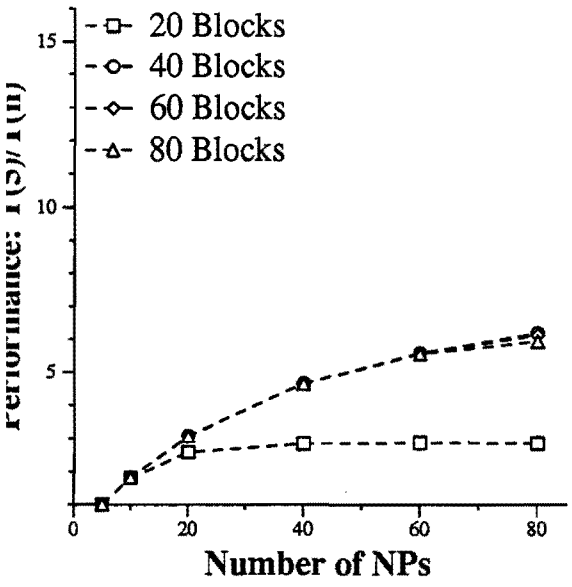

Figure 9: Parallel processing performance (total).

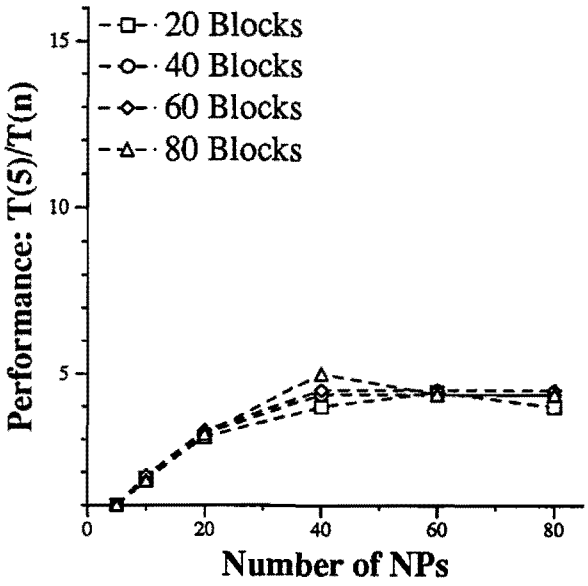

Figure 10: Parallel processing performance (light ray tracing only).

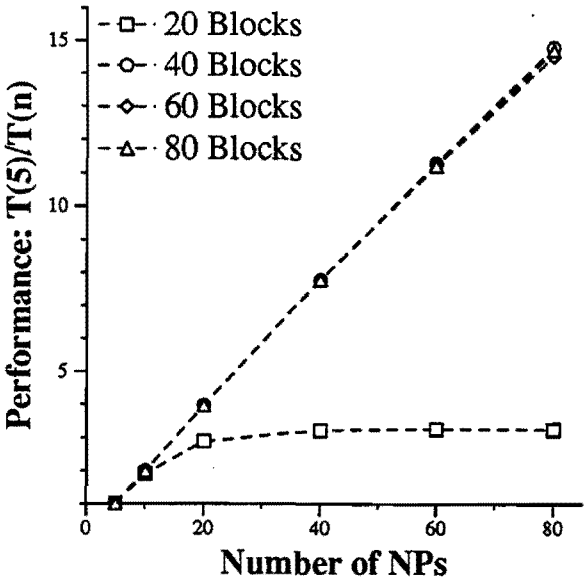

Figure 11: Parallel processing performance (viewing ray tracing only).

6 Conclusion

We have modified the bidirectional ray tracing and implemented on the transputer-based parallel CG system MAGG. We have some valuable results from the experiments described here:

- Preprocessing should be executed in parallel.
- Processing time for VRT should be improved.
- Transfer time or number of transfer for subspace data should be reduced.
- Problems on bus control should be fixed.

And based on these results, now we are making extensions and more experiments to improve the efficiency of the parallel processing.

Another problem is removing the restrictions of our method, such as accounting diffuse-to-diffuse light transport or various light sources. We are adding another path for diffuse-to-diffuse component, and making extensions on the emission of sampling rays from the lights.

References

[1] ARVO, J., AND KIRK, D. Backward ray tracing. In *ACM SIGGRAPH'86 course notes* (Aug. 1986), vol. 12.

[2] ATAMENIA, A., MERIAUX, M., LEPRETRE, E., DEGRANDE, S., AND VIDAL, B. A cellular architecture for ray tracing. In *5th Eurographics workshop on graphics hardware* (1990), pp. 85–91.

[3] BADOUEL, D., BOUATOUCH, K., AND PRIOL, T. Ray tracing on distributed memory parallel computers: strategies for distributing computations and data. In *ACM SIGGRAPH'90 Course 28* (Aug. 1990), pp. 185–198.

[4] BADOUEL, D., AND PRIOL, T. An efficient parallel ray tracing scheme for highly parallel architectures. In *5th Eurographics workshop on graphics hardware* (1990), pp. 93–106.

[5] CHATTOPADHYAY, S., AND FUJIMOTO, A. Bidirectional ray tracing. In *Proc. of CG International '87* (May 1987), pp. 335–343.

[6] CLEARY, J., WYVILL, B., BIRTWISTLE, G., AND VATTI, R. Multiprocessor ray tracing. Research report 83/128/17, University of Calgary, Oct. 1983.

[7] DEGUCHI, H., NISHIMURA, H., YOSHIMURA, H., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. A parallel processing scheme for three-dimensional image generation. In *IEEE ISCAS'84* (May 1984), vol. 3, pp. 1285–1288.

[8] FUJIMOTO, A., TANAKA, T., AND IWATA, K. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications 6*, 4 (Aprl. 1986), 16–26.

[9] GREEN, S., AND PADDON, D. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications 9*, 6 (Nov. 1989), 12–26.

[10] HÉBERT, M.-P., MCNEILL, M., SHAH, B., GRIMSDALE, R., AND LISTER, P. Marti—a multiprocessor architecture for ray tracing images. In *5th Eurographics workshop on graphics hardware* (1990), pp. 69–83.

[11] HECKBERT, P. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics 24*, 4 (Aug. 1990), 145–154.

[12] KAWAI, T., OHNISHI, M., ABEKI, J., AND OHNISHI, H. Transputer-based parallel ray tracing system using demand data transfer. In *7th Eurographics workshop on graphics hardware* (1992), pp. 95–105.

[13] KOBAYASHI, H., KUBOTA, H., HORIGUCHI, S., AND NAKAMURA, T. Effective parallel processing for synthesizing continuous images. In *Proc. of CG International '89* (June 1989), pp. 343–352.

[14] NISHIMURA, H., OHNO, H., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. Links-1: A parallel pipelined multimicrocomputer system for image generation. In *10th Ann. Int. Symp. on Computer Architecture* (June 1983), pp. 387–394.

[15] SATO, H., ISHII, M., SATO, K., IKESAKA, M., ISHIHARA, H., KAKIMOTO, M., HIROTA, K., AND INOUE, K. Fast image generation of constructive solid geometry using a cellular array processor. *Computer Graphics 19*, 3 (July 1985), 95–102.

[16] TAKAHASHI, T., YOSHIDA, M., AND NARUSE, T. Architecture and performance evaluation of the dedicated graphics computer: Sight. In *IEEE MONTECH'87 (COMPINT'87)* (Nov. 1987), pp. 153–160.

[17] WALLACE, J., COHEN, M., AND GREENBERG, D. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics 21*, 4 (July 1987), 311–324.

[18] WALLACE, J., ELMQUIST, K., AND HAINES, E. A ray tracing algorithm for progressive radiosity. *Computer Graphics 23*, 3 (July 1987), 315–324.

19] WARD, G., RUBINSTEIN, F., AND CLEAR, R. A ray tracing solution for diffuse interreflection. *Computer Graphics 22*, 4 (Aug. 1988), 85–92.

20] WATT, M. Light-water interaction using backward beam tracing. *Computer Graphics 24*, 4 (Aug. 1990), 377–385.

21] WHITTED, T. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June 1980), 343–349.

22] YOSHIDA, M., AND NARUSE, T. Trend of the computer graphics hardware. *Information Processing Society of Japan 29*, 10 (Oct. 1988), 1109–1115. In Japanese.