Transputer-based Parallel Ray Tracing System Using Demand Data Transfer

Toshiyuki Kawai, Mitsuhisa Ohnishi, Jun-ichi Abeki and Hironobu Ohnishi

ABSTRACT
This paper describes a parallel ray tracing system MAGG which has 86 transputers and HDTV frame buffers. Our system is based on a screen subdivision algorithm. In this algorithm, each processor essentially requires entire scene database. Therefore huge local storage should be required if the scene is complicated. In order to avoid this, shape descriptions in a scene database should be transferred to the processor when they are required. However, it would lead to lower parallel processing performance. 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. Some experimental results indicate it is effective to improve the efficiency of parallel processing.

1.1 Introduction

Ray tracing[18] is a simple technique which can generate realistic 3D image. However, this method requires the great amount of computation. One of the successful solution to reduce the difficulty is a parallel processing. Some parallel architectures and machines have been proposed[2, 3, 4, 6, 7, 11, 12, 14, 15, 16, 17, 19].

Some are based on a screen subdivision algorithm[7, 15, 17]. In this method, a whole screen area to be rendered is divided into a number of subscreens, which will be assigned to each processor. This is because the calculation of intersection and intensity at each pixel can be executed independently. However, each processor essentially requires entire scene database. Therefore huge local storage should be required if the scene is quite complicated, and the storage utilization of the whole system could be lower.

In the case of a space subdivision algorithm[1, 5, 6, 8, 9, 10, 14], an object space is divided into subspaces, which will be assigned to each processor. Rays proceed through the subspaces, and the calculation of intersection and local illumination can be done at each subspace. So that the each processor requires only a part of scene database. The number of the objects and the incidence rays in a subspace would affect the load of the processor. However, it is difficult to predict or control the number of rays and therefore difficult to balance the load.

We have developed a multi-transputer CG system called MAGG (Mitsubishi Advanced Graphic Generator) and the ray tracing renderer for this system[13]. Our system is based on a screen subdivision algorithm and a simple dynamic load balancing method. In order to avoid the storage problem, shape descriptions in a scene database should be transferred to the processor when they are required. In general, this method would cause frequent communication between processors, and it would lead to lower parallel processing performance as the result. 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.

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

1.2 Hardware Configuration

The CG system MAGG consists of an I/O processing unit(IOP), 5 cards of graphic processing unit(GPU), 2 cards of HDTV frame buffer and video input/output signal processing units.

These are connected to 2 independent busses called G-bus as shown in Figure 1.1. Each G-bus is 32 bit width and the transfer rate is up to 84MB/sec. Round robin bus arbitration is used. Both of them are equivalent and every unit is able to use unoccupied one, so that the occurrence of the bus confliction will decrease.

IOP consists of 68020/68030, 68882(20MHz), 8MB local memory, and 8MB shared memory which can be accessed by FPs described below. It also has 2 channels of serial link adaptor, RS232C/422 ports, VME, SCSI and floppy disk interfaces. So that it can communicate with FPs, VTR, other hosts, hard disks and other peripherals.

GPU consists of a fork processor(FP) and 16 node processors(NP). All of them are INMOS T800-20 transputers. They are connected together with serial links and DMA bus which is 64 bit width as shown in Figure 1.2. The FP and the outer 10 NPs are also connected to FPs or NPs on other GPUs with serial links as shown in Figure 1.3. Up to 7 GPUs are able to connect to the G-bus.

Each FP has 256KB local memory and 8MB shared memory which can be accessed by other FPs or IOP via G-bus. All of the FPs and IOP can also access the HDTV frame buffers mapped onto the memory space of them via G-bus. The total of 48MB of memories and frame buffers are shared among FPs and IOP.

Each NP has 512KB local memory. DMA transfer is available between this local memory and any shared memories or any frame buffers through DMA bus and G-bus, which is under the control of FPs. In the case of DMA transfer or shared memory access by FPs, if the target shared memory is inside of the GPU, G-bus will not be used.

Host processor(HP) is also T800-20 transputer on the board in a host computer NEC/IBM PC, and has 2MB of main memory. This time we use the PC as a host because of simplicity of the software development.

The HDTV frame buffer has 2048×1280 pixels and the display resolution is 1920×1035. Up to 4 frame buffers are able to connect to the G-bus. Animated frames generated and stored in frame buffers can be recorded frame by frame to VTR through display bus and video output unit.

1.3 Parallel Processing Scheme

Our software is organized from two concurrent processes running on a processor. One is the communication process described by OCCAM for data transfer between the processes on transputers. The other is rendering process described by C with ray tracing approach (Figure 1.4). 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 parallel processing performance greatly.

1.3.1 Communications

Processor communications can be done by not only serial link transfer but also DMA transfer between FP's shared memories or frame buffers 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

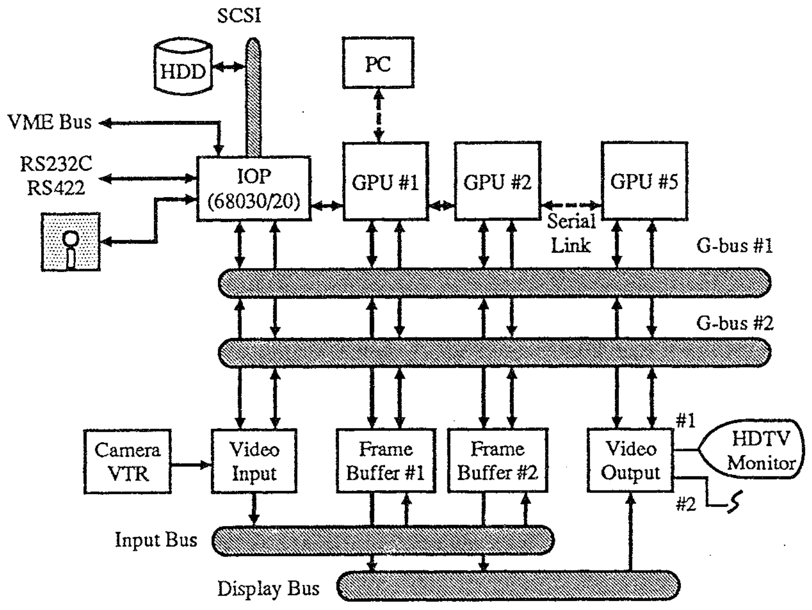

FIGURE 1.1. System configuration.

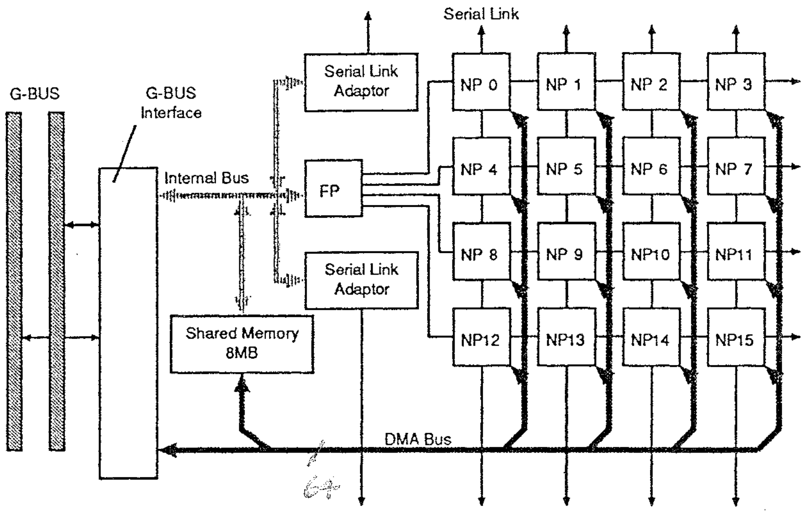

FIGURE 1.2. Hardware configuration of GPU.

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, sender and receiver processor addresses. In the NP network, transmitting time T (μs) is approximately $310 + \Delta row \times 234 + \Delta col \times 300$ (Δrow, Δcol: distence between NPs), on condition that the 84 bytes packet transmitted under no other process works except the communication process.

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

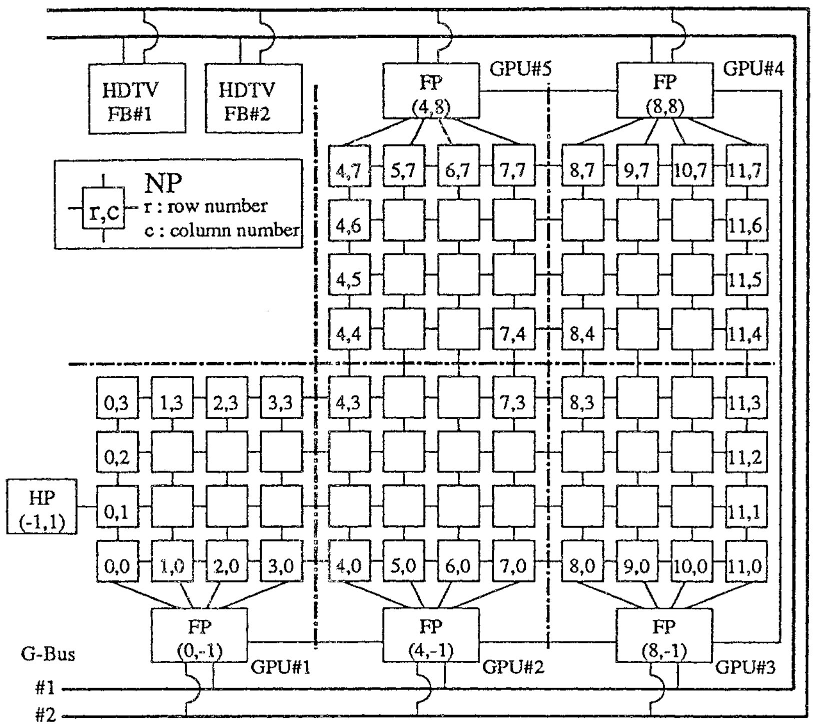

FIGURE 1.3. Processor connection.

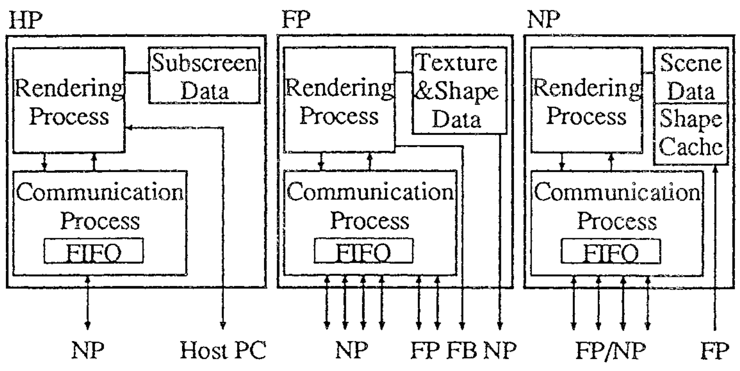

FIGURE 1.4. Software configuration running on transputers.

the NP is located, the FP will get G-bus at first.

Figure 1.5, 1.6 shows average transfer time of DMA and serial link. Used NPs are specified in the legends. Other NPs are idle and they will forward the packets to neighbors if they receive them. The measurement of transfer time starts at sending a request packet from NP to FP, and stops at completing transfer from FP to NP. Transfer data are placed on the shared memory of GPU#1 at the beginning. In the case of serial link transfer, one packet(1KB) includes 640 bytes of data. In Figure 1.6, 16 or 80 NPs have started to send request packets to FPs simultaneously. It can be seen from this figure that the number of GPU has only a little effect on the average transfer time.

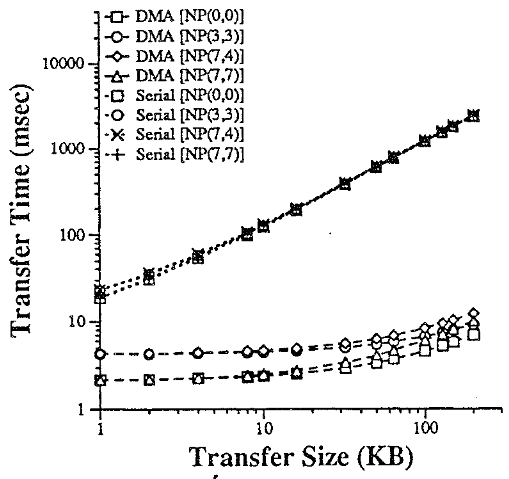

FIGURE 1.5. Average transfer time under only 1 NP is active.

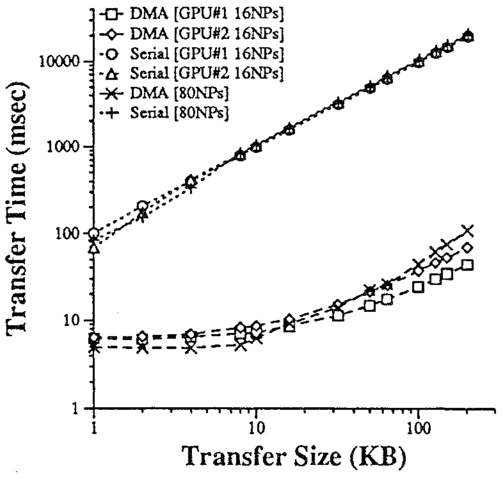

FIGURE 1.6. Average transfer time under 16 or 80 NPs are active.

1.3.2 Rendering

HP, FP and NP execute different tasks as rendering process. Figure 1.7 shows the control flow of the rendering process.

HP reads scene descriptions from the disk and transfers them to the FP located at $(0, -1)$. Then HP subdivides a screen area into a number of subscreens sufficiently greater 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).

Geometric transformations and construction of bounding boxes are carried out on the FP located at $(0, -1)$. The entire preprocessed scene data are placed on FPs' shared memories. FP$(0, -1)$ broadcasts an address table of the data placement to other FPs. After that FPs transmit all scene data except shape data and texture data for bump or texture mapping to the NPs in the same GPU by DMA.

Then 5 FPs are ready to accept packets from the NPs. If a NP requests a part of shape or texture data, FP will transmit the applicable data to the NP by DMA (demand data transfer), or if the packet is the result of intensity calculation executed on the NP, then it will write the intensity values of pixels contained in the packet into the frame buffer.

After receiving scene data from FPs, NPs prepare the memory space for storing shape data (shape data cache), and request a subscreen to the HP individually. Then NPs execute intersection seeking and intensity calculation for each pixel in the subscreen assigned by HP.

We use clipping technique and tree structured bounding boxes to decrease the cost of the intersection seeking[7]. At first NPs traverse the bounding box tree which is already transmitted from FPs, and find out which shape data are required. If some shape data are required, in case they are not on the shape data cache, the NP will request them to the FP in the same GPU (Figure 1.8). If the NP does not have enough space to store them, it will continue to free the space from the existing shape data which has been unused for the longest time. If some texture data are required in the intensity calculation, the NP will request them to the FP as well as shape data. Computed intensity values in each scan line of the subscreen are packed and also passed to the FP via serial link. Then NPs request a subscreen again.

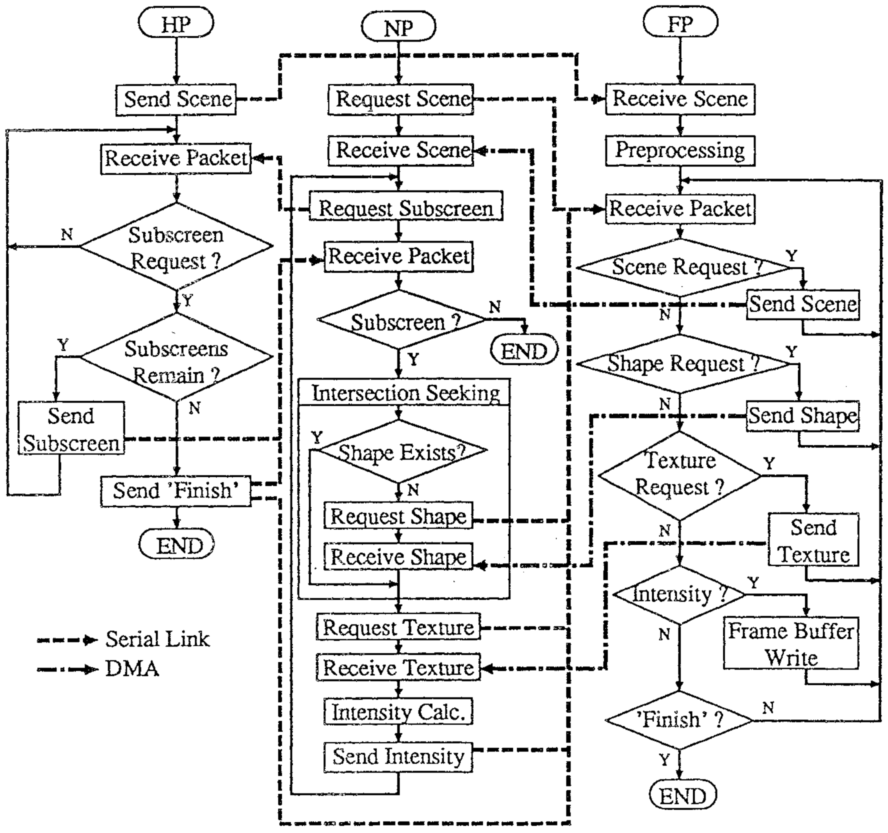

FIGURE 1.7. Control flow of rendering process.

1.4 Performance Evaluations

1.4.1 Conditions

We have made some experiments to evaluate our system, especially the efficiency of the demand transfer of shape data by DMA. Therefore we have also made the same experiments on batch transfer of shape data in which the entire scene data except texture data are transmitted to each NP at the beginning of rendering[13].

An experimental scene consists of 135 primitives including transparent objects and some with textures as shown in Figure 1.9. All of them except texture data can be stored in NP's local memory, so that the shape data cache is forcibly limited to 5, 10, 20, 40, 60, or 80% of their amount in order to invoke the transfer.

The resolution of the image is 640 × 480. The experiments are done on condition that reflection is limited to 1 time, refraction to 2 times with anti-aliasing, varying the number of processors and the number of subscreens. We have chosen 5 types of the number of processors as 1, 20, 40, 60 and 80, 4 types of the number of subscreens as 300, 1200, 4800 and 19200. Each type of subscreen contains 32 × 32, 16 × 16, 8 × 8 and 4 × 4 pixels.

1.4.2 Load Balancing

Figure 1.10 shows the maximum difference among the operating times of 80 NPs. The load balancing works better as the number of subscreens increases.

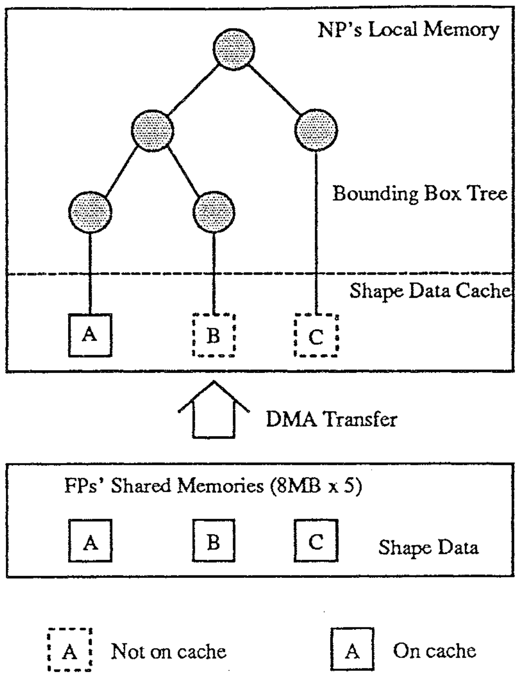

FIGURE 1.8. Demand data transfer and shape data cache.

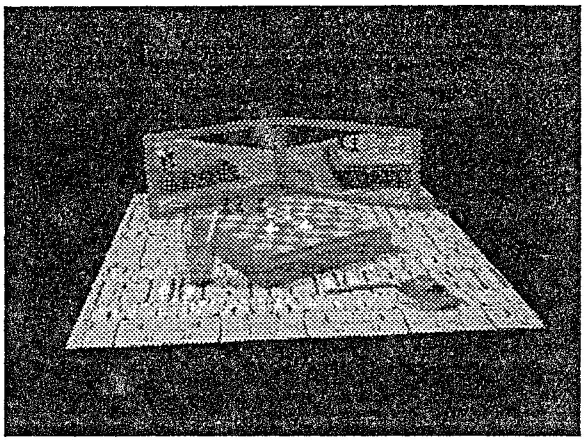

FIGURE 1.9. An experimental scene.

1.4.3 Rendering Time

The relationships between the rendering time of the image and the number of subscreens with 80 NPs are shown in Figure 1.11.

It can be seen from the results that to increase the number of subscreens improves the rendering time until the number of the subscreens exceed about 5000 when the shape cache becomes large. This is because the load balancing should be inadequate in the case of the smaller number of subscreens. However, the greater number of subscreens would cause frequent occurrence of the request for a subscreen from NPs to the HP. Thus, the communication time for the assignment of the next subscreen to NPs would be longer.

In the case, the shape data cache is small such as 5 or 10%, the rendering time is still decreasing when the number of the subscreens exceed about 5000. This is because that

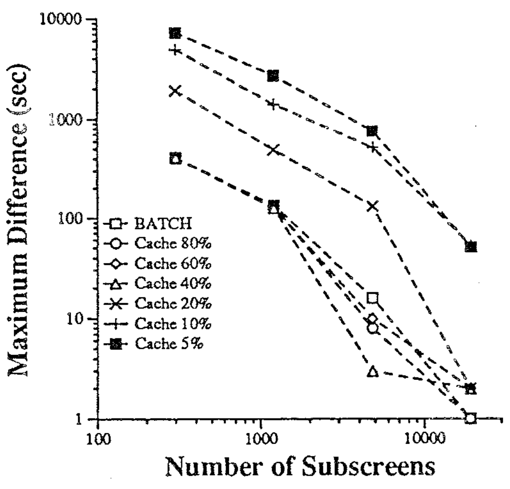

FIGURE 1.10. Effectiveness of load balancing.

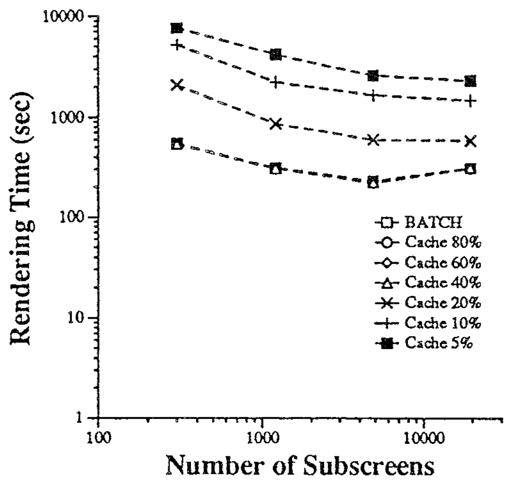

FIGURE 1.11. Rendering time using 80 NPs.

the computation time of one pixel becomes longer so that the frequency of the subscreen request decreases.

The optimum number of subscreens would be dominated by the distributions of the communication time and the computation time, therefore we could not know this before rendering.

It also can be seen from this figure that the rendering time is almost same when the size of shape data cache is larger than 40%.

1.4.4 Parallel Processing Performance

The relationships between the parallel processing performance and the number of NPs are shown in Figure 1.12–1.15. This performance is defined by $T(1)/T(n)$, here $T(n)$ is the rendering time of a whole image with n NPs.

In the case of the size of shape data cache is larger than 40%, the performance is almost same as that of batch transfer. Thus the figure was omitted.

Unexpectedly it has better performance when the shape data cache is small though the rendering time becomes longer. This is because that the performance would be also dominated by the frequency of the communication for a subscreen assignment as mentioned above.

Though there is no specification in the figures, the performance is terribly low in the case of the demand data transfer by serial link. This is because the communications waste almost time of the rendering.

1.5 Conclusions

We have rather good results from the experiments described here. When the shape data cache larger than 40%, the rendering time and the parallel processing performance are almost same as the batch transfer. Even though the rendering time is much longer than the batch transfer when the cache size is small, the parallel processing performance becomes more linear and better. This means we will be able to improve the rendering time by increasing the number of NPs. Here we can conclude the demand data transfer by DMA is effective, however, we will make more experiments to improve the efficiency of parallel processing furthermore.

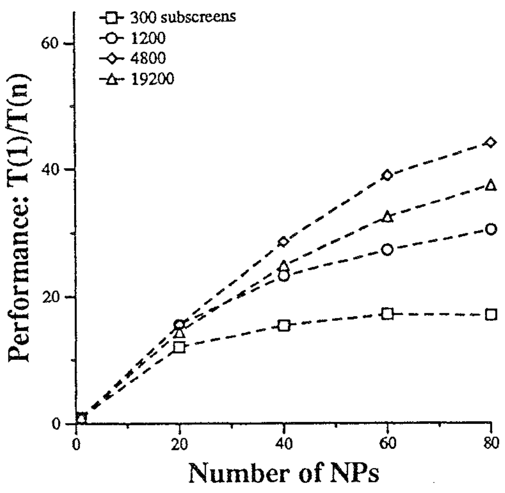

FIGURE 1.12. Parallel processing performance (batch transfer).

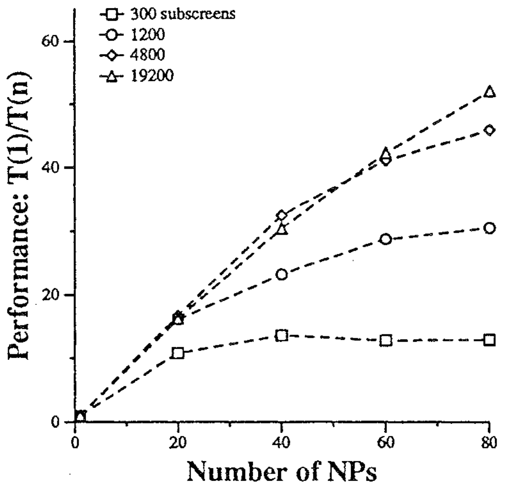

FIGURE 1.13. Parallel processing performance (demand transfer, cache size is 20%).

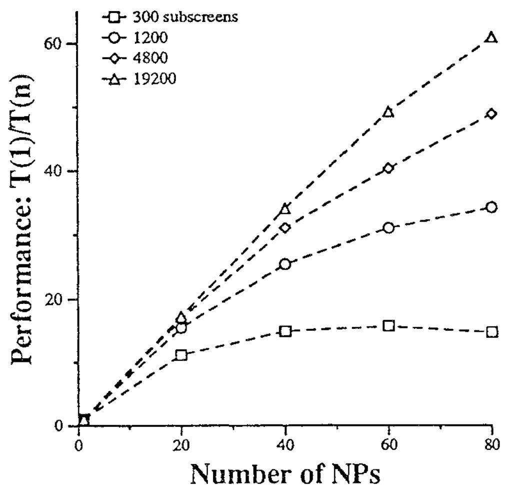

FIGURE 1.14. Parallel processing performance (demand transfer, cache size is 10%).

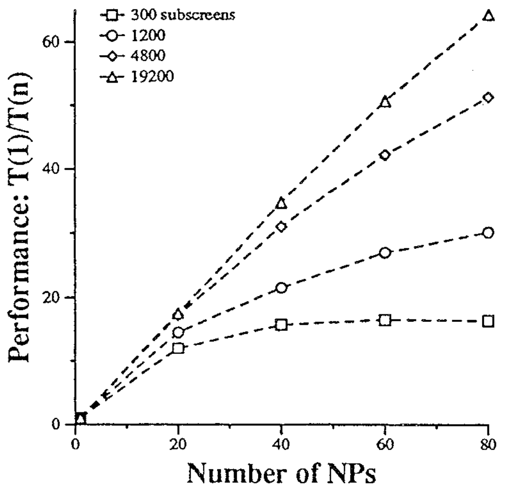

FIGURE 1.15. Parallel processing performance (demand transfer, cache size is 5%).

In our load balancing method, we need the appropriate number of subscreens to obtain the good results. In order to avoid this, an adaptive redivision of subscreens should be effective. If some NPs have finished but the others have been still working when all the subscreens were already assigned, HP will request to the working NPs to return some portion of the remaining area of the assigned subscreen. HP will be able to redistribute them to idle NPs. This method has been implemented and tested, we can obtain some good results and will report them soon.

Parallel processing should be also effective to speed up the backward or bidirectional ray tracing which improves the reality of the image drastically, thus we are making extensions to our algorithm.

1.6 References

[1] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision for ray tracing CSG modelled scenes. *The Visual Computer*, 3(2):98–108, Aug. 1987.

[2] A. Atamenia, M. Meriaux, E. Lepretre, S. Degrande, and B. Vidal. A cellular architecture for ray tracing. In *5th Eurographics workshop on graphics hardware*, pages 85–91, 1990.

[3] D. Badouel, K. Bouatouch, and T. Priol. Ray tracing on distributed memory parallel computers: strategies for distributing computations and data. In *ACM SIGGRAPH'90 Course 28*, pages 185–198, Aug. 1990.

[4] D. Badouel and T. Priol. An efficient parallel ray tracing scheme for highly parallel architectures. In *5th Eurographics workshop on graphics hardware*, pages 93–106, 1990.

[5] K. Bouatouch, M.O. Madani, T. Priol, and B. Arnaldi. A new algorithm of space tracing using a CSG model. In *Eurographics'87*, Aug. 1987.

[6] J.G. Cleary, B. Wyvill, G. Birtwistle, and R. Vatti. Multiprocessor ray tracing. Research report 83/128/17, University of Calgary, Oct. 1983.

[7] H. Deguchi, H. Nishimura, H. Yoshimura, T. Kawata, I. Shirakawa, and K. Omura. A parallel processing scheme for three-dimensional image generation. In *IEEE ISCAS'84*, volume 3, pages 1285–1288, May 1984.

[8] M. Dippe and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. In *ACM SIGGRAPH'84*, volume 18, pages 149–158, July 1984.

[9] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, Aprl. 1986.

[10] A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, Oct. 1984.

[11] S. Green and D. Paddon. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications*, 9(6):12–26, Nov. 1989.

[12] M-P. Hébert, M.D.J. McNeill, B. Shah, R.L. Grimsdale, and P.F. Lister. MARTI—A multiprocessor architecture for ray tracing images. In *5th Eurographics workshop on graphics hardware*, pages 69–83, 1990.

[13] T. Kawai, M. Ohnishi, J. Abeki, and H. Ohnishi. Ray tracing for parallel image generation system MAGG. In *Proc. of ATOUG'91*, pages 89–94, Sept. 1991.

[14] H. Kobayashi, H. Kubota, S. Horiguchi, and T. Nakamura. Effective parallel processing for synthesizing continuous images. In *Proc. of CG International '89*, pages 343–352, June 1989.

[15] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omura. LINKS-1: A parallel pipelined multimicrocomputer system for image generation. In *10th Ann. Int. Symp. on Computer Architecture*, pages 387–394, June 1983.

[16] H. Sato, M. Ishii, K. Sato, M. Ikesaka, H. Ishihara, M. Kakimoto, K. Hirota, and K. Inoue. Fast image generation of constructive solid geometry using a cellular array processor. In *ACM SIGGRAPH'85*, volume 19, pages 95–102, July 1985.

[17] T. Takahashi, M. Yoshida, and T. Naruse. Architecture and performance evaluation of the dedicated graphics computer: SIGHT. In *IEEE MONTECH'87 (COMPINT'87)*, pages 153–160, Nov. 1987.

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

[19] M. Yoshida and T Naruse. Trend of the computer graphics hardware. *Information Processing Society of Japan*, 29(10):1109–1115, Oct. 1988. In Japanese.