AN IMPLEMENTATION OF A PARALLEL RAY TRACING ALGORITHM ON HYBRID PARALLEL ARCHITECTURE

Chang-Geun Kwon, Hyo-Kyung Sung, and Heung-Moon Choi
The School of Electronics and Electrical Engineering,
Kyungpook National University, Taegu 702-701, Korea
pdp@ee.kyungpook.ac.kr

ABSTRACT

In this paper, we present a parallel ray tracing algorithm on hybrid parallel architecture with processor farm model to speed up the ray tracing. Hybrid parallel architecture, a hybrid of a tightly- and a loosely-coupled one, is used in which reconfiguration for local and virtual shared memory is made through a crossbar network with local and global bus. The proposed architecture enhances the overall performance of the parallel ray tracing by reducing the data communication time between the processors in dynamic load balancing while maintaining data coherency. The proposed algorithm is implemented on TMS320C80, an MVP (multimedia video processor), which has one master processor and four slave processors. The experimental results show that the proposed algorithm gives almost a linear speedup for parallel ray tracing of a complex image.

1. INTRODUCTION

Ray tracing is widely recognized as a powerful, effective technique for generating realistic 3D images in such fields as animations, movies, entertainments, and advertisements due to its simplicity of implementation and can modeling of a variety of visual phenomena such as specular reflection, transparency, and shadows. However, despite of the continuous improvements in reduction of the computation complexity required to generate an image, the ray tracing algorithm remains as a highly time consuming task[1]. Recent researches[2-9] to solve this problem involve the application of parallel processing by having a number of processing elements working on the same problem simultaneously. The independence of the computation for rays, traced through an environment in image generation, makes the ray tracing algorithm an obvious candidate for parallel processing.

The division of a computation among the processors in a multiprocessor system can be broadly categorized according to the nature of parallelism that they exploit. Data-oriented (with ray dataflow) parallel implementation[3] based on message passing partitions the object space to each processor so that ray messages are communicated between processors as they traverse the 3D space. In this case, the model database can be divided in a straightforward way between processors, but balanced distribution of the load between processors is difficult because the relationship between the computation and data are unknown prior to execution. On the other hand, pixel-oriented (without ray dataflow) parallel implementation[4-7] partitions image space to each processor. A sequential process running in each processor is in charge of the computing of a subset of pixels determined according to the processor's load. In this case, load balancing can be achieved easily by using both the static and dynamic technique, but an efficient division of the database between the processors while maintaining data coherency is difficult to determine.

In this paper, we present a pixel-oriented parallel ray tracing algorithm on hybrid parallel architecture with processor farm model to speed up the ray tracing. Hybrid parallel architecture is used in which reconfiguration for local and virtual shared memory is made through a crossbar network with local global bus. Although the database is divided into local memory of each processor, tightly-coupled architecture can maintain data coherency while sharing the database of other processors. Besides, the architecture reduces interprocessor communication by using a virtual shared memory of tightly-coupled architecture through a crossbar network. The proposed algorithm is implemented on TMS320C80 which has one master processor and four slave processors. Speedup, efficiency, and scalability of the proposed parallel architecture are examined.

2. PARALLEL RAY TRACING ON PARALLEL ARCHITECTURE

2.1. Parallel Ray Tracing Algorithm

Ray Tracing algorithm is intrinsically parallel because rays, shot from the viewpoint through each pixel in Figure 1, are calculated independently. Parallel ray tracing algorithm in a multiprocessor system can be broadly categorized according to the nature of parallelism that they exploit.

Data-oriented parallel implementation partitions database into subdomains, each associated with a processor, and assign the computation to the processor that owns the data involved in the computation. If a computation requires other data not located at the processor, the processor sends it to the relevant processors by messages. In this case, balanced distribution of the load between processors is difficult.

On the other hand, pixel-oriented parallel implementation partitions image space to each processor. A sequential process running in each processor is in charge of the computing of a subset of pixels determined according to the processor's load. In this case, load balancing can be achieved easily by using both the static and dynamic techniques, but an efficient division of the database between the processors while maintaining data coherency is difficult to determine.

Data coherency implies that rays traced through adjacent pixels will traverse similar regions of space, and give rise to references to similar subset of object database. When a partition of object database is performed, maintaining of data coherency should be considered.

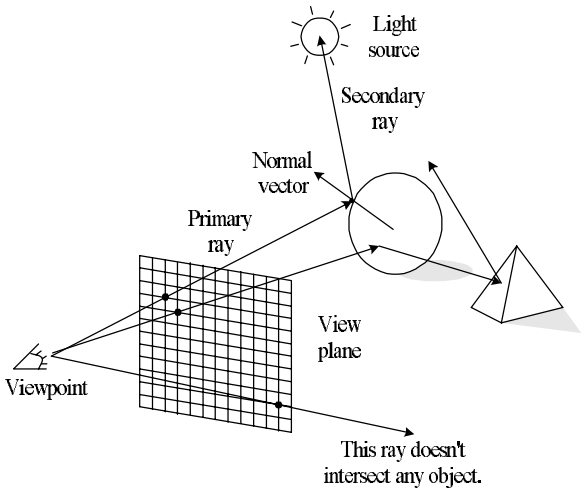

Figure 1. Trajectory of ray tracing.

2.2. Parallel Architecture for Parallel Ray Tracing

The simplest way to implement parallel ray tracing is to duplicate the entire object database in the local memory of each processor. However, this method is wasteful and the limited size of each processor's local memory prohibits its use for rendering complex scenes. An alternative approach is to emulate a shared memory. Green and Paddon[6] used a virtual memory to store the database and a cache mechanism to reduce communication with other processor to maintain data coherency. Green and Paddon located the virtual memory in the host. This can increase the communication cost between host and nodes, and reduce efficiency.

Figure 2 shows a hybrid parallel architecture. L presents local port for accessing its own local memory and G presents global port for accessing local memory of other processors.

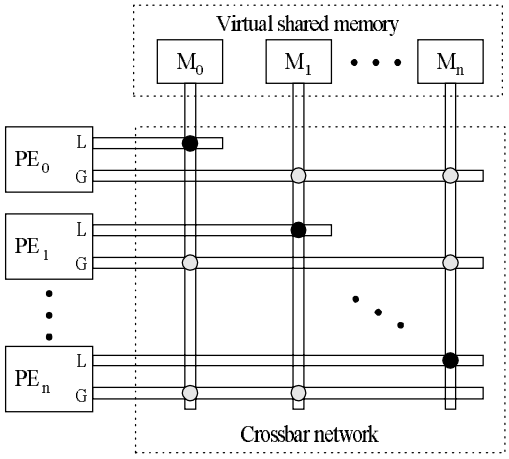

Figure. 2. Hybrid parallel architecture.

In the hybrid parallel architecture, each processor has its dedicated local memory, so it has characteristics of the loosely-coupled architecture. But they also can access other processor's local memory through the reconfigurable crossbar network, so they have the characteristics of both of the loosely-and tightly-coupled architectures concurrently. So, by implementing the ray tracing algorithm on the hybrid parallel architecture, data coherency can be maintained while the database of other processors can be shared. Besides, the architecture reduces interprocessor communication overhead by using a virtual shared memory of tightly-coupled architecture through a crossbar network.

3. PARALLEL RAY TRACING ON HYBRID PARALLEL ARCHITECTURE

Hybrid parallel architecture with processor farm model is shown in Figure 3. A root processor check the status of each node whether it is busy or idle, then dynamically allocates load to idle nodes and collects results from finished nodes.

Each node performs two processes. One is work process (WP) and the other is database manager (DM). The WP performs ray tracing computation and is said to be in one of the two states, busy or idle. The DM services database references of WP during the course of ray tracing and minimize the average access time to object database by each node in loosely-coupled architecture. As DM allows each node reference object data in its own local memory as well as local memory of other nodes through a crossbar network, and therefore reduces the data communication time between the processors in dynamic load balancing while maintaining data coherency.

Each node has two command buffers, used in ping-pong fashion. These enable nodes to immediately perform the next job in the other command buffer instead of waiting for the job allocated by root processor.

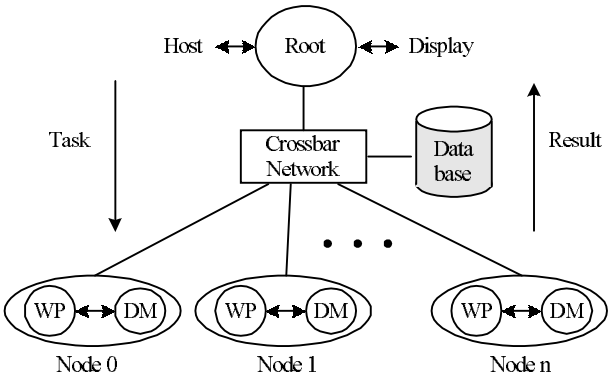

Figure. 3. Overall system architecture.

4. SIMULATION RESULTS

The proposed parallel ray tracing algorithm is implemented on Griffin DSP board with a TMS320C80 and 8Mbytes of RAM. The TMS320C80 contains five processors: a master processor (MP) and four parallel processors (PP). The MP manages processors and has floating-point unit. Each PP is a DSP with a fixed-point processing unit. Each of the five processors has its own 10Kbytes local memory and can access other processor's local memory in parallel over a crossbar switching network[9].

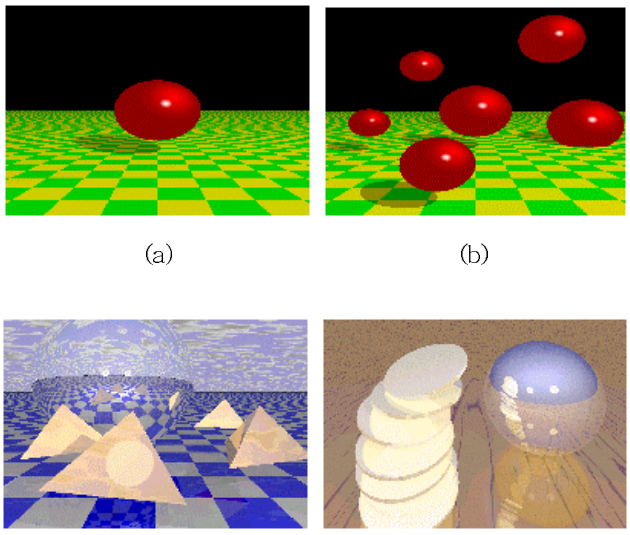

Figure 4. Ray traced images for the experiment (a) single sphere, (b) spheres, (c) tetra, and (d) wealth.

Thus TMS320C80 is compatible with the implementation of a hybrid parallel architecture. Experiments are conducted on four models and the results of the ray tracing are listed in Figure 4. All ray traced images are 1027x768 in resolution.

Table 1 shows parallel ray tracing time for each image with different number of nodes. Figure 5 shows the number of nodes and the speedup for each image. These experimental results show that the proposed algorithm gives almost a linear speedup

Table 1. Ray tracing time and speedup for each image with different number of nodes.

[Unit: sec]

Proce-ssor / Images	TMS320C80				
	Numbers of Nodes				Efficiency (4 nodes)
	1	2	3	4	
S-sphere	206 (1)	112 (1.89)	82 (2.51)	68 (3.03)	75.7%
Spheres	665 (1)	346 (1.92)	237 (2.81)	189 (3.52)	87.9%
Tetra	963 (1)	492 (1.96)	331 (2.91)	251 (3.84)	95.9%
Wealth	1432 (1)	723 (1.98)	486 (2.95)	368 (3.89)	97.2%

( ): Speedup

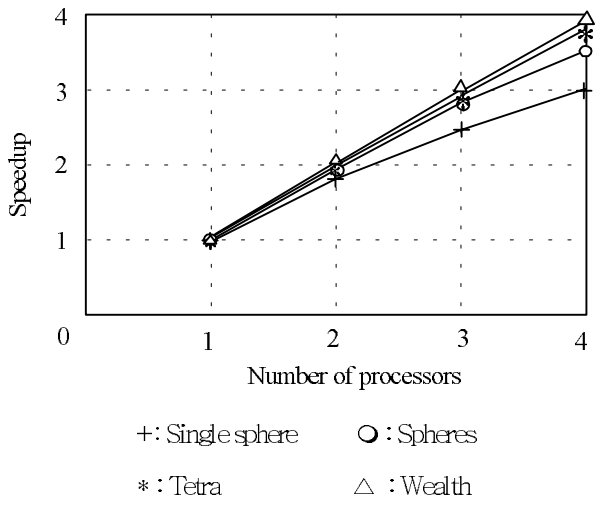

Figure. 5. Speedup for each image.

proportional to the number of nodes in parallel ray tracing of complex images. Single sphere image shows poor speedup because it is so simple an image that the ratio of the interprocessor communication to actual ray tracing computation is the highest among the four images.

To see the effect of the granularity (grain size) to the performance, the grain size is varied from 16 pixels to 16383 pixels in 4-node system. Table 2 shows the results. For small grain size , it is difficult to maintain data coherency and the frequent interprocessor communication is the cause of the overhead. As the grain size increases, maintaining the data coherency becomes easier but the load balancing becomes difficult. If the grain size is one pixel, then we can obtain the best load balance, but the minimal grain size highly increases the communication overhead, leading to worst performance.

Table 2. Ray tracing time for each image with different grain size.

[Unit: sec]

Grain Size (pixels) / Images	16	64	256	1024	4096	16384
S-sphere	96	75	70	68	71	75
Spheres	220	198	193	189	199	212
Tetra	263	252	250	251	273	290
Wealth	379	371	369	368	395	413

5. CONCLUSION

A parallel ray tracing algorithm is implemented on hybrid parallel architecture with processor farm model to speed up the ray tracing. In the hybrid parallel architecture, each processor can access its own local memory as well as local memory of other processors through a crossbar network, and therefore data coherency can be maintained while the database of other processors can efficiently be shared. Besides, the proposed algorithm reduces interprocessor communication by using a virtual shared memory of tightly-coupled architecture through a crossbar network. The experimental results show that the proposed architecture can speed up the ray tracing by 3.89 with four processing elements within the MVP.

REFERENCES

[1] R. A. Hall and D. P. Greenberg, "A testbed for realistic image synthesis," *IEEE CG&A*, Vol. 3, No. 8, pp. 10-20, Nov. 1983.

[2] S. Gaudet, R. Hobson, P. Chilka, and T. Calvert, "Multiprocessor experiment for high-speed ray tracing," *ACM Transactions on Graphics*, Vol. 7, No. 3, pp. 151-179, July 1988.

[3] P. Pitot, "The voxar project," *IEEE CG&A*, Vol. 13, No. 1, pp. 27-33, Jan. 1993.

[4] D. Badouel, K. Bouatouch, and T. Priol, "Distributing data and control for ray tracing in parallel," *IEEE CG&A*, Vol. 14, No. 4, pp. 69-76, July 1994.

[5] S. Horiguchi, A. Katahira, and T. Nakada, "Parallel processing of incremental ray tracing on a multiprocessor workstation," *Proceedings of the International Conference on Parallel Processing*, pp. 192-196, 1991.

[6] S. A. Green and D. J. Paddon, "Exploiting coherence for multiprocessor ray tracing," IEEE CG&A, Vol. 9, No. 6, pp. 12-26, Nov. 1989.

[7] S. Whitman and P. Sadayapan, "Computer graphics rendering on a shared memory multiprocessor," *Proceedings of the International Conference on Parallel Processing*, pp. 197-200, 1991.

[8] S. Green, *Parallel Processing for Computer Graphics*, London: Pitman, 1991.

[9] *TMS320C80 User's Guide*, Texas Instruments, 1995.