Next-Generation Visual Supercomputing using PC Clusters with Volume Graphics Hardware Devices

Shigeru Muraki* Masato Ogata† Kwan-Liu Ma‡ Kenji Koshizuka†
Kagenori Kajihara† Xuezhen Liu† Yasutada Nagano†
Kazuro Shimokawa*

*The National Institute of Advanced Industrial Science and Technology (AIST), Japan
†Mitsubishi Precision Co., Ltd., Japan
‡University of California, Davis, U.S.A.

ABSTRACT

To seek a low-cost, extensible solution for the large-scale data visualization problem, a visual computing system is designed as a result of a collaboration between industry and government research laboratories in Japan, also with participation by researchers in U.S. This scalable system is a commodity PC cluster equipped with the VolumePro 500 volume graphics cards and a specially designed image compositing hardware. Our performance study shows such a system is capable of interactive rendering 512^3 and 1024^3 volume data and highly scalable. In particular, with such a system, simulation and visualization can be performed concurrently which allows scientists to monitor and tune their simulations on the fly. In this paper, both the system and hardware designs are presented.

Keywords: volume rendering, parallel rendering, graphics hardware, visualization, scalable systems, distributed computing systems, high performance I/O

1. INTRODUCTION

Many scientific and medical applications require the capability to visualize volumetric data sets. While realtime rendering of 256^3 volume data can be achieved by using either texture hardware [1] or special volume rendering hardware like the *VolumePro 500* card (TeraRecon, Inc.) [2], large-scale volume rendering (e.g. for 512^3 volume data or larger) must utilize a parallel computer. Parallel software rendering has proved to be scalable [3, 4, 5, 6, 7] but for rendering a large scale data set it must utilize a very large number of processors to achieve interactive rates. More recently, the use of multiprocessor, multipipe graphics supercomputer for interactive rendering of large-scale volume data has been demonstrated [8], but to most scientific researchers it is not an affordable solution.

In this paper, we present the design of a visual computing system and preliminary experimental results on the prototype system that we have built. The proposed system is a commodity PC cluster with each slave PC equipped with a VolumePro 500 card and an *GeForce 2* card (NVIDIA Corp.). We have also designed and built a special *image compositing hardware* which allow us to expand the system for interactive rendering extremely large-scale volume such as 1024^3 to 2048^3 volume data.

While such a visual computing system can be utilized for post-processing volume rendering, our main objective is to use it for runtime visualization of large-scale *volumetric simulation*. In a time-varying neuron excitement simulation, for example, while the CPUs of the cluster are responsible for the simulation calculations, the volume rendering hardware can continuously produce visualization of the current excitement propagation.

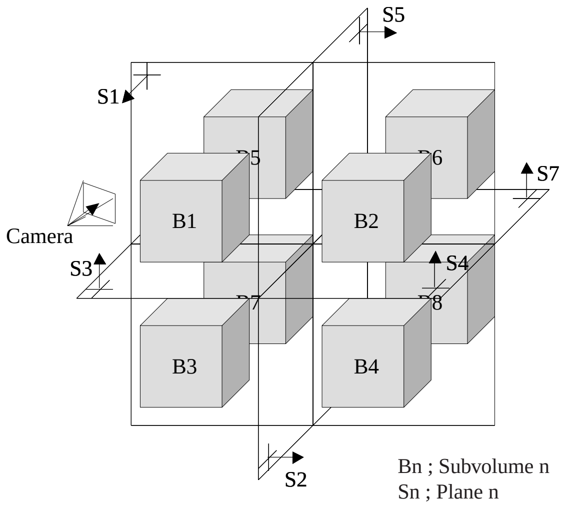
The proposed system is highly scalable and will become low cost as volume graphics cards are becoming commodity parts. Most importantly, by implementing a high-performance, low-cost visual computing capability into a commodity PC cluster, we are able to eliminate the data transfer bottleneck in a conventional runtime visualization setting. This visual computing capability will change the way we conduct scientific research and engineering design. In this paper, we report results of our prototype developement of the *visual computing cluster*.

2. PARALLEL VOLUME RENDERING

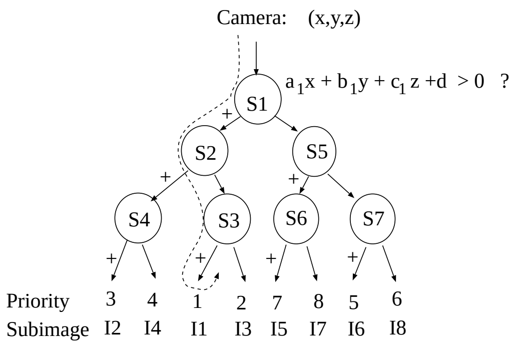
Volume rendering is a very powerful technique for visualizing voxel data typically describing physical phenomena in a spatial domain. It can display more information in a single visualization than techniques such as isosurface or slicing. It is more flexible because it can also be used to approximate isosurfaces and cut-planes, or a mixture of them. Most importantly, direct volume rendering is particularly effective for visualizing fine features and those features that cannot be defined analytically.

The basic volume rendering algorithm steps through the volume

*Collaborative Research Team of Volume Graphics, AIST Tokyo Waterfront (Room 428), 2-41-6 Aomi, Koto-ku, Tokyo, 135-0064 Japan {s-muraki@aist.go.jp}
†Mitsubishi Precision Co., Ltd., 345 Kamimachiya, Kamakura-shi, 247-8505 Japan {ogata@mpcnet.co.jp}
‡Department of Computer Science, 2063 Engineering II, University of California-Davis, One Shields Avenue, Davis, CA 95616-8562, USA {ma@cs.ucdavis.edu}



(a) Subdivide a volume data into subvolumes (B_n).



(b) Priority assignment by using binary-tree.

Figure 1: Binary tree image composition method.

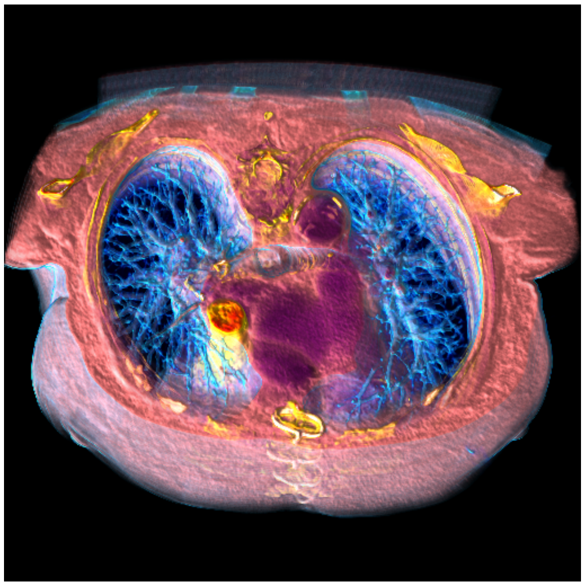


Figure 2: CT lung: 512^3 voluem data (rendered in 512×512)

Table 1: Image composition time (ms) for $n = 2$.

# of pixels	communication		blending
	100BASE	Myrinet	computation
256×256	62	2.5	35
512×512	230	9.7	143
768×768	514	21.5	323

data, integrating color and opacity along a ray for each pixel. A straightforward way to parallelize volume rendering is to partition and distribute the volume evenly among the participating processors. Each processor performs rendering of local subvolume independent of other processors. This local rendering step generates a partial image at each processor, and is followed by a global composition process which results in the final, complete image by merging all partial images in a front-to-back or back-to-front depth order. This is an *object-space parallel algorithm*. This algorithm works correctly because the composition operator is associative [10] which allows us to break each ray into segments, compute them separately, and finally compose them to derive the corresponding pixel value. The global composition process requires communication between processors. Therefore, the key to efficient parallel volume rendering is to control the computing and communication cost required by the image composition task.

The other way to parallelize volume rendering is to partition and distribute the image space among participating processors — an *image-space parallel algorithm*. Each ray is computed by a single processor. Communication is required before the resampling step to move data to the processor responsible for the corresponding projected image area. Consequently, this algorithm is more commonly used for shared-memory parallel architectures due to its high communication requirement. When memory space is abundant, replicating the whole volume data among processors can eliminate the communication cost [11]; however, this approach is not feasible for rendering large-scale data. In our design, we use the object-space parallel volume rendering algorithm. As shown in Fig 1(a), a subvolume (B_n) is distributed to each processors by the binary object-space partitioning. A depth priority value is assigned for each subvolume by considering the spatial relation between the camera position and the subdivision surface (S_n). Volume rendering and image composition of subimage (I_n) is carried out as shown in Fig. 1(b). That is, pairs of subimages are composited concurrently through $log(n)$ stages where n is the number of processors. A software implementation of this binary-tree image composition ought to be inefficient. The problem is that at each phase of composition, half of the processors become idle. Finally, at the top of the compositing tree, only one processor is active, doing the final composite for the entire image. Table 1 compares the communication and computing time for compositing two subimages ($n = 2$) using our PC cluster (Pentium III 800 MHz CPU). An image of the test data set, 512^3 lung CT volume data, is displayed in Fig. 2. Theoretically, the numbers in Table 1 increase in proportion to $log(n)$. It is clear that the communication time of 100BASE-TX Ethernet is too slow to achieve the interactive rendering speed. Even if we use a high-speed network, Myrinet (1.28 Gbits/s) [12], the compositing operation is sufficiently slow which hampers interactive rendering.

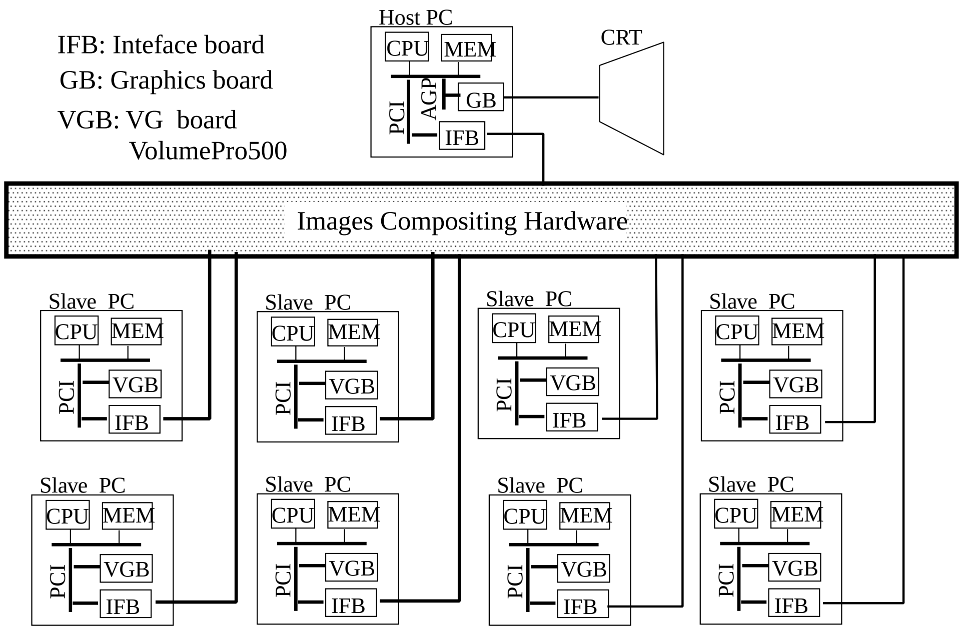


Figure 3: Visual computing cluster structure

```
//========= Host PC Process =========//
ProcessServer()
{
  Load_and _Send_Subvolumes() ;
  for (;;) {                                   // Start of Loop
    Broadcast_Camera_Info() ;
    MPI_Barrier() ;                            // Synchronization
    Receive_Image_from_IFB() ;                 // Composition Start
    Display_Image_by_GB() ;
  }                                            // End of Loop
}

//========= Slave PC Process ========//
ProcessNode()
{
  Recieve_Subvolume();
  for (;;) {                                   // Start of Loop
    Recieve_Camera_Info() ;
    Compute_Priority_from_Camera_Info() ;
    Render_Subvolume_by_VGB() ;
    MPI_Barrier() ;                            // Synchronization
    Send_Image_To_IFB() ;                      // Composition Start
  }                                            // End of Loop
}
```

Figure 4: Pseudo-code of control program in the prototype system.

In this way, when running on a massively parallel computer with a large number of processors, composition would become a serious bottleneck. As a result, several parallel image compositing algorithms [5, 13] have been introduced for scalable parallel software volume rendering. For hardware implementation, however, this simple binary-tree approach is in fact desirable.

3. PROTOTYPE SYSTEM

3.1 Visual Computing Cluster

To realize steerable volumetric simulation, we propose a PC cluster system with a powerful volume graphics functionality (visual computing cluster) [14]. This system implements the image composition parallel rendering method by using commercially available volume graphics cards (VolumePro 500) to achieve realtime rendering, as well as employing a specially developed image compositing hardware to eliminate the computation bottleneck of the image composition.

Figure 3 shows the structure of the visual computing cluster. The host PC divides a volume data among eight slave PCs to perform volume renderings in parallel. The subimage generated by each volume rendering engine (VGB) is sent to the image compositing hardware via the interface board (IFB) on the PCI bus, and merged with the other subimages based on their depth priorities. The resultant image is displayed by writing it into the frame buffer of the graphics board (GB) via the PCI interface board (IFB) of the host PC. Repeating this procedure generates an animation. Figure 4 shows the pseudo-code of the animation generation.

3.2 Image Compositing Hardware

There were special image compositing hardware devices employing either a pipeline [15] or bus [16] architecture, but they were designed for polygon rendering, in which composition may be done in arbitrary order by using the z-buffer technique. To support the depth order composition for the translucent volume rendering, our compositing hardware faithfully implements the binary-tree subimage composition process explained in Fig. 1 into a hardware.

Figure 5 shows the block diagram of the image compositing hardware. Slave PCs send the priority and color (R, G, B, A) information of the subimages into the input channel of the compositing hardware (CH. n) as a command sequence. The priority information is sent prior to the frame data (Fig. 7) and used to control the selector (SEL). According to the size of the priority values, the selector decides which one is superimposed over the other. The

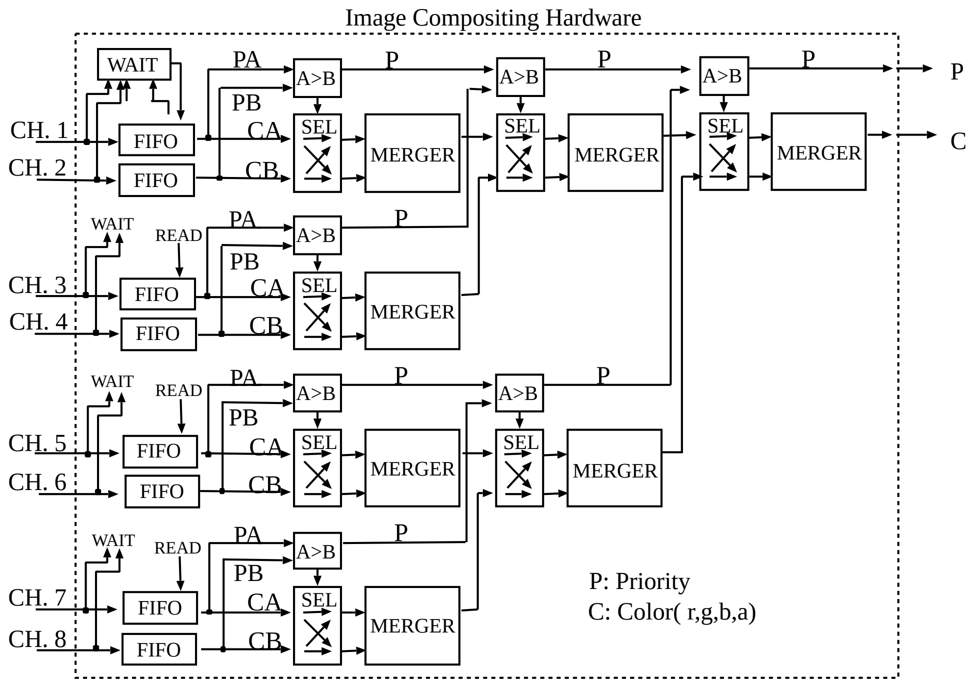


Figure 5: Block diagram of the image compositing hardware.

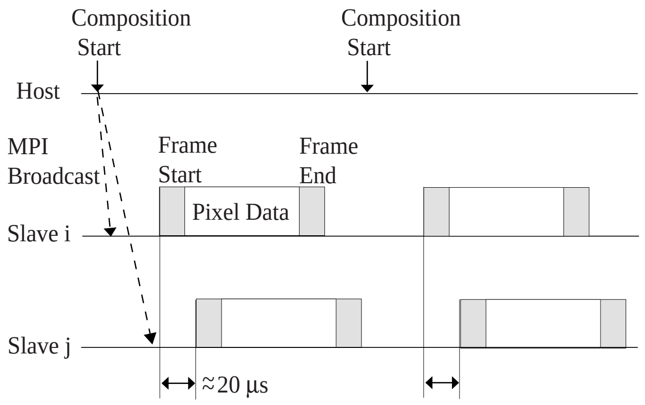


Figure 6: Time delays between input channels of the compositing hardware.

merger (MERGER) performs the compositing operation of all color channels (R, G, B). Since each input channel can be enabled or disabled by the command sequence, the number of slave PCs is variable. The merged image is sent to the host PC from the output channel of the compositing hardware.

This image composition requires the synchronization of sending and receiving timings of images. As shown in Fig. 4, we are using *MPI_Barrier()* command for this synchronization; however, it causes time discrepancy around 20 micro-seconds for each PC in case of Myrinet (Fig. 6). To compensate this timing mismatch, we put a FIFO (256K depth, 36 bits width) for each input channel as shown in Fig. 5.

3.3 Communication Architecture

The connection between the interface board and the compositing hardware uses LVDS (Low Voltage Differential Signaling), which converts parallel data into serial to avoid clock skew inherent to the high-speed data transfer. We used commercially available chips (National Semiconductor, DS90CR483/484) for the conversion between the CMOS/TTL data and LVDS. Since the LVDS assigns 6 bits data stream in a conductor, we can transfer up to 36 bits data by a cable having 7 conductors (six LVDS data streams and one clock channel). To perform this serial transmission, we converted the base frequency of the PCI bus clock (33 MHz) to 33×6 MHz. The transmission rate for a conductor is 198 Mbits/s and we can achieve 1.19 Gbits/s using 6 conductors, which is sufficiently fast to send the data via PCI whose throughput is 1.064 Gbits/s. The compositing hardware receives the serial data and converts it back to parallel.

Figure 8 shows the component parts of the visual computing cluster prototype. Figure 8(a) shows the input interface board, which is inserted in the PCI slot of the host computer. Figures 8 (b) display the inside of the compositing hardware. The daughter board does the compositing operation of two subimages, and a motherboard and four daughter boards construct the compositing hardware for eight slave PCs. Figure 8(c) shows the outside of the compositing hardware, an LVDS cable and an interface board of a slave PC.

The image compositing hardware consists of a 21-stage pipeline of 36 bits band-width. Table 2 shows the assignment of command and status signals to the 36 bits data. The most of the circuits of the interface boards and the compositing hardware were implemented by using Field Programmable Gate Array (FPGA) of ALTERA, Co. Ltd., whose logics were reprogrammable by using VHDL language. Figure 9 shows the outside look of our prototype system, and the system specification is given in Table 3.

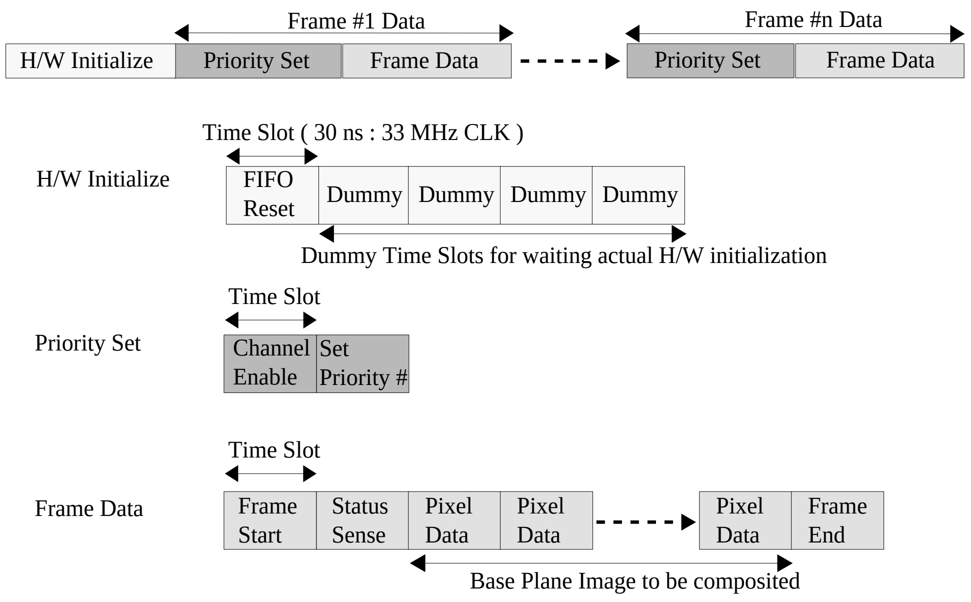


Figure 7: Command sequences for the compositing hardware.

Table 2: Assignment of 36 bits Data

Bit		35· · · 32	31· · · 24	23· · · 16	15· · · 8	7· · · 0
Command	Dummy Command	0	-	-	-	-
	Pixel Data	1	Red	Green	Blue	Alpha
	Channel Enable	2	1	-	-	-
	Channel Disable	2	2	-	-	-
	Set Priority	2	4	-	Priority (9 bits)	
	FIFO Reset	2	8	-	-	-
	Frame Start	2	16	-	-	-
	Frame End	2	32	-	-	-
Status	Channel not Ready	4	1	-	-	-
	Pixel Count Mismatch	4	2	-	-	-
	Illegal Command	4	4	-	-	-

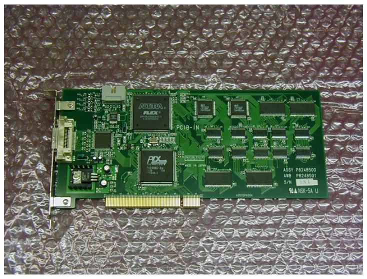
4. PERFORMANCE STUDY

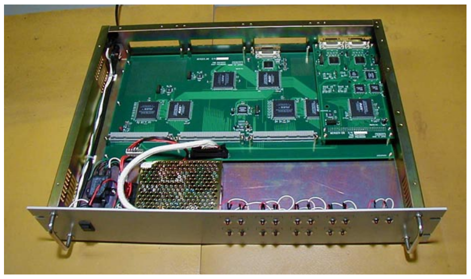
We evaluated the rendering performance of the visual computing cluster by using a CT lung dataset (512^3 volume data). We wrote all of our test programs in C++ language using MPICH-SCore. SCore [17] is a Linux based operating system developed at the Parallel and Distributed System Software Laboratory of the Real World Computing Partnership (RWCP) in Japan, which employs:

- a user-level zero-copy message transfer mechanism between nodes and one copy message transfer mechanism within a node based on high performance communication facility called PM,
- a high-performance MPI implementation called MPICH-SCore that integrates both zero-copy message transfer and message passing facilities in order to maximize performance,
- and a multi-user environment using gang scheduling without degrading the communication performance realized by an operating system daemon called SCore-D.

Table 4 shows processing times of sub-processes, i.e., hardware subvolume rendering (H/W Rend.), image composition (H/W Composit.) and image drawing (Draw Pixel), and frame rates for three different image resolutions. Each number is an average of 100 flames by varying the view angle. We achieved an interactive frame rate of 16.1 Hz for 256×256 image resolution. The hardware image composition took 5 ms for this image, however, this is spent for the memory access via the PCI bus and the latency of the image compositing hardware is only 0.64 micro second (21 clocks of 33MHz). Table 4 shows that the hardware rendering time have rooms for further improvement. This problem is caused by the pixel format conversion between VolumePro 500 and the compositing hardware. We believe that the hardware rendering time can be reduced as the order of DrawPixel time by tuning up the firmware of the compositing hardware.

We decided the size of FIFOs of the compositing hardware sufficiently large to compensate the synchronization mismatch, and confirmed that the image compositing hardware worked both cases of Myrinet and 100 BASE-TX Ethernet. Therefore, we can build low-cost and high performance visual computing cluster without


(a) Interface board for host PC.


(b) Inside of the image compositing hardware.

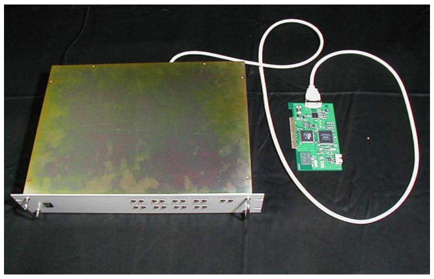

(c) Image compositing hardware, LVDS cable and interface board for slave PC.

Figure 8: Visual computing cluster parts

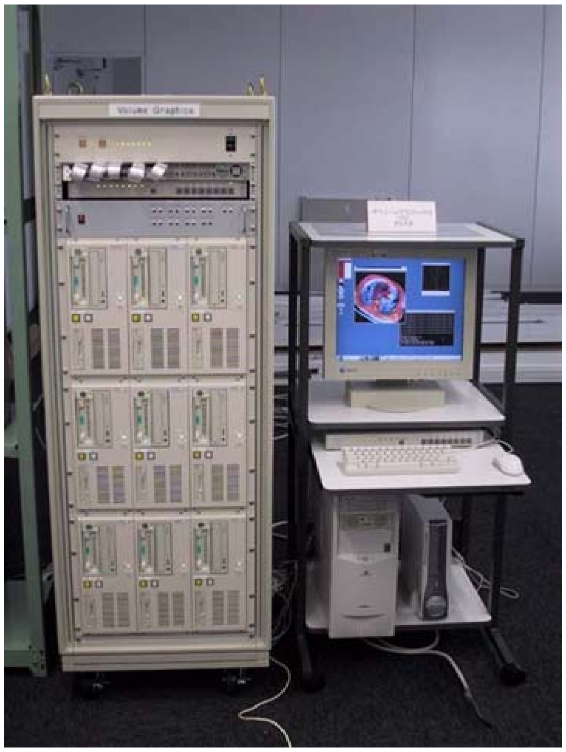


Figure 9: The prototype of the visual computing cluster.

Table 3: The specification of the visual computing cluster

CPU	Pentium III 800 MHz $\times 9$
Memory	512MB $\times 9$
Video Card	nVIDIA GeForce 2 $\times 9$
VG Card	TeraRecon VolumePro 500 $\times 8$
Network	Myrinet (1.28 Gbits/sec), 100BASE-TX
OS	SCore 3.3 (Real World Computing Partnership)
	Linux kernel 2.2.14
Graphics API	XFree86 4.0.3, GtkGLarea-1.2.2, nVIDIA OpenGL driver for Linux

Table 4: Rendering performance of visual computing cluster.

# of pixels	Rendering time (ms)			Frame Rate (Hz)
	H/W Rend.	H/W Composit.	Draw Pixel	
256×256	52	5.0	5.1	16.1
512×512	80	18.1	19.5	8.5
768×768	139	28.0	43.4	4.8

employing Myrinet.

5. CONCLUSIONS

In this paper, we present the design and the performance study of our visual computing cluster. The prototype system shows the performance of rendering up to 512^3 volume data at interactive rate by using eight VolumePro 500 boards in parallel.

The latency of the image compositing hardware is 0.64 micro second, which is sufficiently small comparing to the volume rendering time of each PC. Therefore, a hierarchical connection of the compositing hardware allows the massively parallel processing without reducing the visualization frame rate (Fig. 10). Figure 11 plots the expected latency for up to 32768 processors.

Current rendering speed is not sufficiently fast for the large screen visualization; however, this problem will be solved by tuning up the firmware of the image compositing hardware. Because of the bandwidth limitation of the PCI bus, it takes the compositing hardware 61 *ns* to process each pixel. As a result, our system can render 512×512 image at about 60 frames per second, or 1024×1024 image at about 15 frames per second for 512^3 volume data. The performance would become a few times better as the next generation 64 bits 66 MHz PCI becomes available. We are also planning to use a pipeline technique to overlap sub-processes (i.e. rendering, composition and drawing) for further speed-up of the animation generation.

For postprocessing rendering, our target application is in medicine. Recently, very high resolution volume data sets, over 2048^3 volume

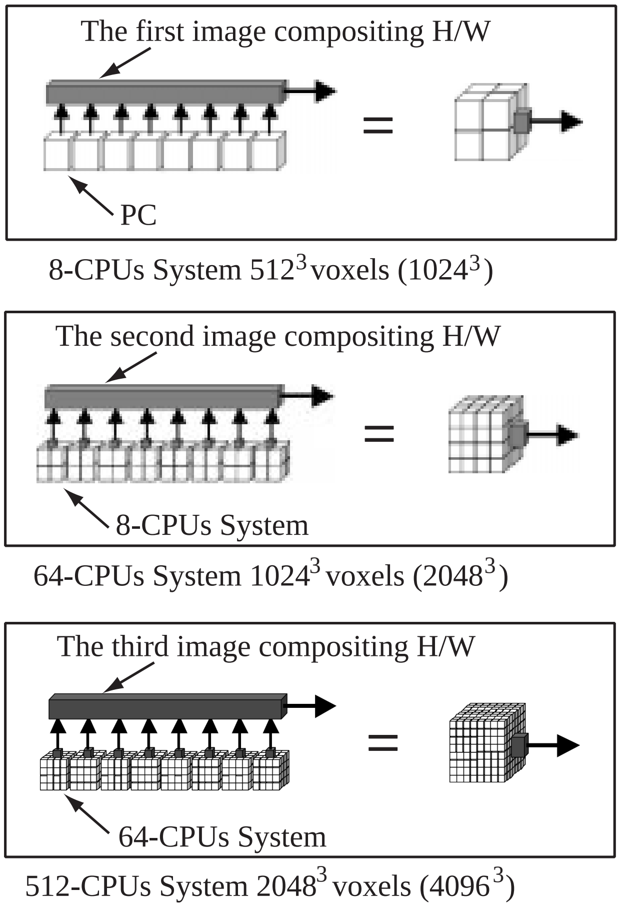

The first image compositing H/W
=
PC
8-CPUs System 512^3 voxels (1024^3)
The second image compositing H/W
=
8-CPUs System
64-CPUs System 1024^3 voxels (2048^3)
The third image compositing H/W
=
64-CPUs System
512-CPUs System 2048^3 voxels (4096^3)

Figure 10: Hierarchical connection of image compositing hardware. Parenthesis show the values of when using next generation VolumePro 1000.

data, have been produced by using techniques such as corn-beam CT. There is no good way to visualize such huge datasets. Our system is promising for such a demanding application. Using the proposed cascading, a 64-CPUs system should allow for interactive medical image processing and visualization of 1024^3 or 2048^3 volume data.

For runtime visualization of 3-d simulation like the 3-d cellular automata, our system is even more attractive. Since the latency of the cascade connection of the compositing hardware is negligible, the runtime visualization of very complicated chemical system like a human brain simulation is possible. Figure 12 is the nerve excitement simulation on a volume data of the lateral geniculate nuclei neuron of a rat ($386\times257\times197$ volume data) obtained by a confocal laser-scanning microscope [18]. The state transition of each voxel is determined by the states of neighboring 27 voxels by using a 3D cellular automata, which approximates the Hodgkin-Huxley equation [1]. The 512-CPUs system of Fig. 10 will make the simulation and visualization of cerebral tissue of 1 mm^3 size possible at an interactive rate. Conducting a volumetric simulation at this scale would make possible understanding of the effectiveness of medicine or elucidation of the information processing mechanism

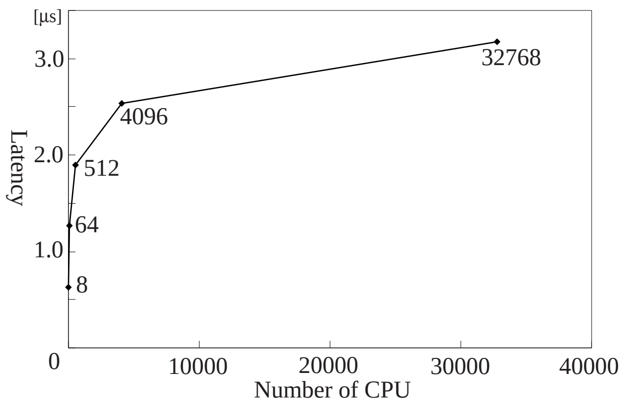


Figure 11: Expected composition latency.

of human brain.

Our prototype system uses Myrinet and VolumePro 500 card which are considerably expensive. However, since the volumetric simulations only need communication between neighboring processors, we can avoid using Myrinet by putting simple inter processor communication functionality into the image compositing hardware.

It is also possible to avoid using VolumePro 500 by using texture mapping hardware of polygon graphics cards for the low-cost volume rendering. We are presently investigating the use of GeForce 2 cards along with our compositing hardware for parallel volume visualization. While the image quality is not as good as VolumePro 500's, this low-cost alternative is very attractive.

Nonetheless, it is clear volume graphics cards will be more widely adopted, and thus their price will drop significantly which makes our visual computing design an affordable solution regardless of the graphics hardware used.

Increasingly, scientists become less dependent of centralized supercomputers, and are able to build their own volume simulation engine at low cost. Using our design, they can put together a scalable system supporting both simulation and visualization. Using such a personal visual computing system would help scientists achieve higher productivity, obtain profound insights, and reach new discoveries sooner.

6. ACKNOWLEDGMENTS

We thank Dr. Yutaka Ishikawa of the Real World Computing Partnership (RWCP) for his instruction on the PC cluster techniques. This research is performed by the Collaborative Research Team of Volume Graphics, the National Institute of Advanced Industrial Science and Technology (AIST), and University of California at Davis.

7. REFERENCES

[1] Cabral, B., Cam, N., and Foran, J., Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware, *Proc. ACM Symposium on Volume Visualization*, October 1994.

[2] Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H. and Seiler, L., The VolumePro Real-Time Ray-Casting System, *Proc.*

[1] A differential equation which models the dynamics of sodium (Na) and potassium (K) ion channels on a nerve cell membrane.

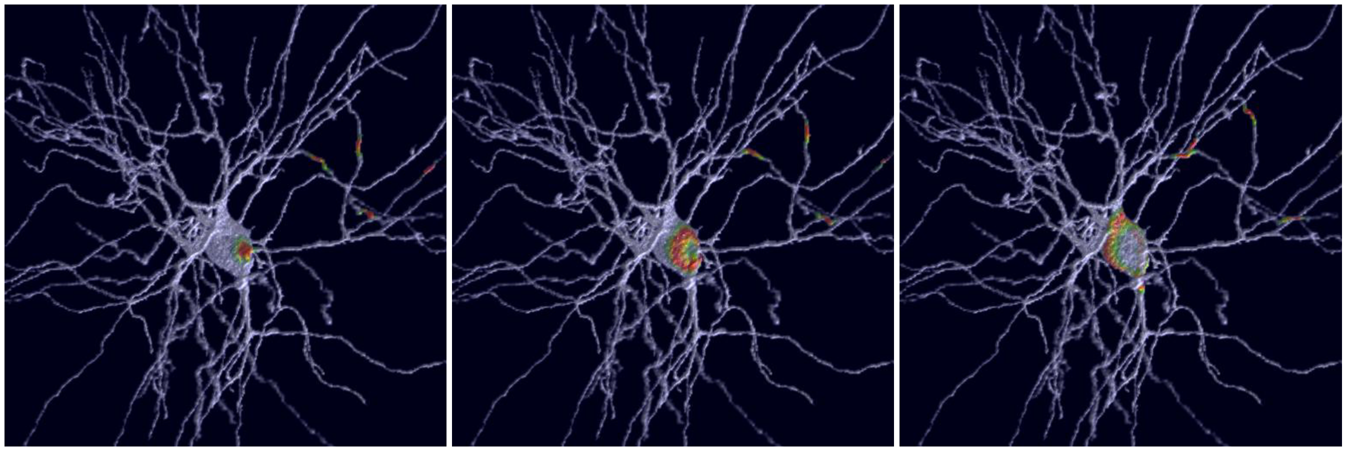


Figure 12: Nerve excitement simulation

SIGGRAPH 99, pp.251-260, August 1999.

[3] Hsu, W. H., Segmented Ray Casting for Data Parallel Volume Rendering, *Proc. Parallel Rendering Symposium*, pp. 7-14, October 1993.

[4] Camahort, E. and Chakravarty, I., Integrating Volume Data Analysis and Rendering on Distributed Memory Architectures, *Proc. Parallel Rendering Symposium*, pp.89-96, October 1993.

[5] Ma, K.-L., Painter, J.S., Hansen, C.D. and Krog, M.F., Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE CG&A*, vol.14, no.4, pp.59-68, July 1994.

[6] Lacroute, P., Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization, *Proc. 1995 Parallel Rendering Symposium*, pp. 15-22, October 1995.

[7] Palmer, M. E., Taylor, S., Totty, B., Exploiting Deep Parallel Memory Hierarchies for Ray Casting Volume Rendering, *Proc. 1997 Parallel Rendering Symposium*, pp. 15-22, October 1997.

[8] Kniss, J., McCormick, P., McPherson, A., Ahrens, J., Painter, J., and Keahey, A., Interactive Texture-Based Volume Rendering for Large Data sets, *IEEE Computer Graphics & Applications*, vol. 21, no. 4, pp. 52-61, July 2001.

[9] Molnar, S., Cox, M., Ellsworth, D. and Fuchs, H., A Sorting Classification of Parallel Rendering, *IEEE CG&A*, vol.14, no.4, pp.23-32, July 1994.

[10] Porter, T. and Duff, T., Compositing Digital Images, *Computer Graphics (Proc. SIGGRAPH 84)*, vol. 18, no. 3, pp.253-259, July 1984.

[11] Knittel, G., The UltraVis System, *Proc. IEEE Volume Visualization and Graphics Symposium*, pp.71-79, October 2000.

[12] Boden, N. J., Cohen, D, Felderman, R. E., Klauik, R. E., Seitz, C. L., Seizovic, J. N. and Su, W-K, Myrinet: A Gigabit-per-Second Local Area Network, *IEEE MICRO*, vol.15, pp.29-36, February 1995.

[13] Lee, T.-Y., Raghavendra, C. S., and Nicholas, J. N., Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers, *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 202-217, September 1996.

[14] Muraki, S., Shimokawa, K., Ogata, M., Kajihara, K., Ma, K.-L. and Ishikawa,Y., VG Cluster: Large Scale Visual Computing System for Volumetric Simulations, *SC2000 Research Gems(Poster)*, November 2000.

[15] Molnar, S., Eyles, J. and Poulton, J., PixelFlow: High-Speed Rendering Using Image Composition, *Computer Graphics (Proc. SIGGRAPH 92)*, vol.26, no.2 , pp. 31-240, July 1992.

[16] Heirich, A. and Moll, L., Scalable Distributed Visualization Using Off-the-Shelf Components, *Proc. 1999 IEEE Parallel Visualization and Graphics Symposium*, pp.55-59, October 1999.

[17] Takahashi, T., Sumimoto, S., Hori, A., Harada, H. and Ishikawa, Y., PM2: High Performance Communication Middleware for Heterogeneous Network Environments, *SC2000*, November 2000.

[18] Shimokawa, K. and Muraki, S., A Study on Spatial and Temporal Visual Simulation of Nerve Excitement Propagation, *Proc. IJCNN 2000*, July 2000.