IPSJ Transactions on Advanced Computing Systems Vol. 2 No. 2 122–130 (July 2009)

Regular Paper

Performance Evaluation of Large-Scale Parallel Image Compositing on a T2K Open Supercomputer

Jorji Nonaka,[†1] Kenji Ono[†1] and Hideo Miyachi[†2]

This paper presents a performance evaluation of large-scale parallel image compositing on a T2K Open Supercomputer. Traditional image compositing algorithms were not primarily designed for exploiting the combined message passing and the shared address space parallelism provided by systems such as T2K Open Supercomputer. In this study, we investigate the *Binary-Swap* image compositing method because of its promising potential for scalability. We propose some improvements to the *Binary-Swap* method aiming to fully exploit the hybrid programming model. We obtained encouraging results from the performance evaluation conducted on *Todai Combined Cluster*, a T2K Open Supercomputer at the University of Tokyo. The proposed improvements have also shown a high potential to tackle the large-scale image compositing problem on leading-edge HPC systems where an ever increasing number of processing cores is involved.

1. Introduction

Scientific computing and visualization have played an important role in computer-aided scientific discovery using high performance computing (HPC) capabilities. The size and complexity of data sets generated from numerical simulations have increased at a rate comparable to that of the computational power and the network bandwidth of leading-edge HPC systems. Sort-last parallel rendering [1)] has proven effective when executing large-scale data visualization on such systems since it avoids costly and sometimes prohibitive data transfer between HPC and visualization oriented systems [2)–6)].

Recent trends in modern high performance computing (HPC) system architectures show an increasing use of high speed, high bandwidth interconnect allied with a huge number of computation nodes with multiple processing cores. This trend can be verified in the T2K Open Supercomputer [7)] specification which is employed in three of the most powerful supercomputers in Japan [8)].

In the sort-last method, full resolution images generated from each rendering node are composited together using a parallel image compositing algorithm. Several compositing algorithms have been proposed so far, and are currently grouped into three main categories: *Direct Send* [9),10)], *Parallel Pipeline* [11)], and *Binary-Tree* which includes *Binary-Swap* [12)]. These methods have widely been utilized on small and medium size parallel computing systems where they have proven efficient. However, a straightforward application to T2K Open Supercomputer systems might produce non optimized results.

Traditional parallel image compositing algorithms were not designed with a hybrid programming model in mind. Most of them are designed for pure distributed or pure shared-memory parallel computing systems. Although distributed memory applications can work on systems with full or partial shared-memory address space, a loss in performance might occur when the hybrid programming model is not exploited. In addition, a special attention to the communication cost is required when targeting large-scale computational systems such as T2K Open Supercomputer systems. This is because the communication time usually dominates the overall parallel image compositing time.

In this study, we focused on the Binary-Swap image compositing method which has an $O(n.log_2 n)$ communication cost compared to $O(n.n^{1/3})$, or even $O(n.(n-1))$, of other image compositing methods. A *Flat MPI* version was implemented and a performance analysis was carried out on a T2K Open Supercomputer installed at the University of Tokyo also known as Todai Combined Cluster (hereafter called *Todai T2K*). On this system, the performance degradation was verified when the number of compositing nodes reaches the order of thousands. In order to minimize this performance degradation, we investigated some improvements to the Binary-Swap method to fully exploit the hybrid programming model on Todai T2K. The improvements include the combination of different parallel image compositing methods.

The remainder of this paper is organized as follows. In Section 2, we describe the sort-last image compositing with special attention to the Binary-Swap

†1 Computational Science Research Program, RIKEN
†2 Visualization Division, KGT Inc.

method. In Section 3, we present some improvements to Binary-Swap in order to exploit the hybrid programming model. Experimental results and discussions are presented in Section 4, and we conclude by presenting some future works in Section 5.

2. Sort-Last Image Compositing

The sort-last image compositing method is responsible for the final stage of the rendering pipeline. That is, the full size images generated by the rendering process are composited, or merged, by using alpha blending or z-buffer techniques, in order to produce the final image. In this study, we focused on alpha blending which is more complex than the z-buffer method since it requires a correct ordering of the entire set of images during the compositing process. In addition, since it handles semi-transparent images we can apply it to volume-rendered images. **Figures 1** and **2** show some examples of the image compositing methods investigated in this study.

Direct Send and Binary-Swap methods have proven effective on small and medium size parallel systems in the order of tens and sometimes hundreds of compositing nodes [12)–18)]. The Parallel Pipeline method has proven suitable for small size parallel systems. In fact, this method has been used on commercial parallel visualization applications such as *AVS/Express PST* (Parallel Support Toolkit) [19)] and *CEI Ensight DR* (Distributed Rendering) [20)].

Recently, Direct Send has received an increasing attention and several optimizations have been proposed so far[21)–23)]. Among them, the Scheduled Linear Image Composition method, or SLIC [21)] for short, emerged as the prime candidate for large-scale data visualization. SLIC generates an on-the-fly scheduling aiming to minimize unnecessary data transmission. SLIC has a communication cost in the order of $O(n.n^{1/3})$ and this makes the number of required message transmissions comparable to that of Binary-Swap, which has a cost in the order of $O(n.log_2 n)$, when up to 1,024 compositing nodes are used. However, in the order of tens of thousands of compositing nodes, SLIC might require two times more message transmissions than Binary-Swap. In addition, the on-the-fly scheduling generation on such a number of compositing nodes might influence the computational cost that reduces the gain obtained by the optimized data transmission. The biggest advantage of SLIC is that it is not limited to using a power of two compositing nodes as required by Binary-Swap. Recently, *2- 3 Swap Image Compositing* [18)] has been proposed as a generalization of Binary-Swap to an arbitrary number of compositing nodes. Although it does not suffer from any limitation on the number of nodes, it is reported that a better performance is always obtained when the number of compositing nodes is a power of two. That is, when using the Binary-Swap algorithm.

2.1 Binary-Swap Image Compositing

Binary-Swap can be considered as a highly optimized binary-tree method where the rendering nodes are kept busy as much as possible during the entire image compositing stage. Binary-Swap is perhaps the most used, it has been widely

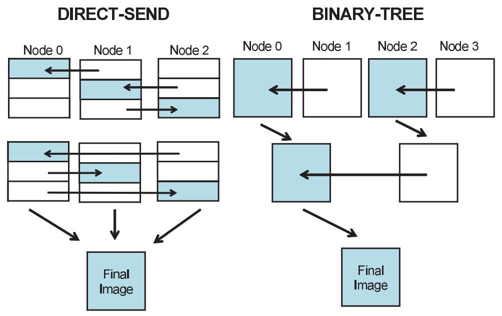
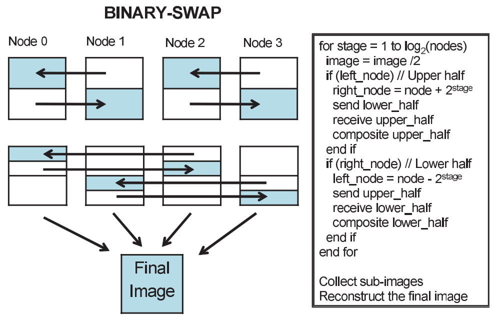


Fig. 1 *Direct-Send* and *Binary-Tree* image compositing methods.



```
for stage = 1 to $log_2$(nodes)
  image = image /2
  if (left_node) // Upper half
    right_node = node + $2^{stage}$
    send lower_half
    receive upper_half
    composite upper_half
  end if
  if (right_node) // Lower half
    left_node = node - $2^{stage}$
    send upper_half
    receive lower_half
    composite lower_half
  end if
end for

Collect sub-images
Reconstruct the final image
```

Fig. 2 *Binary-Swap* image compositing method.

researched and extensive optimization techniques[13)–17)] have been proposed so far.

As shown in Fig. 2, during the Binary-Swap image compositing process, the image is recursively divided into two parts. Half of them is exchanged between pairs of compositing nodes. The other half is then composited with the received image while taking into consideration the correct ordering. The communication distance doubles at each stage, and this linear increase in distance greatly compromises the network traffic, which can generate network contention, when the number of compositing nodes increases. Contrary to the communication distance, the image size becomes smaller and smaller since it diminishes to a half at each stage. At the end, each node possesses $1/n$ of the original image size as the final composited image.

Binary-Swap requires a power of two number of compositing nodes n, and the required number of stages for completion is $log_2 n$. At each stage i, each compositing node will send, receive, and composite $p_{xy}/2^i$ pixels, where p_{xy} is the total number of pixels in the image. Thus the total number of pixels (p_{xy}) that each compositing node needs to send, receive, and composite can be expressed as shown in Eq. (1), where a formula for geometric series is used.

$$\begin{aligned}\sum_{i=1}^{log_2 n} \frac{p_{xy}}{2^i} &= p_{xy} \sum_{i=0}^{log_2 n} \left(\frac{1}{2}\right)^i - p_{xy} \\ &= p_{xy}\left(2 - \frac{2}{2^{1+log_2 n}}\right) - p_{xy} \\ &= p_{xy}\left(1 - \frac{1}{n}\right) \qquad (1)\end{aligned}$$

The final composited image fragments distributed across the compositing nodes have to be gathered and reconstructed at the main compositing node (root node). Since the size of each image fragment is $1/n$, the amount of data to be gathered will be equivalent to the total image size, or p_{xy}. A feasible approach for this step is the use of available MPI collective functions such as *MPI_Gather*. This stage has been ignored for small size parallel systems since seldom influences the compositing performance. However, when the number of nodes increases it has a great potential to become a serious problem.

2.2 Theoretical Performance Analysis

There is a vast and rich literature on the *sort-last image compositing* method, and a detailed theoretical performance analysis for both shared-memory[24)], and distributed memory[25),26)], parallel computing systems can be found. The total time required for the parallel image compositing can be represented as shown in Eq. (2).

$$t_{total} = t_{read} + t_{compose} + t_{collect} + t_{write} \qquad (2)$$

In the pure software rendering context, the time for reading the image (t_{read}) usually can be ignored since the rendered image is already stored in the main memory. The time for writing (t_{write}) the final image represents the time for effectively flushing to a file or the time required for displaying onto a display device. The time for compositing ($t_{compose}$) and collecting ($t_{collect}$) are usually the most costly and define the upper bound of the achievable performance. Thus in this study we focused on these two parameters.

$$\begin{aligned} t_{BS} &= \left(\sum_{i=1}^{log_2 n} t_{compose_i}\right) + t_{collect_n} \qquad (3) \\ &= \left[\sum_{i=1}^{log_2 n} \frac{p_{xy}}{2^i}\left(t_{comm_i} + t_{blend_i}\right)\right] + p_{xy}\left(t_{gather_n}\right) \\ &= \left[p_{xy}\left(1 - \frac{1}{n}\right)\left(t_{comm_n} + t_{blend_n}\right)\right] + p_{xy}\left(t_{gather_n}\right) \end{aligned}$$

where

$$t_{comm} = t_{net\_latency} + \frac{1}{net\_bandwidth}$$

$$t_{gather} = t_{gthr\_net\_latency} + \frac{1}{gthr\_net\_bandwidth}$$

$$t_{blend} = t_{cpu\_overhead} + \frac{1}{cpu\_bandwidth}$$

The Binary-Swap compositing time (t_{BS}) can be expressed as shown in Eq. (3). In this equation, the term n corresponds to the number of compositing nodes. The $t_{compose}$ term includes the time for sending (t_{send}), receiving (t_{recv}) and alpha blending (t_{blend}) at each image compositing stage. Since modern network interconnects support full duplex communication, the time for sending and receiving data between pairs of compositing nodes can be substituted by t_{comm} ($=max(t_{send}, t_{recv})$). All these components are directly influenced by the image size as well as the pixel size of the rendered image. In addition, t_{comm} is directly influenced by the network bandwidth and latency. On the other hand, t_{blend} is directly influenced by the processor performance.

The term t_{comm} represents the time for exchanging messages between each pair of compositing nodes. This term depends on the network latency ($t_{net\_latency}$) and the network bandwidth ($_{net\_bandwidth}$). The term t_{blend} depends on the computational overhead ($t_{cpu\_overhead}$) and the computational performance ($_{cpu\_bandwidth}$) itself. Here $t_{net\_latency}$ represents the delay for effectively starting the data transmission. For the sake of simplicity, we used hardware specification values and ignored any other factors. However, for our purpose, we considered the aforementioned setting sufficient for predicting the upper bound of the image compositing performance. It is worth noting that in practice, the latency as well as the sustained bandwidth usually differ from their respective theoretical values.

3. Hybrid Programming Model oriented Image Compositing

The Binary-Swap image compositing method as well as most of other image compositing methods was not designed with the hybrid programming model in mind. To take advantage of the shared-memory address space, it is possible to simplify the data communication pattern and the final collecting process of the Binary-Swap method. However, we investigated the use of another image compositing method, named *Shared-Memory Compositing* (SMC), designed for shared-memory environments.

3.1 Shared-Memory Compositing (SMC) + Binary-Swap (BS)

We have investigated the use of *Shared Memory Compositing* [24] (SMC) which is directly derived from Direct-Send. The composited image generated from SMC will then be composited via Binary-Swap (BS) as shown in **Fig. 3**. In this case, we can expect a light-weight BS compositing because of the reduction in the number of compositing nodes.

Considering that the total number of compositing nodes n can be decomposed into m groups of p shared-memory compositing nodes, the required image compositing time (t_{SMC+BS}) for this approach will be as shown in Eq. (4). The t_{SMC}

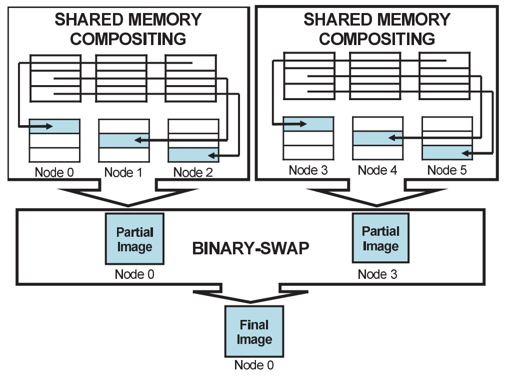


Fig. 3 *Shared-Memory Compositing* + *Binary-Swap* (SMC + BS).

will almost correspond to the time for alpha blending (t_{blend}) images with a size of $1/p$. However, it should be taken into consideration that in NUMA systems, such as Todai T2K, the performance can be affected when blending image data stored in a non-local memory. Since SMC generates m partially composited images, a BS image compositing involving only m nodes will be necessary. This SMC allied with light-weight BS compositing has a great potential to help alleviate the performance degradation in comparison to the traditional BS when a large number of compositing nodes n is involved.

$$t_{SMC+BS} = \max_{Blocks1:m}\left(t_{SMC_p}\right) + \left(\left(\sum_{i=1}^{log_2 m} t_{compose_i}\right) + t_{collect_m}\right)$$

where

$$t_{SMC_p} \approx t_{blend\left(\frac{1}{p}\right)} \quad (4)$$

3.2 Binary-Swap (BS) + Binary-Tree (BT)

Following the same methodology as before, we investigated the synergistic use of different image compositing methods in order to attenuate the network traffic in a large-scale Binary-Swap image compositing. As described in Section 2.1, the communication distance, between the compositing pairs, doubles at each stage. This peculiar communication pattern generates network-wide traffic across the entire system at the final stages. In addition, the gathering process via *MPI_Gather* involving a large number of nodes has the potential to degrade the

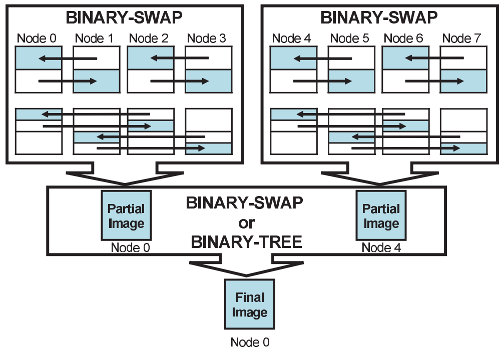


Fig. 4 *Binary-Swap* + (*Binary-Tree* or *Binary-Swap*) (BS + BT or BS + BS).

image compositing performance.

Although the aforementioned approach, which combines SMC and BS shows promising to alleviate this problem, thousands of BS compositing nodes might still be necessary. For instance, if we use 4 nodes SMC, it will still remain 2,048 BS nodes when using 512 nodes of Todai T2K (8,192 processing cores). Considering that the message exchange pattern in BS is highly localized due to the use of a hierarchical binary-tree configuration, we investigated the use of subgroup partitioning aiming to reduce the impact of large-scale data gathering and network-wide traffic in the final stages of BS.

As shown in **Fig. 4**, the compositing nodes can be grouped into subgroups in order to concurrently perform BS compositing. In the next step, we will have a reduced number of images to be composited. If the number of local root nodes is small, for instance less than eight, we can also apply Binary-Tree (BT) compositing to avoid the final image collecting process. For instance, 8,192 BS nodes on Todai T2K can be decomposed into 8 groups of 1,024 BS nodes each in the first step, and 8 BT nodes in the second step. Considering that the total number of nodes n can be decomposed into m groups of p compositing nodes, the required image compositing time for t_{BS+BT} will be as shown in Eq. (5).

$$t_{BS+BT} = \max_{Blocks1:m}\left(\left(\sum_{i=1}^{log_2 p} t_{compose_i}\right) + t_{collect_p}\right) + \left(\sum_{i=1}^{log_2 m} t_{compose_i}\right) \quad (5)$$

4. Experimental Results

4.1 Experimental Setup

We implemented a parallel image compositing application using C programming language together with the MPI communication library and OpenMP directives. This application generates 32-bit RGBA images on-the-fly, with traditional size of 512 × 512 used for performance comparison. We opted for generating a full colored image without any background pixel in order to force the execution of alpha blending throughout the entire image. This therefore eliminates the performance variation due to the different ratios of foreground and background pixels in different images. We did not apply any acceleration technique, such as bounding box or image compression, in order to verify the lower bound of the image compositing performance. We measured the image compositing time using the traditional *MPI_Wtime* function.

On Todai T2K, we used up to 2,048 processing cores for performance evaluation. Hitachi compiler (default) was used by applying "-Os" compiler option. Memory-processor affinity using *numactl* was also explored. A loop of 10 image compositing instances was executed for each measurement and the best compositing time was selected. The inverse function of the measured time gives us the results in FPS (frames per second).

4.2 Theoretical Compositing Performance

Equation 3 was used for calculating the maximum obtainable BS image compositing frame rate on Todai T2K. For this purpose, we used some values available from the T2K Open Supercomputer specification. We considered that the network can deliver data with the rate of 4 GB/s with a hardware latency of 8.5 μs. We also roughly considered that the 2.3 GHz AMD Opteron computational cores can deliver alpha blending operation in the same scale (that is, 2.3 Giga pixels/s). Ignoring any kind of computational overhead, the theoretically maximum achievable image compositing performance for 512 × 512 32-bit RGBA images will be as shown in **Fig. 8**, and this curve shows the theoretical scalability behavior.

4.3 Measured BS Operation Performance

In order to analyze the performance behavior of BS operations on the target

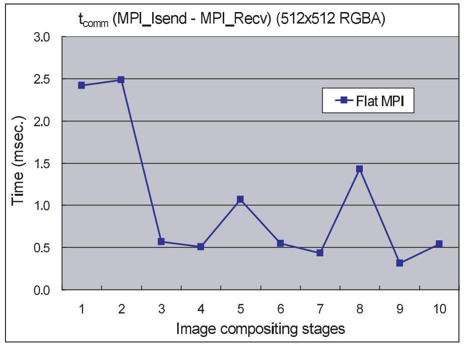


Fig. 5 BS message exchange time using *MPI_Isend* and *MPI_Recv*.

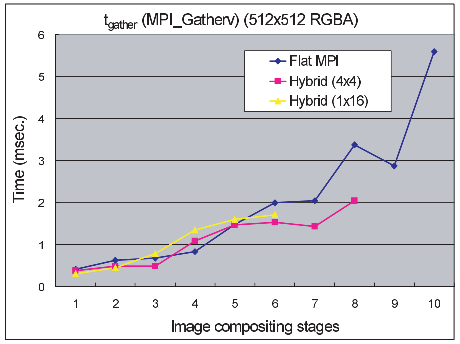


Fig. 6 BS image gathering time using *MPI_Gatherv*.

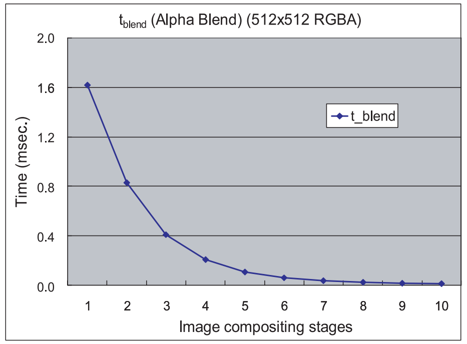


Fig. 7 BS image blending time using alpha blend operation.

hardware, we executed some measurements using up to 1,024 compositing nodes (10 BS compositing stages). We measured the image data exchanged between pairs of compositing nodes (t_{comm}), the final image gathering (t_{gather}), and the image blending process (t_{blend}). As detailed in Section 2.1, as the Binary-Swap image compositing stage advances the image data size required for sending, receiving, and blending is reduced by half. However, we can clearly verify that this decrease in image size has little impact on t_{comm} (**Fig. 5**) and t_{gather} (**Fig. 6**), in contrast to t_{blend} (**Fig. 7**) which does not require network communication. We can also observe a substantial performance degradation of image gathering when involving a large number of compositing nodes.

4.4 BS Compositing Performance

We measured the Flat MPI BS compositing performance using up to 2,048 compositing nodes. Figure 8 shows the results with and without applying the final collecting process. We can observe that the final collecting process produces a considerable performance degradation when the number of compositing nodes becomes large. This performance degradation can be explained in part by the low image gathering and reconstruction performance on a large number of compositing nodes as shown in Fig. 6.

4.5 SMC Compositing Performance

We measured the SMC compositing performance using 4 and 16 SMC nodes, taking into consideration the hardware configuration of Todai T2K where a computational node is composed by four Quad-core AMD Opteron processors. In **Fig. 9**, "OpenMP SMC(16)" shows the measured image compositing performance when using 16 SMC nodes, that is, 16 OpenMP threads. On the other hand, "OpenMP SMC(4)" shows the performance when using 4 SMC nodes. As expected, a similar performance is obtained with up to 4 compositing nodes. However, from this point, on SMC(4) each remaining node will start the BS compositing thus a drop in performance compared to SMC(16) occurs. Roughly speaking, the expected image compositing time of SMC(4) when using 8 compositing nodes will be equivalent to the time of SMC(4) on 4 nodes added with the time of BS on 2 nodes. However, it is worth noting that these 2 BS nodes (inter-CPU) differ from those of Flat MPI BS involving 2 nodes (intra-CPU).

Comparing OpenMP SMC with Flat MPI BS, when using 4 compositing nodes, a performance increase of 97.7% (136.1 FPS $\Rightarrow$ 269.2 FPS) was obtained. When using 16 compositing nodes on "OpenMP SMC(16)", a performance increase of 20.9% (108.4 FPS $\Rightarrow$ 131.1 FPS) was obtained.

4.6 SMC + BS Compositing Performance

The measured performance of SMC allied with BS as well as of Flat MPI is shown in **Fig. 10**. From 32 compositing nodes, only BS compositing will be carried out. However, each of these compositing approaches will have a different number of involved nodes and will be in different BS stages. "SMC(16)-BS" will have 2 nodes and will be in the first BS stage, "SMC(4)-BS" will have 8 nodes and

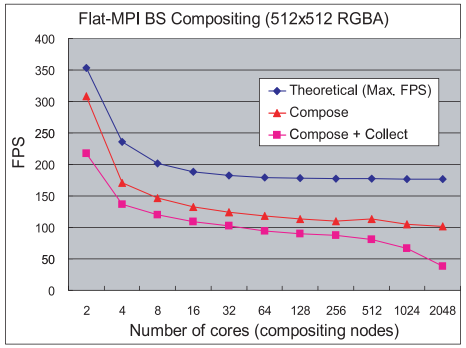


Fig. 8 Theoretical and measured BS image compositing performance.

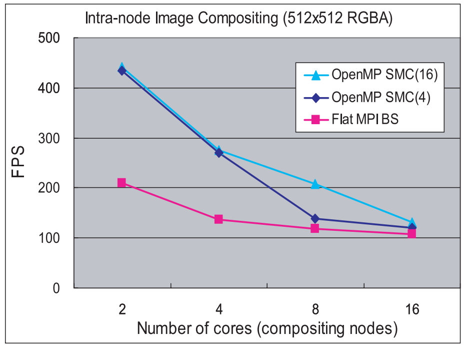


Fig. 9 SMC intra-node image compositing performance.

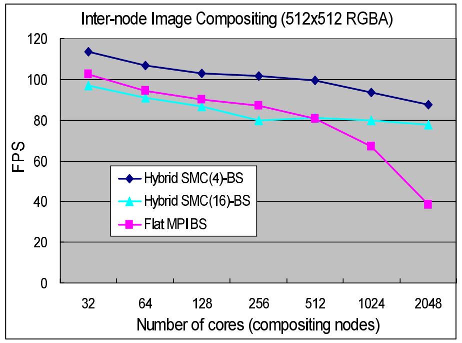


Fig. 10 SMC + BS inter-node image compositing performance.

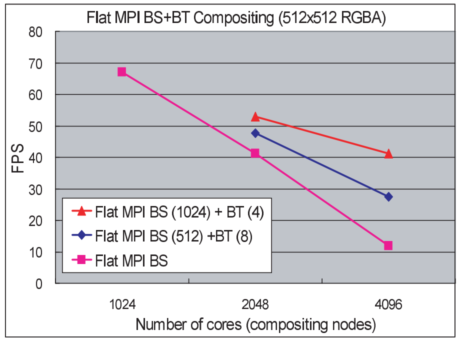


Fig. 11 BS + BT image compositing performance.

will be in the third BS stage, and "Flat MPI BS" will have 32 nodes and will be in the fifth BS stage. Therefore, the amount of data ($p_{xy}/2^{stage}$ per node) that they will send, receive, and blend will differ from each other. This explains the lower image compositing performance of "SMC(16)-BS" compared to "SMC(4)-BS", since the former one requires a higher amount of data to be processed between the compositing pairs at each of the BS stages. The other fact is that the number of nodes involved in the final image gathering will also be different in these three approaches. This therefore might impact the performance when a large number of compositing nodes is involved. For instance, even starting with 2,048 compositing nodes, only 512 nodes will remain for the final image gathering on "SMC(4)-BS". In addition, only 128 nodes will remain when applying "SMC(16)-BS". These aforementioned observations can help us understand the performance of "SMC(16)-BS" compared to "Flat MPI BS". After the inversion on 32 nodes due to the higher amount of data to be processed between compositing pairs, the network-wide traffic and the final image gathering process compromise the performance of "Flat MPI BS" when a large number of nodes (over 1,024 nodes) is involved.

Comparing "SMC(16)-BS" with "Flat MPI BS", a performance increase of 101.0% (38.6 FPS ⇒ 77.6 FPS) was obtained when starting with 2,048 compositing nodes. On the other hand, a performance increase of 126.9% (38.6 FPS ⇒ 87.6 FPS) was obtained when applying "SMC(4)-BS".

4.7 BS + BT Compositing Performance

We used subgroup sizes of 512 and 1,024 BS nodes in order to measure the compositing performance when using 2,048 and 4,096 compositing nodes. In the latter case, when using subgroup of 512 BS nodes, a total of 8 images will result for the subsequent BT compositing. This is equivalent to executing 8 concurrent BS with 512 nodes, and then BT with 8 nodes. On the other hand, four images will be generated when using subgroups of 1,024 BS nodes. This is equivalent to executing 4 concurrent BS with 1,024 nodes, and then BT with 4 nodes. The obtained BS + BT compositing performance is shown in **Fig. 11**. When using subgroups of 512 BS nodes in the case of 4,096 nodes, a performance improvement of 128.3% (12.0 FPS ⇒ 27.4 FPS) was obtained. In addition, a performance improvement of 244.1% (12.0 FPS ⇒ 41.3 FPS) was obtained when using subgroups of 1,024 BS nodes. Although, BS with 512 nodes (80.5 FPS) appears faster than with 1,024 nodes (67.0 FPS), the inversion in the performance is due to the lower BT performance on a higher number of nodes. BT compositing with 8 nodes becomes much slower than with 4 nodes since an additional full size image data is required for processing (send, receive, and blend).

5. Conclusions

In this paper, we presented an image compositing performance evaluation on a T2K Open Supercomputer (Todai T2K). We investigated the performance and the scalability of the Binary-Swap image compositing method focusing on large-scale image compositing. We presented and evaluated a simple modification of Binary-Swap, resulting in a combined image compositing method, in order to exploit the available hybrid programming model. We have also investigated the use of subgroup partitioning aiming to alleviate the network-wide traffic and large-scale data gathering in order to diminish the undesired performance degradation when using a large number of compositing nodes. We obtained encouraging results when applying these improvements to the original Binary-Swap image compositing. Despite its simplicity, the approach of combining existing compositing methods has shown a high potential in tackling the ever increasing number of processing cores on leading-edge high-end HPC systems. The future work includes the investigation of a semi-automatic method for selecting the best combination of image compositing methods.

Acknowledgments This research was conducted using the T2K Open Supercomputer (Todai Combined Cluster) at the University of Tokyo under the "HPC Special Project". This research was supported by Research and Development of the Next-Generation Integrated Simulation of Living Matter, a part of the Development and Use of the Next-Generation Supercomputer Project of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

References

1) Molnar, S., Cox, M., Ellsworth, D. and Fuchs, H.: A Sorting Classification of Parallel Rendering, *IEEE Computer Graphics and Applications*, Vol.14, No.4, pp.23–32 (1994).
2) Chen, L., Fujishiro, I. and Nakajima, K.: Optimizing Parallel Performance of Unstructured Volume Rendering for the Earth Simulator, *Parallel Comput.*, Vol.29, No.3, pp.355–371 (2003).
3) Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K.-L. and O'Hallaron, D.R.: From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing, *SC '06: Proc. 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, ACM, p.91 (2006).
4) Ma, K.-L., Wang, C., Yu, H. and Tikhonova, A.: In-Situ Processing and Visualization for Ultrascale Simulations, *Journal of Physics: Conference Series* (*Proc. DOE SciDAC 2007 Conference*), Vol.78, p.012043 (2007).
5) Peterka, T., Yu, H., Ross, R. and Ma, K.-L.: Parallel Volume Rendering on the IBM Blue Gene/P, *Eurographics/ACM SIGGRAPH Symposium on Parallel Graphics and Visualization* (2008).
6) Rao, A.R., Cecchi, G. and Magnasco, M.: High Performance Computing Environment for Multidimensional Image Analysis, *BMC Cell Biology*, Vol.8, No.Suppl 1, p.S9 (2007).
7) T2K: T2K Open Supercomputer Alliance, http://www.open-supercomputer.org/.
8) Top500: Top500 Supercomputer Sites, http://www.top500.org/.
9) Hsu, W.M.: Segmented Ray Casting for Data Parallel Volume Rendering, *PRS '93: Proc. 1993 Symposium on Parallel Rendering*, New York, NY, USA, ACM, pp.7–14 (1993).
10) Neumann, U.: Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers, *PRS '93: Proc. 1993 Symposium on Parallel Rendering*, New York, NY, USA, ACM, pp.97–104 (1993).
11) Lee, T.-Y., Raghavendra, C.S. and Nicholas, J.B.: 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 (1996).
12) Ma, K.-L., Painter, J.S., Hansen, C.D. and Krogh, M.F.: Parallel Volume Rendering Using Binary-Swap Image Composition, *Computer Graphics and Application*, Vol.14, No.4, pp.59–68 (1994).
13) Ahrens, J. and Painter, J.: Efficient Sort-Last Rendering using Compression-based Image Compositing, *2nd Eurographics Workshop on Parallel Graphics and Visualization*, pp.145–151 (1998).
14) Yang, D.-L., Yu, J.-C. and Chung, Y.-C.: Efficient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers, *J. Supercomput.*, Vol.18, No.2, pp.201–220 (2001).
15) Takeuchi, A., Ino, F. and Hagihara, K.: An Improved Binary-Swap Compositing for Sort-Last Parallel Rendering on Distributed Memory Multiprocessors, *Parallel Comput.*, Vol.29, No.11-12, pp.1745–1762 (2003).
16) Sano, K., Kobayashi, Y. and Nakamura, T.: Differential Coding Scheme for Efficient Parallel Image Composition on a PC Cluster System, *Parallel Comput.*, Vol.30, No.2, pp.285–299 (2004).
17) Lin, C.-F., Chung, Y.-C. and Yang, D.-L.: TRLE–An Efficient Data Compression Scheme for Image Composition of Volume Rendering on Distributed Memory Multicomputers, *J. Supercomput.*, Vol.39, No.3, pp.321–345 (2007).
18) Yu, H., Wang, C. and Ma, K.-L.: Massively Parallel Volume Rendering using 2-3 Swap Image Compositing, *SC '08: Proc. 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp.1–11, IEEE Press (2008).
19) AVS: Advanced Visual Systems, http://www.avs.com/.
20) CEI: Computational Engineering International, http://www.ensight.com/.
21) Stompel, A., Ma, K.-L., Lum, E.B., Ahrens, J. and Patchett, J.: SLIC: Scheduled

Linear Image Compositing for Parallel Volume Rendering, *PVG '03: Proc. 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, Washington, DC, USA, p.6, IEEE Computer Society (2003).

22) Strengert, M., Magallón, M., Weiskopf, D., Guthe, S. and Ertl, T.: Large Volume Visualization of Compressed Time-Dependent Datasets on GPU Clusters, *Parallel Comput.*, Vol.31, No.2, pp.205–219 (2005).

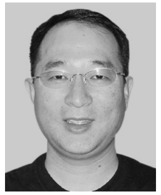
23) Eilemann, S. and Pajarola, R.: Direct Send Compositing for Parallel Sort-Last Rendering, *Eurographics Symposium on Parallel Graphics and Visualization*, pp.29–36 (2007).

24) Reinhard, E. and Hansen, C.: A Comparison of Parallel Compositing Techniques on Shared Memory Architectures, *Third Eurographics Workshop on Parallel Graphics and Visualisation*, pp.115–123 (2000).

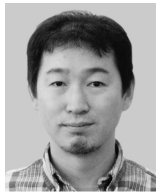
25) Cavin, X., Mion, C. and Filbois, A.: COTS Cluster-based Sort-last Rendering: Performance Evaluation and Pipelined Implementation, *Visualization Conference, IEEE*, Los Alamitos, CA, USA, p.15, IEEE Computer Society (2005).

26) Tay, Y.C.: A Comparison of Pixel Complexity in Composition Techniques for Sort-Last Rendering, *Journal of Parallel and Distributed Computing*, Vol.62, No.1, pp.152 – 171 (2002).

(Received October 3, 2008)
(Accepted January 12, 2009)



Jorji Nonaka is a member of the Computational Science Research Program at RIKEN, Japan. He received his B.Sc. and M.Sc. degrees in Computer Science from University of Brasilia, Brazil, in 1997 and 2001 respectively. He then received his Ph.D. in Informatics from Kyoto University in 2006. From 2006 to 2007, he was a post-doctoral researcher at the Center for the Promotion of Excellence in Higher Education, Kyoto University. His current research interests include Scientific Visualization and Parallel Processing. He is a member of IEEE Computer Society and Visualization Society of Japan.



Kenji Ono is currently the laboratory head of Functionality Simulation and Information Team, VCAD System Research Program, as well as a researcher at the Next-Generation Supercomputer Development Project, both at RIKEN, Japan. He received his degree of D. Eng. in Mechanical Engineering from Kumamoto University, Japan, in 2000. From 1990 to 2001, he was a research engineer at Nissan Motor Company, and from 2001 to 2004, he was an associate professor at the University of Tokyo. His research fields are Computational Fluid Dynamics and Visualization. He is a member of IEEE and ACM.

Hideo Miyachi is the head of Technical Department of Visualization Division at KGT Inc., Japan, where he has been working in the fields of Computer Graphics and Visualization. Prior to joining KGT Inc., he worked at Kubota Ltd. from 1987 to 2005. He obtained his M. Eng. degree from Okayama University, Japan, in 1987. He also obtained his degree of D. Eng. from the University of Tokyo in 2007.