Multi-Step Image Composition Approach for Sort-Last Massively Parallel Rendering

Jorji Nonaka¹,* , Masahiro Fujita¹,², Kenji Ono¹

¹Advanced Visualization Research Team, RIKEN AICS, Kobe, Japan
²Light Transport Entertainment, Inc., Tokyo, Japan

*jorji@riken.jp

Received: November 14, 2014; Accepted: February 21, 2015; Published: May 31, 2015

Abstract. Large-scale numerical simulations on modern leading-edge supercomputers have been continuously generating tremendous amount of data. In-Situ Visualization is widely recognized as the most rational way for analysis and mining of such large data sets by the use of sort-last parallel visualization. However, sort-last method requires communication intensive final image composition and can suffer from scalability problem on massively parallel rendering and compositing environments. In this paper, we present the Multi-Step Image Composition approach to achieve scalability by minimizing undesirable performance degradation on such massively parallel rendering environments. We verified the effectiveness of this proposed approach on K computer, installed at RIKEN AICS, and achieved a speedup of 1.8× to 7.8× using 32,768 composition nodes and different image sizes. We foresee a great potential of this method to meet the even larger image composition demands brought about by the rapid increase in the number of processing elements on modern HPC systems.

Keywords: Parallel rendering, parallel image composition, sort-last parallel visualization, in-situ visualization, K computer.

1. Introduction

Visual data analysis and mining have been playing an important role in assisting computer-aided research and development by means of numerical simulations in a broad range of science and engineering fields. More particularly, there is a recent coined term “High Performance Visualization” [1] referring to the field of scientific visualization focusing on HPC (High Performance Computing) systems. HPC systems stated here refer not only to the supercomputers used to run large-scale numerical simulations, but also to the hardware graphics accelerated systems, such as visualization clusters. In-situ processing and visualization [2] is widely recognized as the most rational way for executing visualization in HPC systems.
environments since it allows a flexible combination for coupling simulation and visualization processes. Currently, it is classified into “Co-processing” (Tightly coupled), “Concurrent processing” (Loosely coupled), and “Hybrid processing” approaches [1, 3].

Co-processing approach refers to the situation where the visualization process is executed in the same system used for numerical simulations. In such case, visualization code will have direct access to the simulation data stored in the memory (internal and external), thus avoiding sometimes prohibitive transfer cost to another system for visualization and analysis. The process of converting these simulation data into visual images through a rendering process is known as Visualization (Fig. 1). Rendering pipeline is an abstraction model for a sequence of operations to transform a given data into visual images. In HPC environments, this rendering pipeline is usually executed in parallel, and it is classified into sort-first, sort-middle and sort-last approaches depending on where the sorting operation takes place [5]. Depending on this sorting position, the data to be sorted can be raw primitives, screen-space primitives or pixels. Among them, the sort-last parallel rendering which works by sorting pixel data is widely recognized as the most suitable approach for massively parallel environments.

Figure 2 shows a schematic view of the sort-last parallel rendering approach. As shown in this figure, each “rendering node” generates the image corresponding to the portion of the assigned data. Therefore, there is a need to merge all these generated images in a sorted order for generating the single final image. This merging process is known as image composition, and when it is executed in parallel by the “composition nodes” it is called parallel image composition. Figure 2 also shows the Tree-based image composition approach, which Binary-Swap method used in this work is derived from, as an example of parallel image composition method. This image composition process is expected to handle the entire set of image data types generated by the rendering processes, including opaque pixel data as shown in Fig. 3, as well as, semi-transparent pixel data as used in volume rendering. The scalability requirement is directly proportional to the available parallelism of modern supercomputers.

![Image Diagram](image.png)
which is currently in the order of tens of thousands of computational nodes and hundreds of thousands or even millions of computational cores [4]. In the case of K computer, it has 82,944 computational nodes or 663,552 computational cores. It is important to make clear that the term “composition nodes” used in this paper can refer to both computational nodes (Hybrid MPI-OpenMP mode) or computational cores (Flat MPI mode), and will depend on the parallel rendering run-time environment.

It is worth noting that current sort-last massively parallel rendering environment can reach tens of thousands of rendering nodes [1]. It is not different in the case of K computer which possesses a visualization library provided by Fujitsu [6]. They provided a theoretical performance study for rendering and compositing 1 MPixel images (1024×1024) using up to 80,000 rendering nodes. There is also the LSGL (Large-Scale Graphics Library) [7] being developed which is capable to render an impressive 128 MPixel images (16,384×8192) using full 82,944 computational nodes of K computer as shown in the left side of Fig. 3. Although, ultra high-resolution images are useful for displaying and analyzing large simulation data, as static or still images; moderate resolution images (For instance: FullHD and 4K2K) can be useful for displaying large number of sequential images, as movies or animations (right side of Fig. 3). Each image can represent each time step of time-varying numerical simulation or also different viewpoint images as described in [8]. In the aforementioned case, Kageyama et al. reported a generation of 390 images with 1600×1200 resolution (almost 2 MPixels). In such situations, hundreds of image composition calls of moderate image resolution are required and a scalable image composition becomes important. Taking all these into consideration, the goal of this work was to develop a scalable parallel image composition algorithm, for moderate resolution images, which can work efficiently on K computer using tens of thousands of rendering and composition nodes.

In massively parallel rendering environments where tens of thousands of rendering nodes are involved, it is important to have a scalable image composition code for this degree of parallelism. However, scalability problems on massively parallel image composition environments have already been reported on different HPC systems such as T2K Open Supercomputer [9], IBM Blue Gene/L [10], IBM Blue Gene/P [11], and K Computer [12].

Figure 2: Sort-last parallel rendering approach which requires image composition right after the rendering process, and an example of parallel image composition method.
Figure 3: Some examples of massively parallel rendered images on K computer and the usual image resolution requirements for still and dynamic images.

K computer, besides this performance degradation when using a large composition node counts, there is a more critical issue since MPI_Gatherv, an MPI collective function used to implement these aforementioned image composition algorithms, does not work over 50K composition node counts due to the buffer error (MRQ Overflow) in the current MPI implementation for K computer. Although it might be a temporary problem, large-scale parallel image composition using 64K composition nodes, such as presented in [11], is not directly possible. In this paper, we focused on the Multi-Step approach presented in [12] and expanded the investigation including floating point pixel format for attending high-quality rendering requirements; a method for selecting the maximum group size to be used in each Step; and the Hybrid MPI-OpenMP image composition approach. This Multi-Step approach is simple to implement since it uses standard MPI functions, and works by minimizing the performance degradation of MPI collective communication in a massively parallel environment. In addition, by using the proposed approach, we could avoid the buffer overflow problem on K computer when calling a large-scale MPI_Gatherv. More practical results of this are the images presented in Fig. 3 which were composited using 65,536 nodes on K computer. It is worth noting that although these images possess information regarding transparency (A), they were composited without using it since they have only opaque pixels. We worked on image composition of semi-transparent pixel images to meet the future requirements of ongoing parallel rendering development [7].

2. Related Work

Parallel image composition algorithms have been widely studied over the past two decades, and several algorithms have been proposed and evaluated on different parallel architecture systems. Two of the most popular methods, Direct Send [13, 14] and Binary-Swap [15] (Fig. 4), have been proposed almost two decades ago. Parallel Pipeline [16] is another important method which was used on some commercial parallel visualization applications such as AVS/Express PST (Parallel Support Toolkit) and CEI Ensight DR (Distributed Ren-
dering). However this method is limited to small-scale parallel environments, and there is no report of using this for large-scale image composition. *Shift-based* image composition method [17] has been proposed for medium-size parallel systems based on Infiniband Fat-Tree interconnection network which uses shift permutation communication pattern.

Some optimizations and extensions for Direct-Send and Binary-Swap have been proposed to overcome their performance and operational drawbacks on large-scale image composition environments. *SLIC* (Scheduled Linear Image Compositing) [18] can be considered an optimized Direct-Send where unnecessary communications are eliminated by generating on-the-fly composition scheduling taking into consideration the image overlapping in the viewing direction. *2-3 Swap* [19] can be considered an extension for Binary-Swap to provide image composition of non-power of two number of composition nodes. This method was used on the visualization library for K computer provided by Fujitsu [6]. *Radix-k* [20] is a flexible image composition method based on Direct-Send which allows numerous configurations by changing the *k* value which represents the number of elements inside a group. The most interesting aspect of Radix-k is that when *k* value is the number of total composition nodes it becomes the traditional Direct-Send, and when *k* value is two, it becomes equivalent to the Binary-Swap, that is, it unifies the Direct-Send and Binary-Swap while enabling other configurations. Some optimizations, including active-pixel encoding and compression, have also been evaluated for Radix-k [22].

More recently, it has been reported a scalability problem of Radix-k when using large factors of *k* values, and an alternative algorithm called *Telescope* method which can work with both Binary-Swap and Radix-k was proposed [11]. Telescope method works as a recursive approach to reduce an arbitrary number of nodes to the closest power-of-two number of nodes as shown in Fig. 5. Telescope Radix-k has shown better performance on smaller composition node counts while Telescope Binary-Swap has shown better performance on larger
node counts, in the order of thousands, on IBM Blue Gene/P. However, from the image composition performance graphs shown in [11], we can clearly verify a performance degradation of image composition when reaching tens of thousands of nodes. Considering that there are other approaches for reducing non-power-of-two number of nodes \((m = 2^n + r)\) into the largest power-of-two number of nodes \((m = 2^n)\) such as Reduce [19], or Fold [11], and 2-3-4 Decomposition [23] as shown in Fig. 5, therefore scalability improvements of original Binary-Swap will directly benefit all of these aforementioned versions of Binary-Swap.

### 3. Parallel Image Composition

In sort-last parallel rendering approach, parallel image composition works by merging all rendered images generated from \(m\) rendering nodes, where \(2^n < m < 2^{n+1}\), into a single final image as shown in Fig. 2. The images to be composited are usually a sequence of pixels \((p)\) which can have a combination of color (RGB), transparency (A) and depth (Z) information. Although these images are 2D data, with width and height attributes, they can be represented as 1D data (sequence of pixels) in order to facilitate the computation. The image composition process can be analyzed separately as communication and computational tasks. The computational part is the common task of different image composition methods, and works as a per-pixel basis executing alpha blending, for semi-transparent image data, or Z-order depth composition for opaque image data. In the case of opaque pixel data, the image composition order does not interfere in the final image results. However, alpha blending operation of semi-transparent images is usually a noncommutative operation and requires correct ordering. For instance, volume rendered images are usually combined by using the well-known Over operator [24] which is noncommutative. For the sake of simplicity, in this paper, we assume that the images are sorted by the process rank number from the closest to farthest distance from the viewing point for correct alpha blending operation. This image data sorting can be simply done by creating a new MPI Global Communicator with sorted process ranks to avoid costly image data moving among the processes (image composition...
nodes). The communication part and the applied image data partitioning approach provide
the uniqueness between different image composition methods. Parallel image composition
works by gradually compositing the image data which is executes in several stages. In the
following subsections we will describe the Binary-Swap image composition method show-
ing how it works, and will discuss the scalability issue on massively parallel environments
involving tens of thousands of nodes.

3.1. Binary-Swap

Binary-Swap image composition algorithm works by swapping portions of image data be-
tween pairs of composition nodes in order to keep every node busy in all stages of the
composition process as shown in Fig. 4. At each stage, it exchanges and merges portions of
image data, and at the end it gathers the composited image fragments with size \( \frac{1}{2^{\log_2 m}} \) and
reconstruct the final composited image. The image data here is treated as 1D sequence
of pixels, and is recursively divided into two parts. In order to optimize the communication
process, it slightly differs from the original approach [15] where the image data was treated
as 2D data, and successively divided in horizontal and vertical manner. In order to execute
the composition in noncommutative manner, the distance pairs are recursively doubled. Be-
cause of this peculiar message exchanging pattern, it works only with power-of-two number
of nodes \( 2^n \). However, as shown in Fig. 5, Binary-Swap of arbitrary number of nodes can
be achieved by combining it with the existing conversion methods to power-of-two number
of nodes.

Considering \( m \), a power-of-two \( 2^n \) number of nodes, Binary-Swap requires exactly
\( \log_2 m \) stages for image data exchanging and merging. At the end, each composition node
will possess a portion of the final image which should be gathered in order to reconstruct the
final image. If the total number of pixels \( p \) in the image is an even number thus each compo-
sition node will have \( \frac{1}{2^{\log_2 m}} \) of the total number of pixels. However, if the number of pixels
is an odd number, an uneven distribution of pixel will occur, and this is one of the reasons
of using \( MPI_{\text{Gatherv}} \) function for gathering the fragments of final composited images. As
shown in Fig. 4, the Binary-Swap image composition time will be the accumulated time of
image data exchanging (communication) and merging (computation) added with the gather-
ing (communication) time. Equation 1 shows the communication and computational costs of
Binary-Swap where \( m \) corresponds to the number of composition nodes, and \( p \) corresponds
to the number of pixels. Peterka et al., in a book chapter of [1], and Cavin et al. in [17] used
the simplified model proposed by Chan et al. in [25] for describing the theoretical commu-
nication and composition costs. They added the latency term for each of the stages, that is,
a constant multiplied by the number of stages \( \log_2 m \). However, they ignored the gathering
time which can become a bottleneck as reported in [11] in a large-scale image composition
when tens of thousands of image composition nodes can be involved.

In practice, we can expect other variables affecting the composition time, however from
the aforementioned cost model, we can expect small variation in exchanging and merging
parts in large composition node counts \( m \), since the term \( (m - 1)/m \) will converge to 1. On
the other hand, the gathering part can be affected by the gathering time for large \( m \) \( (t_{\text{gather}_m}) \).
In the case of K computer, it is reported that better communication performance is obtained
Figure 6: Flat MPI Binary-Swap performance (Best time) on K computer.

when using larger message size [26]. Considering that, the image data size to be exchanged is halved as the composition stage advances, we can expect some performance loss when reaching to the small image sizes.

\[
\begin{align*}
t_{BS} &= \left(\sum_{i=1}^{\log_2 m} \frac{1}{2^i} p(t_{exchange} + t_{merge})\right) + p(t_{gather}) \\
&\approx \frac{m-1}{m} p(t_{exchange} + t_{merge}) + p(t_{gather}) \\
&\approx p\frac{m-1}{m} t_{exchange} + p\frac{m-1}{m} t_{merge} + p(t_{gather}) 
\end{align*}
\]

3.2. Scalability Issue of Massively Parallel Binary-Swap

In order to verify the performance behavior of Binary-Swap on massively parallel environments, we implemented a benchmarking application for running on K computer which generates 128-bit RGBA images with 32-bit floating point pixel components (RGB color information and A degree of transparency) simulating high-quality volume rendering. We used plain color images (single color for the entire image) with full active pixels (without background pixels) which correspond to the worst case scenario since background pixels can be skipped during the merging process. Besides the image size of 4 MPixels (2048\times2048) used in [11], we also utilized the image sizes of 1 MPixel (1024\times1024), 2 MPixels (2048\times1024), and 8 MPixels (4096\times2048). These range of image sizes includes the FullHD (approx. 2 MPixels) and 4K2K (approx. 8 MPixels) which is one of the highest resolution available in the commodity monitor market.

We evaluated by using Flat MPI run-time environment with up to 32,768 composition nodes. Figure 6 shows the best time of Binary-Swap image composition time from 5 measurements. Actually, 6 measurements have been executed for each image resolution and number of composition nodes, however as similar to [11] we ignored the first measurement which can have higher initialization overhead and might produce biased results. From this figure, we can clearly verify that Binary-Swap does not scale in large composition node...
counts especially on thousands of composition nodes. This performance behavior on large image composition node counts is similar to the performance evaluation done on Intrepid (an IBM Blue Gene/P supercomputer at the Argonne National Laboratory, USA) using up to 65,536 composition nodes in Flat MPI run-time environment [11]. The performance evaluation graph presented in [11] shows that Binary-Swap and some versions of Radix-k time almost doubled from 512 to 65,536 composition nodes, and in the case of Radix-k with \( k \) value equal to 32 the time almost tripled. In order to minimize this image composition performance degradation on large node counts, we will describe the proposed Multi-Step image composition approach in the next section.

4. Multi-Step Image Composition

In the previous section, we could observe that the performance degradation becomes larger as the number of composition nodes increases thus a rational way to minimize such performance loss is to avoid image composition using larger composition node counts. From Fig. 4 we can observe that the message exchange pattern of Binary-Swap is highly localized since it is based on tree configuration. At each stage \( i \), a group of \( 2^i \) composition nodes will have their portion of images exchanged and merged, and by gathering all the composited image portions, we can reconstruct a partially composited image from \( 2^i \) images. Taking the right side of Fig. 4 as an example, we can obtain 4 partially composited images in stage 1, and we can execute another Binary-Swap with these 4 images in order to obtain the final composited image. In the same manner, in stage 2 we can obtain 2 partially composited images to be merged to generate the final image. From this observation, we can divide the entire composition nodes into groups of composition nodes in order to execute small sets of independent Binary-Swap at each of the groups. As a result, by executing the image composition in multiple steps, we can avoid the performance degradation observed on massively parallel image composition executed in single step. This approach can theoretically
be applied to any image composition approaches, and the schematic view of this process applied to Binary-Swap is shown in Fig. 7, and the pseudocode for group creation is shown at the left side, which starts with a pre-processing for converting to a power-of-two number of nodes (Fig. 5).

The idea of Multi-Step image composition first appeared in [10] where different image composition approaches could be combined (Binary-Swap with Direct Sent and Binary Tree) on IBM Blue Gene/L supercomputer. In a more recent evaluation, Binary-Swap only Multi-Step approach has been evaluated on K computer using 32-bit RGBA images [12]. In this paper, we made a more profound investigation extending its use for 128-bit RGBA images, with floating point color and transparency information, required in high-quality volume rendering. We also investigated the use of performance degradation ratio, from the best time measurements, as a criterion for determining the optimum group size for node decomposition. Multi-Step image composition works by decomposing the image composition nodes into groups with equal number of nodes. Since alpha blending operation is noncommutative, there is a need to divide recursively into contiguous portions. In order to generate these groups we can use the standard collective MPI functions such as MPI_Comm_Split. Actually, we are just creating new MPI communicators which will only communicate among the grouped composition nodes. The automatic renumbering process defined by the MPI function will define each local root nodes for the generated groups. These local root nodes will be responsible for gathering and reconstructing the composited image fragments inside the groups, and only these local root nodes will participate in the next Step of image composition process. Considering that \( q \) is the selected group size, and \( m \) can be decomposed into \( r \) number of groups \( (m = r \times q) \), the Multi-Step (2-Step in this example) image composition time will be as shown in Eq. 2. We can observe that it is equivalent to the sum of Binary-Swap time of \( q \) nodes and Binary-Swap time of \( r \) nodes.

\[
\begin{align*}
    t_{MS} &= t_{\text{slowest}} \left[ \left( \sum_{i=1}^{\log_2 q} \frac{1}{2} p \left( t_{\text{exchange}} + t_{\text{merge}} \right) \right) + p \left( t_{\text{gather}} \right) \right] + \\
    &\left( \sum_{i=1}^{\log_2 r} \frac{1}{2} p \left( t_{\text{exchange}} + t_{\text{merge}} \right) \right) + p \left( t_{\text{gather}} \right) 
\end{align*}
\]  

The main question that arises is how we can define the optimum group sizes for the sets of image sizes and composition nodes. This is a hard question to answer since communication and computational performance on HPC systems with shared resource can have dynamic run-time performance behavior influenced by several factors. In this paper, we propose the use of image composition time degradation ratio for defining the group sizes for each of the image sizes. The degradation ratio here refers to the ratio between composition times in sequence \( (2^n/2^{n-1}) \). In the next subsections we will discuss this approach for determining the group sizes, and the execution of group creation process as a pre-processing to minimize runtime overhead.

4.1. Determination of Group Sizes

Figure 8 shows the Binary-Swap time degradation ratio in the range of 256 and 32,768 composition nodes of the measured timings shown in Fig. 6. In the case of degradation ratio
between consecutive measurements \( (2^n \text{ and } 2^{n-1}) \) number of nodes), we can verify that most of the degradation ratios surpasses 10% over 1024 composition nodes (Graphs in the left side). In the case of measurements by skipping one measurement \( (2^n \text{ and } 2^{n-2}) \) number of nodes), we can verify similar behavior of steeper curve from 1024 nodes (Graphs in the right side). Analyzing this performance degradation behavior from 1024 nodes, we can set “10%” \( (2^n \text{ and } 2^{n-1}) \) as being the threshold for determining the maximum group sizes, that is, we should select a number of nodes which does not surpass this threshold. Considering that the group size to be used should be smaller than the number of nodes defined in Tab. 1, and also that the group size should always be a power-of-two, we can use the previous power-of-two number of nodes as being the maximum group sizes as shown in Tab. 1. In order to verify whether it is an optimum threshold, we also verified using other group sizes as shown in Tab. 2. In the next section, we present and discuss the performance evaluation results obtained on K computer.

Figure 8: Degradation ratio between consecutive and nonconsecutive (by skipping one) measurements for different image sizes.

Table 1: Points where the degradation ratio surpasses the threshold, and the maximum group sizes which do not surpass it.

| Image Size | \( (2^n / 2^{n-1}) > 10\% \) | Previous Group Size |
|------------|-------------------------------|---------------------|
| 1 MPixel   | 1024                          | 512                 |
| 2 MPixels  | 2048                          | 1024                |
| 4 MPixels  | 2048                          | 1024                |
| 8 MPixels  | 4096                          | 2048                |

4.2. Initialization Overhead

Binary-Swap is based on static communication pattern based on distance doubling approach which facilitates the decomposition tasks for generating the groups. Taking advantage of the deterministic communication pattern, we can generate all the necessary groups (MPI Communicators) in advance as a pre-processing stage. Graphs in the Fig. 9 show the group gen-
Figure 9: Multi-Step initialization overhead (group creation and memory allocation) on K Computer.

eration time and some memory (buffer) allocation time for executing the Multi-Step image composition when using 10% ($2^n/2^{n-1}$) as the threshold. Although it includes the memory allocation time, most of the required time is spent on the group creation. We can verify that higher initialization time is required on large composition node counts. We presume that this is caused by the MPI_Comm_Split, since it is a collective operation. However, once the groups (or MPI Communicators) are created there is no need to call the initialization again while the number of composition nodes remains the same.

5. Experimental Results

We implemented a benchmarking application, briefly explained in Subsection 3.2, using C programming language together with MPI library to run on K computer. It generates 128-bit RGBA images with 32-bit floating point pixel components (RGB color information and A degree of transparency) simulating high-quality volume rendering. We used plain color images (single color for the entire image) with full active pixels (without background pixels). Image sizes of 1, 2, 4 and 8 MPixels have been used for the performance evaluation measured by using the MPI_Wtime function. The MPI_Comm_Split function has been used for creating the MPI communicators for each of the groups used in Multi-Step image composition. The benchmarking application was compiled using the Fujitsu C cross-compiler for K computer (mpifccpx) with “-Kfast”, which is a part of the recommended compiler options and corresponds to the “-O3” optimization level. We used the Flat MPI run-time environment with up to 32,768 composition nodes. During the job submission, for the evaluation purposes, we have not requested any specific hardware topology in the resource allocation.

5.1. Multi-Step Image Composition

We evaluated the Multi-Step image composition approach utilizing the group sizes defined by the threshold of 10% ($2^n/2^{n-1}$), that is, 512 for the image size of 1 MPixel, 1024 for the image sizes of 2 and 4 MPixels, and 2048 for the image size of 8 MPixels. For other image
Figure 10: Comparison between “Original” and “Multi-Step” versions of Binary-Swap (in Flat MPI mode) using image sizes of 1, 2, 4 and 8 MPixels.
Figure 11: *Multi-Step* image composition performance when using different group sizes.

sizes different from the aforementioned sizes, we can use the image size in between as the threshold for group size selection as shown in [12]. For instance, the group size of 1024 will be valid for any image sizes between 1.5 and 6 MPixels. Figure 10 shows the boxplot graphs for the five measurements of “Original” Binary-Swap (graphs at the left side), and the “*Multi-Step*” version (graphs at the right side). In these graphs, the red horizontal lines represent the median of the measurements, that is, where it divides them into two halves. The vertical boxes graphically represent the ranges of the middle points of these two halves. The measured times which were considered distant from the normal range were considered outliers and were represented as blue “+” signs in the graphs. Graphs in both sides are shown in the same scale for visual comparison purposes, and we can verify that the performance degradation on large composition node counts is minimized bringing high scalability even on massively composition environment with tens of thousands of nodes. From the graphs at the right side, we can also verify a subtle performance loss when changing from “Original” Binary-Swap, which is marked as “(BS)” at the horizontal axis, to the *Multi-Step* version of Binary-Swap. However, we can clearly verify that the benefit of executing Binary-Swap using small number of nodes, delimited by the group size, is larger than the cost of executing it in multiple steps.

In order to verify the effect of using different group sizes, we selected a range of group sizes for executing the performance evaluation of *Multi-Step* image composition. The utilized group sizes are described in Tab. 2. Figure 11 shows the obtained *Multi-Step* image
Table 2: Some group sizes used for verifying the performance behavior of Multi-Step image composition.

| Image Size | Group Size 1 | Group Size 2 | Group Size 3 |
|------------|--------------|--------------|--------------|
| 1 MPixel   | 128          | 256          | 512          |
| 2 MPixels  | 256          | 512          | 1024         |
| 4 MPixels  | 512          | 1024         | 2048         |
| 8 MPixels  | 1024         | 2048         | 4096         |

Figure 12: Scalability analysis of Multi-Step image composition using threshold of 10% and up to 131,072 composition nodes on K computer.

composition time (Best time), and from these graphs, we can verify that the group size selection can have higher interference when using larger image sizes. The selected group sizes (for the threshold of 10% ($2^n/2^{n-1}$)): 512 (1 MPixel); 1024 (2 and 4 MPixels); and 2048 (8 MPixels) showed good performance on large composition node counts (tens of thousands of nodes) and seem to be reasonable thresholds for group size selection. The only drawback was the group size of 1024 for 4 MPixel image size, where better results were obtained when using group size of 2048. However, from the obtained results, we can say that this threshold value of 10% can serve as a good initial guess for determining the group sizes. Besides the scalability issue, there was another issue, on massively parallel Binary-Swap on K computer, related to MPI_Gatherv which prevented the execution using more than 32,768 nodes. By using the proposed Multi-Step approach where the image composition size does not surpass the pre-determined group size, we can avoid this MPI_Gatherv problem. In order to verify this assumption, we executed a massively parallel Multi-Step image composition using up to 131,072 composition nodes in Flat MPI mode. Graphs on Fig. 12 show the best time of image composition performance obtained from six measurements, and we can verify that the scalability is not lost even using a hundred of thousands of composition nodes.
6. Conclusions

In this paper we presented the Multi-Step Image Composition approach suited for massively parallel rendering environments where tens of thousands of rendering and composition nodes can be involved. This method works by recursively dividing the composition nodes into smaller groups, with pre-defined group size, and executes the image composition process independently within each of the groups. The name Multi-Step comes from the fact that it executes in multiple steps until the number of groups becomes smaller than the pre-defined group size. We further extended previous works by applying to floating point image data required for high-quality volume rendering, and we presented a new approach for determining the group size based on performance degradation ratio and threshold values. Performance evaluation on K computer showed promising results in a massively image composition environment with tens of thousands of nodes. From these obtained results, we can foresee a great potential of this method to meet the large-scale image composition demands brought about by the rapid increase in processor counts of current and expected next-generation HPC systems. Since a complete image composition is executed at each step within each of the groups, thus we can theoretically use any of the available image composition algorithms. Obviously, it is better to select the most suitable algorithm for each of the cases. Therefore, as a future work, we can include the investigation of combining other image composition methods, as well as dynamic group size selection using some run-time parameters.

Acknowledgement

Part of the results was obtained by using the K computer at the RIKEN AICS (Advanced Institute for Computational Science) in Kobe, Japan. This work has been partially supported by JSPS under KAKENHI Grant Number 26280043.

References

[1] E. W. Bethel, H. Childs, C. Hansen: High Performance Visualization: Enabling Extreme-Scale Scientific Insight, Chapman & Hall/CRC, Boca Raton, 2012.

[2] K.-L. Ma, C. Wang, H. Yu, A. Tikhonova: In-Situ Processing and Visualization for Ultrascale Simulations, Journal of Physics: Conference Series (Proceedings of DOE SciDAC 2007 Conference), 78 (2007).

[3] M. Rivi, L. Calori, G. Muscianisi, V. Slavnic: In-Situ Visualization: State-of-the-Art and Some Use Cases, Partnership for Advanced Computing in Europe (PRACE) Whitepaper, (2012).

[4] Top500 Supercomputer Sites, http://www.top500.org, (2014).

[5] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs: A Sorting Classification of Parallel Rendering, IEEE Computer Graphics & Applications, 14:4 (1994), 23–32.
[6] A. Ogasa, H. Maesaka, K. Sakamoto, S. Otagiri: Visualization Technology for the K Computer, *Fujitsu Scientific & Technical Journal*, 48:3 (2012), 348–356.

[7] M. Fujita, J. Nonaka, K. Ono: LSGL: Large-Scale Graphics Library for Peta-scale Computing Environments, in *High Performance Graphics 2014 (HPG 2014)*, Lyon, 2014.

[8] A. Kageyama, T. Yamada: An Approach to Exascale Visualization: Interactive Viewing of In-Situ Visualization, *Computer Physics Communications*, 185:1 (2014), 79–85.

[9] J. Nonaka, K. Ono, H. Miyachi: Performance Evaluation of Large-Scale Parallel Image Compositing on a T2K Open Supercomputer, *IPSJ Transactions on Advanced Computing Systems*, 2:2 (2009), 122–130.

[10] J. Nonaka, K. Ono, H. Miyachi: Theoretical and Practical Performance and Scalability Analyses of Binary-Swap Image Composition Method on IBM Blue Gene/L, in *Proceedings of the 1st International Workshop on Super Visualization (IWSV’08) (Unpublished Manuscript)*, Kos, 2008.

[11] K. Moreland, W. Kendall, T. Peterka, J. Huang: An Image Compositing Solution at Scale, in *Proceedings of the 2011 ACM/IEEE conference on Supercomputing (SC’11)*, Seattle, 2011, 1–10.

[12] J. Nonaka, K. Ono, M. Fujita: Multi-Step Image Composition for Massively Parallel Rendering, in *Proceedings of the 2014 International Conference on High Performance Computing & Simulations (HPCS 2014)*, Bologna, 2014, 627–634.

[13] W. M. Hsu: Segmented Ray Casting for Data Parallel Volume Rendering, in *Proceedings of the 1993 Symposium on Parallel Rendering (PRS’93)*, San Jose, 1993, 7–14.

[14] U. Neumann: Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers, in *Proceedings of the 1993 Symposium on Parallel Rendering (PRS’93)*, San Jose, 1993, 97–104.

[15] K.-L. Ma, J. S. Painter, C. D. Hansen, M. F. Krogh: Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE Computer Graphics & Applications*, 14:4 (1994), 59–68.

[16] T.-Y. Lee, C. S. Raghavendra, J. B. Nicholas: Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers, *IEEE Transactions on Visualization and Computer Graphics*, 2:3 (1996), 202–217.

[17] X. Cavin, O. Demengen: Shift-based Parallel Image Compositing on Infiniband Fat-Trees, in *Proceedings of the Eurographics 2012 Symposium on Parallel Graphics and Visualization (EGPGV’12)*, Cagliari, 2012, 129–138.
[18] A. Stompel, K-L. Ma, E. B. Lum, J. Ahrens, J. Patchett: SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering, in Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics, Seattle, 2003, 33–40.

[19] H. Yu, C. Wang, K-L. Ma: Massively Parallel Volume Rendering using 2-3 Swap Image Compositing, in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC’08), Austin, 2008, 1–11.

[20] T. Peterka, D. Goodell, R. Ross, H-W. Shen, R. Thakur: A Configurable Algorithm for Parallel Image-Compositing Applications, in Proceedings of the 2009 ACM/IEEE Conference on Supercomputing (SC’09), Portland, 2009, 1–10.

[21] T. Peterka, H. Yu, R. Ross, K-L. Ma: Parallel Volume Rendering on the IBM Blue Gene/P, in Proceedings of the Eurographics 2008 Symposium on Parallel Graphics and Visualization (EGPGV’08), Crete, 2008.

[22] W. Kendall, T. Peterka, J. Huang, H-W. Shen, R. Ross: Accelerating and Benchmarking Radix-k Image Compositing at Large-Scale, in Proceedings of the Eurographics 2010 Symposium on Parallel Graphics and Visualization (EGPGV’10), Norrköping, 2010, 101–110.

[23] J. Nonaka, C. Bi, K. Ono, M. Fujita: 2-3-4 Decomposition Method for Large-Scale Parallel Image Composition with Arbitrary Number of Nodes, in Proceedings of the 2014 International Conference on Systems Informatics, Modelling and Simulation (SIMS 2014), Sheffield, 2014, 59–64.

[24] T. Porter, T. Duff: Compositing Digital Images, ACM SIGGRAPH Computer Graphics, 18:3 (1984), 253–259.

[25] E. Chan, M. Heimlich, A. Purkayastha, R. van de Geijn: Collective Communication: Theory, Practice, and Experience: Research Articles, Concurrency and Computation: Practice and Experience, 19:13 (2007), 1479–1783.

[26] N. Shida, S. Sumimoto, A. Uno: MPI Library and Low-Level Communication on the K Computer, Fujitsu Scientific and Technical Journal, 48:3 (2012), 324–330.