# An Improved Binary-Swap Compositing for Sort-Last Parallel Rendering on Distributed Memory Multiprocessors ⋆

Akira Takeuchi [a,1], Fumihiko Ino [b,*], Kenichi Hagihara [b]

[a] *Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan*

[b] *Graduate School of Information Science and Technology, Osaka University, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan*

**Abstract**

Sort-last parallel rendering is a good rendering scheme on distributed memory multiprocessors. This paper presents an improvement on the binary-swap (BS) method, which is an efficient image compositing algorithm for sort-last parallel rendering. Our compositing method uses three acceleration techniques, compared to the original BS method: (1) the interleaved splitting, (2) multiple bounding rectangle, and (3) run-length encoding. Through the use of the three techniques, our method balances the compositing workload among processors, exploits more sparsity of the image, and reduces the cost of communication.

We also show some experimental results on a PC cluster. The results show that our method completes the image compositing faster than the original BS method, and its speedup to the original increases with the number of processors.

*Key words:* Cluster computing; Image compositing; Parallel rendering; Performance evaluation; Binary-swap compositing

⋆ A preliminary version of this work was presented in the 18th ACM Symposium on Applied Computing (Melbourne, Florida, USA, March 2003).

* Corresponding author.
*Email addresses:* ino@ist.osaka-u.ac.jp (Fumihiko Ino), hagihara@ist.osaka-u.ac.jp (Kenichi Hagihara).

[1] Presently with Matsushita Electric Industrial Co., Ltd.

## 1 Introduction

*Volume rendering* [5,6] is an intuitive technique for understanding large amounts of three-dimensional (3-D) data sets, or volumes. For example, the rendering technique helps us visualize scientific volumes created by numerical computations and clinical volumes created by X-ray computed tomography (CT) scans. Furthermore, some high speed commercial renderers such as Maya [1] and RenderMan [19] are also useful to create outstanding visual effects for films. However, to render these enlarging volumes at interactive rates, we require adequate computing resources such as fast processors and large memories. One good solution to meet these requirements is to parallelize sequential volume rendering algorithms on distributed memory multiprocessors.

Using distributed memory multiprocessors, many researchers have developed parallel volume rendering methods [4,8,9,12,17,20,21,25]. Most of existing methods are *sort-last* methods [15], which partition the entire volume into subvolumes and distribute them to processors. In addition to this data distribution phase, sort-last methods consist of two phases: the rendering phase and the compositing phase. Each processor produces a subimage by rendering its assigned subvolume, then composites the final image by merging the produced subimages into one. Thus, all processors operate independently until the compositing phase. Therefore, by exploiting the data parallelism in the compute-intensive rendering phase, sort-last methods provide high performance rendering on distributed memory multiprocessors.

For the rendering phase, we can use efficient algorithms such as *segmented ray casting* (SRC) [8] and *shear-warp factorization* [9]. The SRC method, which produces realistic images without warping, suits for medical diagnosis. However, for the compositing phase, we still have to develop more efficient compositing algorithms, because the image compositing becomes the performance bottleneck of sort-last methods as the number of processors increases.

The *binary-swap* (BS) method [12] is an efficient and simple compositing algorithm, which repeatedly splits the subimages and distributes them to the appropriate processors. Many sort-last systems [14,16,20,24,25] have used this method, because it provides an efficient compositing with less implementation effort compared with others: *projection* [4], *direct send* [8,17], and *parallel pipeline* [10].

In [25], Yang et al. have incorporated run-length encoding into the BS method. Their method, the binary-swap with bounding rectangle and run-length encoding (BSBRC) method, has showed better performance than the original BS method on the IBM SP2. They also challenged to ensure static load-balancing by splitting the image plane into interleaved regions. However, this method, the binary-swap with static load-balancing and run-length encoding (BSLC) method, failed to outper-

form the BSBRC method.

In this paper, to develop an efficient compositing method, we present an improvement on the BS method, the binary-swap with static load-balancing, multiple bounding rectangle, and run-length encoding (BSLMBRC) method. We have incorporated three acceleration techniques into the original BS method. First, to ensure static load-balancing, we split the image plane into interleaved regions and assign them to processors like the BSLC method, while the original does into two-dimensional (2-D) block regions. Second, to avoid more redundant computations, we restrict the image compositing to specific regions by the multiple bounding rectangle, while the original does by the single bounding rectangle. Last, to reduce communication costs, we transmit run-length encoded pixels like the BSBRC method, while the original optionally does LZRW1 [22] encoded pixels.

The rest of this paper is organized as follows. Section 2 briefly describes sort-last parallel volume rendering. Section 3 describes the details of the BSLMBRC method and presents a theoretical analysis on its performance. Section 4 presents some experimental results on a PC cluster [3] of 64 nodes. At last, Section 5 concludes this paper.

## 2 Sort-last parallel rendering

Sort-last parallel rendering [15], which Molnar et al. have classified, is a broad class of parallel rendering methods. They have regarded the rendering problem as a problem of sorting objects to the screen and have classified parallel rendering methods, based on where the sort from object space to screen space occurs.

Fig. 1 shows an overview of sort-last parallel volume rendering. In the following, let $n$ be the number of processors. The entire volume is partitioned into subvolumes and the partitioned subvolumes are distributed to each processor. In the rendering phase, each processor independently produces a subimage by rendering its own subvolume. Thus, $n$-subimages have been produced at the end of the rendering phase. In the subsequent compositing phase, the processors produce the final image by merging $n$-subimages in a back-to-front [23] or front-to-back [11] order.

Thus, sort-last methods aim at exploiting the data parallelism in the compute-intensive rendering phase, and this gives us high performance rendering on distributed memory multiprocessors. However, since processors have to communicate each other to produce the final image, the image compositing becomes a performance bottleneck of the volume rendering with the increase of $n$. Therefore, to develop an efficient sort-last method that gives a linear speedup with $n$, we also have to develop an efficient compositing method that provides high performance compositing with the increase of $n$.

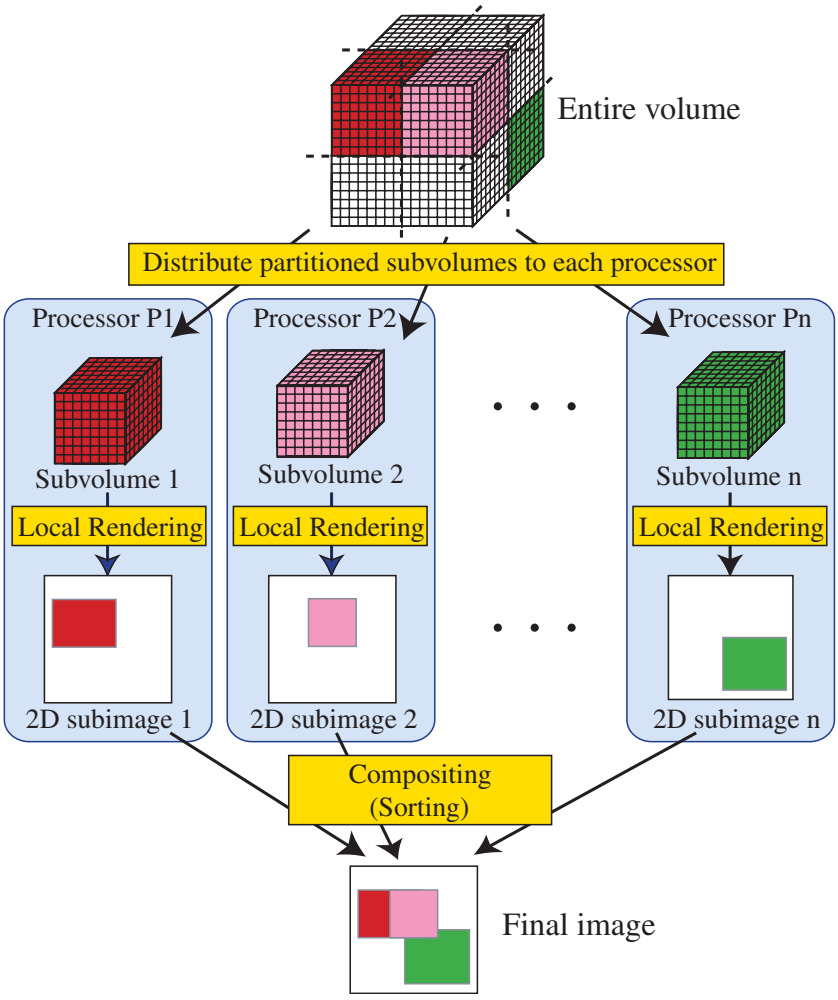



Fig. 1. Sort-last parallel volume rendering.

## 3 An improvement on binary-swap compositing

This section describes the details of our BSLMBRC method and presents a theoretical analysis on its performance. We first describe the BS method, the base of our BSLMBRC.

### *3.1 Binary-Swap Compositing*

The binary-swap (BS) method [12] merges all produced subimages into the final image as shown in Fig. 2(a). The key idea is that splitting the subimages and swapping them between processors exploits more parallelism. At each compositing stage, all processors are paired up, and the two processors involved in a compositing split the image plane into two pieces. Each processor then takes responsibility for one of the two pieces and swaps the other piece. This method requires exactly $\log n$

Table 1
Existing compositing methods based on BS method. RLE means run-length encoding. Original BS method uses LZRW1 encoding as an option.

| Method | Load balancing | Bounding rectangle | Data compression |
|---|---|---|---|
| BS [12] | — | Single | — / LZRW1 |
| BSBRC [25] | — | Single | RLE |
| BSLC [25] | Static | — | RLE |
| *BSLMBRC* | Static | Multiple | RLE |

compositing stages, and every processor participates in all compositing stages.

Notice that only non-blank pixels affect the composited results. Thus, the BS method exploits the sparsity of the subimage by using a bounding rectangle, which encloses the entire non-blank region of the subimage. Each processor composites and transmits inside the bounding rectangle.

Given an $A$ pixel subimage, determining its bounding rectangle takes $O(A)$ time. On the other hand, given the coordinates of two bounding rectangles, merging the two takes $O(1)$ time. Therefore, once we have determined all bounding rectangles at the initial compositing stage, it takes only $O(1)$ time to update the bounding rectangle of the composited subimage at each following stage. Thus, the bounding rectangle is an efficient technique to reduce redundant computations and communications.

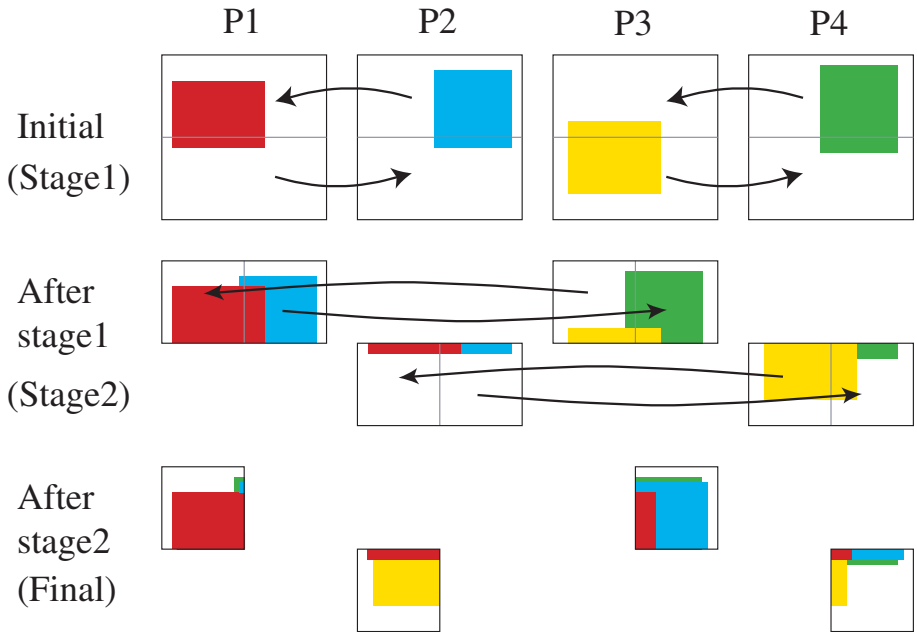
### *3.2 Proposed BSLMBRC compositing*

Our BSLMBRC method has the same rule of subimage exchange as the BS method. Table 1 summarizes the differences among the BS based methods.

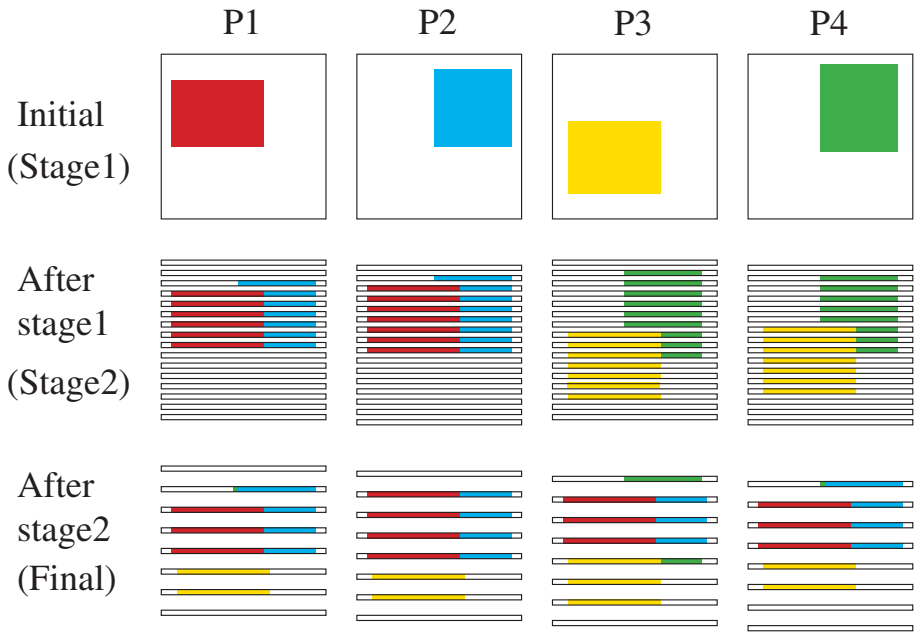
The acceleration techniques incorporated into our BSLMBRC method are the following three.

**(1) Interleaved splitting:** To ensure static load-balancing, we split the image plane into interleaved regions and assign them alternately to processors, while the original does into 2-D block regions (see Fig. 2). This technique enables processors to exchange almost equal number of pixels.

**(2) Multiple bounding rectangle:** To avoid more redundant computations, we restrict the image compositing to specific regions by the multiple bounding rectangle, while the original does by the single bounding rectangle. So, processors composites and transmits only inside the multiple bounding rectangle, which allows the processors to cover all non-blank pixels with less blank pixels by applying bounding rectangles to each scanline (see Fig. 3).




(a) BS

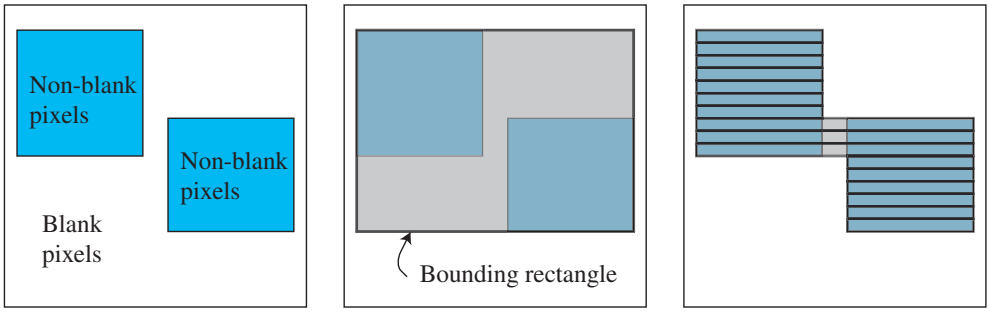



(b) BSLMBRC

Fig. 2. Compositing processes of BS method (upside) and BSLMBRC method (downside) using four processors. BSLMBRC uses interleaved splitting while BS uses block splitting.

**(3) Run-length encoding:** To reduce communication costs, we transmit run-length encoded pixels, while the original optionally does LZRW1 encoded pixels. Note that processors have to encode only the pixels inside the multiple bounding rectangle to compress the sequence of blank pixels.

In the following, we introduce why our method adopts the above three acceleration




(a) Target image (b) Single (c) Multiple

Fig. 3. Single bounding rectangle (center) and multiple bounding rectangle (right).

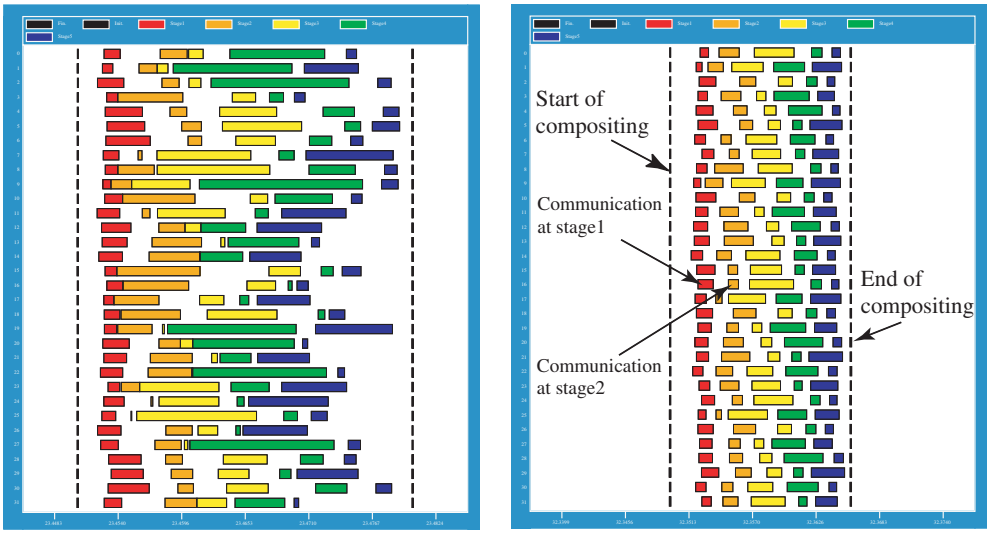
techniques.

First, the original BS method accelerates the image compositing by using the bounding rectangle, but the bounding rectangle brings another issue: the load imbalance. The load imbalance in the BS method can take place in all pairs of processors. In Fig. 2(a), processor P1 and P2 exchange their splitted subimages, and P2 sends many pixels compared to P1. This load imbalance is caused by the block splitting of the BS method, in which the number of pixels that a processor sends ranges from 0 to $B$ pixels, where $B$ represents the number of pixels in the bounding rectangle that the processor has. Therefore, to assign half the bounding rectangle to each processor, we split the image plane into interleaved regions as Yang et al. does in [25]. By using the interleaved splitting, the processor sends approximately $B/2$ pixels.

Notice that such load imbalances take place especially at an early stage, because subimages at the stage are relatively sparse to the splitted regions. Furthermore, the BS method varies the pairs of processors at every stage. Therefore, once a load imbalance takes place at an early stage, the load-imbalanced pair causes significant delay at the following stages, and the delay can spread among all processors. Fig. 4(a) shows this situation. In Fig. 4(a), we see that some horizontal bars adjoin their succeeding bars. That is, at those compositing stages, the processors that show such bars perform few calculations and thereby can cause load imbalances between their partners. We also see that the spread of the delay make the adjoined bars relatively long compared to others.

Second, the original BS method avoids redundant computations by using the bounding rectangle. On the other hand, as mentioned in Section 1, the BSLC method, which ensures static load-balancing but unuses the bounding rectangle, failed to outperform the BSBRC method, which uses the bounding rectangle. Therefore, we think that exploiting more sparsity of the subimages is necessary for high performance compositing. To do this, we use the multiple bounding rectangle for our method. That is, we apply a bounding rectangle to each scanline as shown in Fig. 3(c).

Given an $A$ pixel square subimage, the multiple bounding rectangle takes the same




(a) BS (b) BSLMBRC

Fig. 4. Timeline views of BS method (left) and BSLMBRC (right) method using 32 processors. Time is along horizontal axis and processors are along vertical axis. Each horizontal bar corresponds to a communication occurred in a processor.

$O(A)$ time as the single bounding rectangle to determine itself. The multiple bounding rectangle also takes $O(\sqrt{A})$ time to update itself, while the single bounding rectangle takes $O(1)$ time. However, this additional cost is relatively low compared to the total cost of the image compositing (see Section 4.3).

Last, to reduce communication costs, we compress the scanline data by run-length encoding. Although the multiple bounding rectangle can exclude many blank pixels, the blank pixels inside the rectangle remain. For example, a sparse image like a doughnut has many blank pixels inside its circle. To exclude such blank pixels at low cost, we compress only blank pixels and leave non-blank pixels as Yang et al. does in [25].

Run-length encoding and bounding rectangle are similar techniques in terms of excluding blank pixels, but run-length encoding takes more time. At every compositing stage, it takes $O(A)$ time to encode a composited subimage. Therefore, applying the bounding rectangle before run-length encoding is necessary for efficient compositing. We discuss this cost-benefit tradeoff later in Section 4.3.

### *3.3 Theoretical analysis of BSLMBRC*

In this section, we estimate the compositing time of the BSLMBRC method, $T_{\text{all}}$. To do this, we first estimate the total transmitted pixels per processor, $N_{\text{all}}$, then define $T_{\text{all}}$ as a function of $N_{\text{all}}$.

Let $p$ be the number of non-blank pixels in the final image. Let $P_k$ be the number of

non-blank pixels per processor at the end of $k$-th stage, where $k \geq 0$. In a block data distribution, each processor has approximately $p \cdot n^{-2/3}$ non-blank pixels at the end of the rendering phase [17], so that $P_0 = p \cdot n^{-2/3}$.

Let $R_k$ also be the account rate of non-blank pixels in the splitted subimage at the end of $k$-th stage, where $k \geq 0$. That is, if the splitted subimage at the $k$-th stage is filled with blank pixels, we have $R_k = 0$, and if it is filled with non-blank pixels, $R_k = 1$. At the end of $k$-th stage, where $k \geq 0$, each processor is assigned a splitted image of $p/2^k$ pixels, which has $P_k$ non-blank pixels. Therefore,

$$R_k = \frac{2^k}{p} \cdot P_k, \tag{1}$$

where $k \geq 0$.

At the exchange of $k$-th stage, where $k \geq 1$, each processor receive $P_{k-1}/2$ non-blank pixels. Each processor then merges the received pixels with its remained subimage, which has $P_{k-1}/2$ non-blank pixels. At this merge, each of the received non-blank pixels is merged with blank pixels at the rate of $1 - R_{k-1}$. That is, $(1 - R_{k-1}) \cdot P_{k-1}/2$ blank pixels turns into non-blank after this merge. Therefore, $P_k = P_{k-1}/2 + (1 - R_{k-1}) \cdot P_{k-1}/2 = P_{k-1} \cdot (1 - R_{k-1}/2)$. Using Eq. (1), we obtain:

$$P_k = \begin{cases} p \cdot n^{-\frac{2}{3}}, & \text{for} \quad k = 0 \\ P_{k-1} \cdot \left(1 - \dfrac{2^{k-2}}{p} \cdot P_{k-1}\right), & \text{for} \quad k \geq 1 \end{cases}. \tag{2}$$

Summing up half the number of pixels over all stage:

$$N_{\text{all}} = \sum_{k=1}^{\log n} \frac{P_{k-1}}{2}. \tag{3}$$

In Section 4.4, we show a verification of $N_{\text{all}}$.

We next define $T_{\text{all}}$ using $N_{\text{all}}$. The compositing time of BSLMBRC, $T_{\text{all}}$, consists of five major costs as follows (see also Fig. 5).

- $T_{\text{cpy}}$: the copy cost, defined as the time to copy pixels into send buffer with run-length encoding.
- $T_{\text{syn}}$: the synchronization cost, defined as the time to synchronize with the partner processor before an exchange.
- $T_{\text{com}}$: the communication cost, defined as the time to exchange pixels between a pair of processors through the network.
- $T_{\text{cal}}$: the calculation cost, defined as the time to composite the received pixels with run-length decoding.

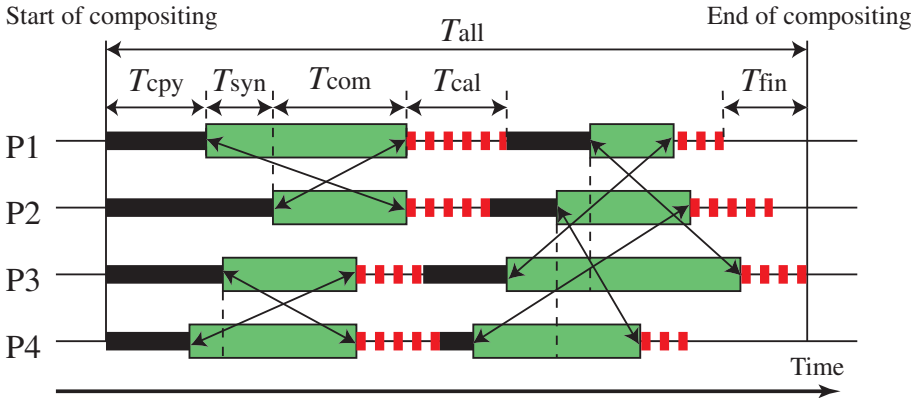



Fig. 5. Breakdown of compositing time on four processors.

- $T_{fin}$: the finalization cost, defined as the time to wait for the completion of the latest processor after the final compositing stage.

Three of the above five costs, $T_{cpy}$, $T_{com}$ and $T_{cal}$, are proportional to $N_{all}$. The rest of costs, $T_{syn}$ and $T_{fin}$, depend on the effect of the load-balancing. If we have perfect load-balancing, $T_{syn}$ and $T_{fin}$ can be approximated as zero.

Summarizing the above discussions, we have

$$\begin{aligned} T_{all} &= T_{cpy} + T_{syn} + T_{com} + T_{cal} + T_{fin} \\ &= (t_{cpy} + t_{com} + t_{cal}) \cdot N_{all} + T_{syn} + T_{fin}, \end{aligned} \tag{4}$$

where $t_{cpy}$, $t_{com}$ and $t_{cal}$ are the execution time per pixel to copy, exchange, and composite, respectively.

## 4 Experimental results

In this section, we present some experimental results using the BS based methods listed in Table 1. The results contain (1) a comparison of the compositing times using clinical volumes, (2) a discussion on the effect of the three acceleration techniques presented in Section 3.2, and (3) a verification of the theoretical analysis presented in Section 3.3.

### *4.1 Experimental environments*

Fig. 6 shows the hierarchical organization of hardware/software configuration. We used a Linux PC cluster [3] of 64 nodes for the experiments. Each node in the cluster has two Pentium III 1GHz processors and connects to a Myrinet-2000 switch

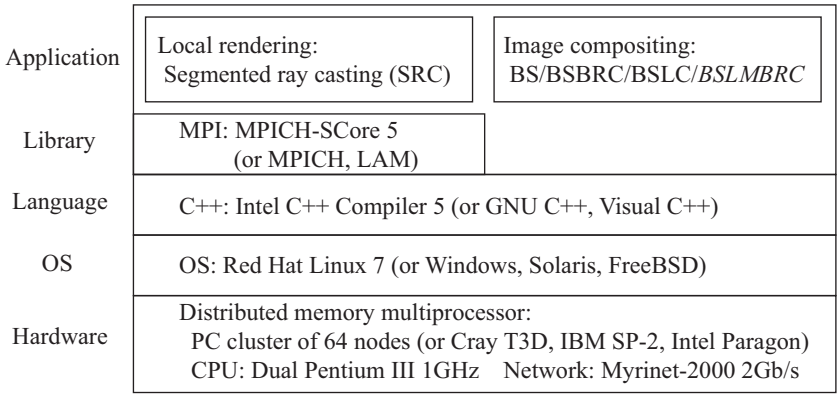



Fig. 6. Hierarchical organization of hardware/software configuration.

[2], which provides bandwidth of 2Gb/s.

We have implemented all the BS based methods listed in Table 1 by using Intel C++ compiler and MPICH-SCore library [18], which is a fast implementation of the Message Passing Interface (MPI) standard [13]. We also have implemented the SRC method [8] for the rendering phase. Thus, our implementation runs on many platforms where MPI and C++ programs are available.

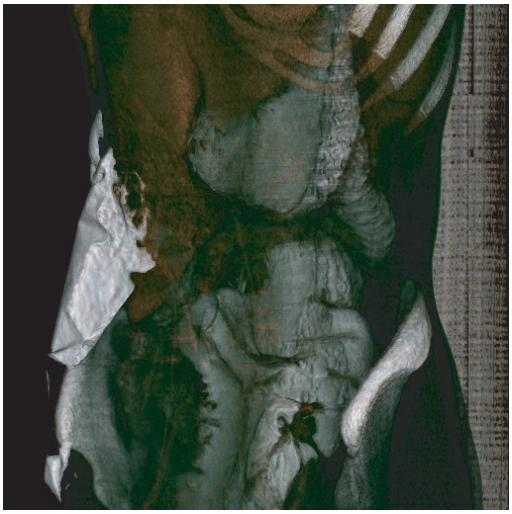
Fig. 7 shows the rendered images from volume data sets used in our experiments. The belly volume $VD_1$ and the skull volume $VD_2$ are created by an X-ray CT scan, and the cube volume $VD_3$ is created by hand.

### *4.2 Measured compositing time*

We measured the compositing times of the BS based methods: BS (without LZRW1 encoding), BSBRC, BSLC, and BSLMBRC. To measure the compositing times, we rotated the view point around the volume objects and rendered them on the screen at $512 \times 512$ pixel.

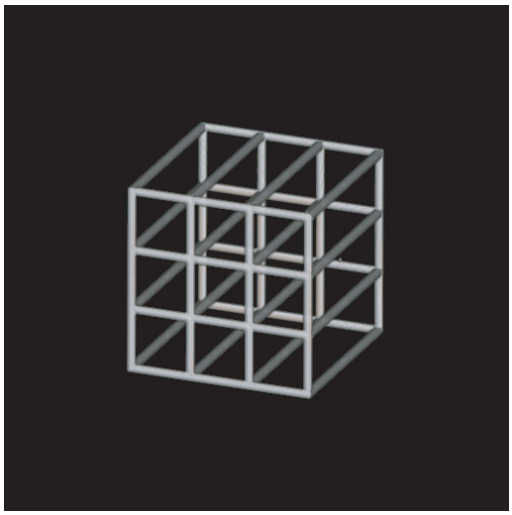
Fig. 8 shows the averaged results. In all averaged results, the BSLMBRC method is the fastest among the four methods. But for some view points when $n = 2$, the BS and BSBRC methods, which use the block splitting, show better performance than the BSLMBRC method, which uses the interleaved splitting. Since the volume is partitioned into blocks, the block splitting can avoid any communications for some view points. For example, when $n = 2$, given a view point that locates on the dividing plane of the volume, processors can composite without any communications.

For the volumes $VD_1$ and $VD_2$, the BS method is the slowest and the BSBRC and BSLC methods show similar performance. The speedup of BSLMBRC to the other methods increases with $n$, ranges from 1.0 to 1.3, when $n = 2$, and from 1.5 to 2.5, when $n = 64$. Thus, with the increase of $n$, the BSLMBRC method provides better



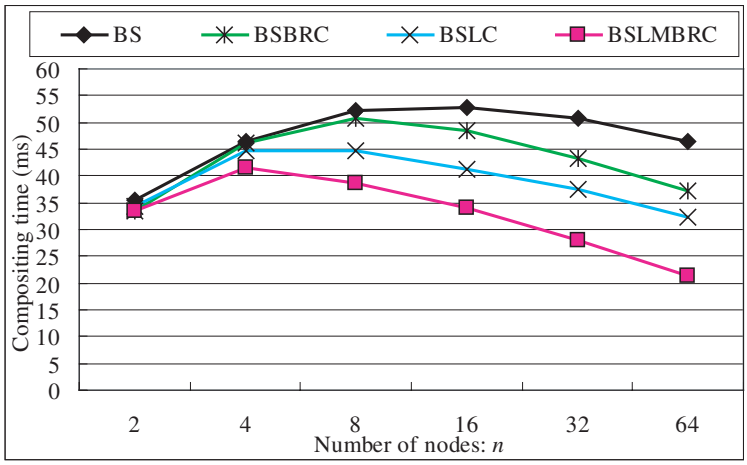
(a) $VD_1$: Belly $512 \times 512 \times 730$ voxel

(b) $VD_2$: Skull $512 \times 512 \times 448$ voxel

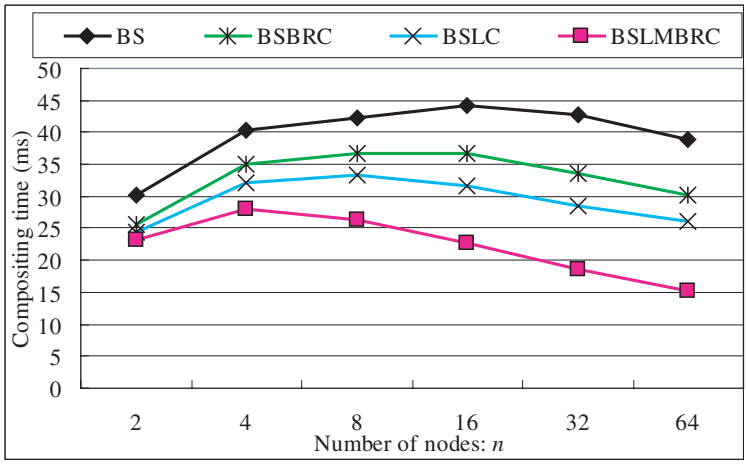


(c) $VD_3$: Cube $256 \times 256 \times 256$ voxel

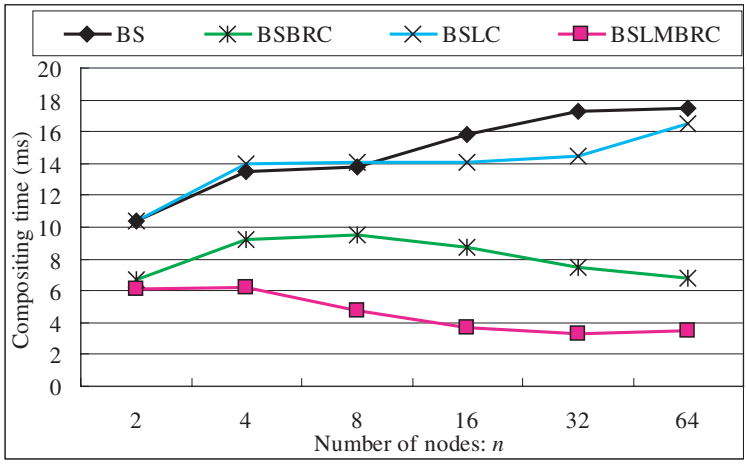
Fig. 7. Rendered images from volume data sets used in our experiments. Screen size is $512 \times 512$ pixel.




(a) $VD_1$: Belly $512 \times 512 \times 730$ voxel




(b) $VD_2$: Skull $512 \times 512 \times 448$ voxel




(c) $VD_3$: Cube $256 \times 256 \times 256$ voxel

Fig. 8. Measured compositing times on a PC cluster with Myrinet-2000 network.

performance than the other BS based methods.

For the volume $VD_3$, which is small compared to the screen and sparse compared to the other volumes, the BSLC method is relatively slow. The BSLC method unuses the bounding rectangle, while the other methods use. Therefore, at every compositing stage, the BSLC method scans all pixels in the splitted subimage, while the others scan only inside the bounding rectangle. The total time to scan all pixels in the splitted subimage is proportional to $A \cdot \sum_{k=1}^{\log n} 1/2^k = A \cdot (1-1/n)$. As a result, the BSLC method takes long time to exclude the blank pixels, so that its compositing time increases with $n$.

In Fig. 8(c), the BS method also increases its compositing time with $n$. For the small volume, the block splitting can easily decrease the parallel efficiency compared to the interleaved splitting. For example, when we select a view point such that the rendered cube locates in one block region on the screen, only a few processors participate the compositing in the block splitting. For such view points, increasing the number of processors only increases the number of the compositing stages.

Summarizing the above results, both the bounding rectangle and interleaved splitting techniques are necessary to achieve high performance compositing for the small and sparse volumes like the volume $VD_3$. We think that the achieved performance is high enough for our target application, or volume rendering for clinical 3-D CT images, which requires realtime manipulation of images such as rotation, shift, and magnification at 10 frames per second (fps). That is, the compositing time below 40 ms is short enough for 10 fps, since the rendering algorithm can take the remaining 60 ms for producing subimages. On the other hand, in Fig. 8(a), all methods except the BSLMBRC method fail to yield a video quality rate of 30 fps, since the methods take more than 33 ms for compositing. Thus, reducing the compositing time is necessary to increase the upper limit of the frame rate. We also think that the BSLMBRC method, which achieves high performance compositing for sparse volumes, is suitable for clinical applications, because such applications are frequently required to produce sparse images such as the blood vessels in CT images.

### *4.3 Discussion on BS acceleration techniques*

We now discuss on the effect of the three acceleration techniques. Table 2 shows the breakdowns of the compositing times measured for the volume $VD_1$, where $n = 64$. To illustrate the effect in clear, we measured additional three methods, BSLBR, BSLBRC, and BSLMBR, which are the subsets of the BSLMBRC method.

First, the effect of static load-balancing appears at $T_{\mathrm{syn}}$ and $T_{\mathrm{fin}}$ in Table 2. The load-imbalanced methods, BS and BSBRC, show $T_{\mathrm{syn}} > 15$ and $T_{\mathrm{fin}} > 5$, while the other load-balanced methods show $T_{\mathrm{syn}} < 10$ and $T_{\mathrm{fin}} < 2$. That is, the interleaved

Table 2
Breakdowns of compositing times measured for volume $VD_1$ using 64 nodes. RLE means run-length encoding.

| Method | Load balancing | Bounding rectangle | Data compression | Measured times (ms) | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | $T_{cpy}$ | $T_{syn}$ | $T_{com}$ | $T_{cal}$ | $T_{fin}$ | $T_{all}$ |
| BS [12] | — | Single | — | 3.0 | 18.2 | 9.7 | 7.1 | 6.4 | 46.8 |
| BSBRC [25] | — | Single | RLE | 4.4 | 16.4 | 5.7 | 3.8 | 5.1 | 38.0 |
| BSLC [25] | Static | — | RLE | 14.4 | 5.9 | 5.7 | 3.4 | 1.0 | 32.5 |
| BSLBR | Static | Single | — | 2.7 | 9.0 | 9.7 | 6.8 | 1.8 | 32.6 |
| BSLBRC | Static | Single | RLE | 4.3 | 6.5 | 5.6 | 3.4 | 1.1 | 23.5 |
| BSLMBR | Static | Multiple | — | 2.8 | 6.4 | 6.2 | 3.9 | 1.2 | 23.1 |
| *BSLMBRC* | Static | Multiple | RLE | 3.1 | 6.2 | 5.6 | 3.6 | 1.1 | 22.0 |

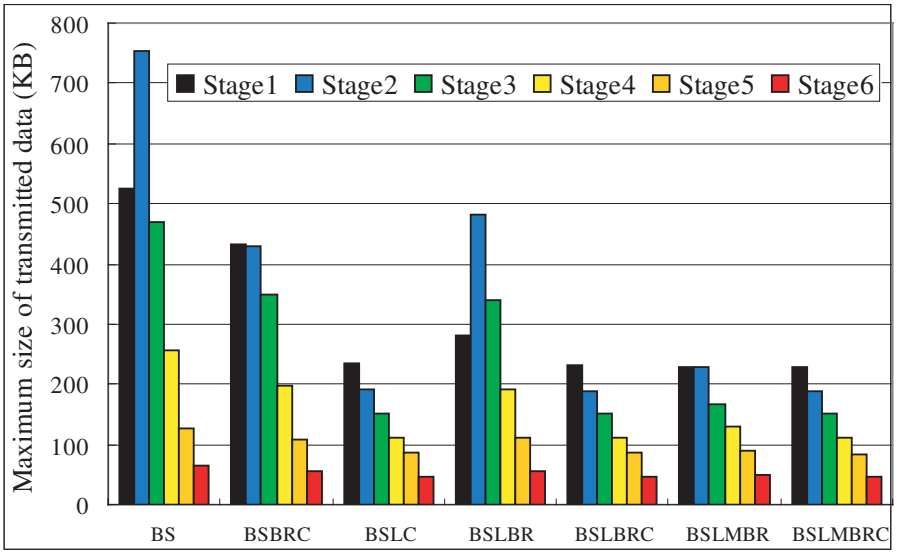
splitting balances the compositing workload among processors and decreases the synchronization and finalization costs. By comparing the BS and BSLBR methods, we see that the interleaved splitting reduces the compositing time by 30% (from 46.8 to 32.6 ms).

Second, the effect of the multiple bounding rectangle appears at all costs except $T_{cpy}$. By comparing the BSLBR and BSLMBR methods, we see that the multiple bounding rectangle decreases all costs except $T_{cpy}$ but slightly increases $T_{cpy}$. That is, the multiple bounding rectangle avoids more redundant computations and communications compared to the single but requires a little time to copy the pixels into send buffer by each scanline. The compositing time is reduced by further 29% (from 32.6 to 23.1 ms).

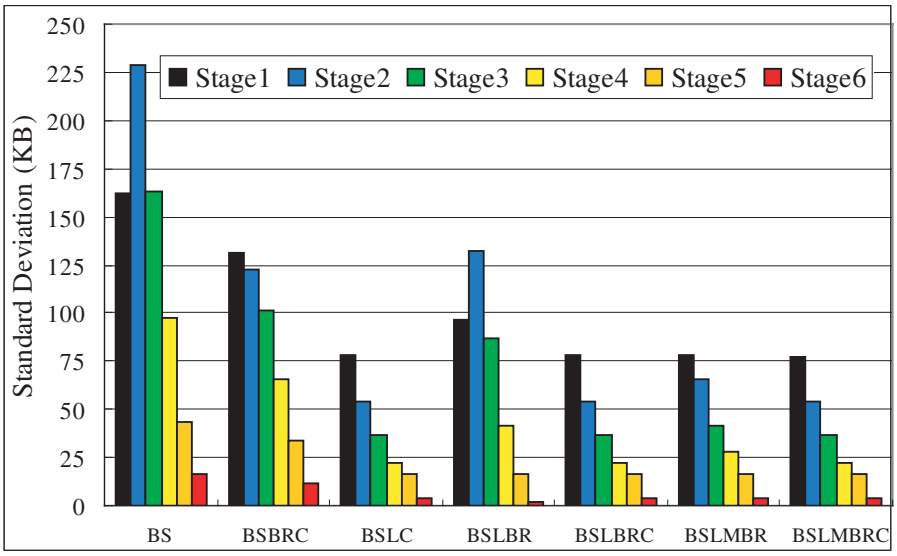
Notice that the update cost of the multiple bounding rectangle, included in $T_{cal}$, is unrevealed in this data set. The update cost is relatively low compared to the compositing cost, so that the decrease of the compositing cost hides the increase of the update cost.

Last, the effect of data compression appears at all costs except $T_{cpy}$. Run-length encoding decreases all costs except $T_{cpy}$ but increases $T_{cpy}$ like the multiple bounding rectangle. By comparing the BSLMBR and BSLMBRC methods, we see that run-length encoding reduces the compositing time by 5% (from 23.1 to 22.0 ms).

As mentioned in Section 3.2, we have a cost-benefit tradeoff between the multiple bounding rectangle and run-length encoding. Therefore, the effect of the multiple bounding rectangle appears in an opposite situation between BSLBRC and BSLMBRC. The multiple bounding rectangle decreases $T_{cpy}$. That is, in the BSLBRC method, adding run-length encoding to BSLBR decreases all costs except $T_{cpy}$ but increases $T_{cpy}$ from 2.7 to 4.3 ms, and in the BSLMBRC, adding the multiple bounding rectangle to BSLBRC decreases $T_{cpy}$ from 4.3 to 3.1 ms by excluding more blank pixels before applying run-length encoding. Thus, the multiple bounding rectangle is also useful to reduce $T_{cpy}$, when we use data compression technique. Without the bounding rectangle, $T_{cpy}$ is significantly increased due to the data com-




(a) Maximum size of transmitted data.



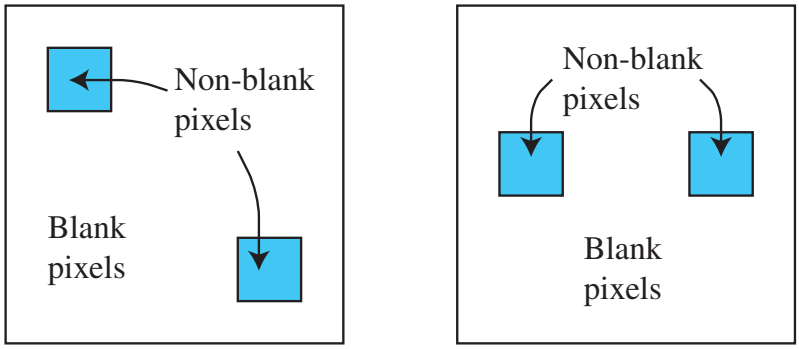

(b) Standard Deviation of transmitted data.

Fig. 9. Maximum size of transmitted data among nodes and its standard deviation.

pression technique as shown in the BSLC method (14.4 ms).

However, more high-rate compression algorithms such as LZRW1 [22] and zlib [7] are effective on low-speed network. For example, when we use zlib compression library with the BSLMBRC method, the compositing time for the volume $VD_1$ decreases by 18% compared to run-length encoding. Although $T_{\mathrm{cpy}}$ increases to 380 ms, the number of transmitted pixels decreases by 30%, thereby shows better performance.

Fig. 9 shows the maximum size of transmitted data among nodes and its standard deviation. At the first compositing stage, the BS and BSBRC methods, which unuse




(a) Horizontally unaligned case (b) Horizontally aligned case

Fig. 10. Examples of subimage at first compositing stage.

the interleaved splitting, exchange twice data compared to the others and show imbalanced loads. At the second compositing stage, the BSLBR method, which shows relatively good load-balancing at the first stage, exchange twice data like the BS and BSBRC methods. The BSLBR method unuses the data compression, so that many blank pixels inside the bounding rectangle are transmitted. Therefore, once a bounding rectangle enlarges as shown in Fig. 3, the BSLBR method exchanges many blank pixels at the following stages. Notice that BSLMBR avoid this situation by using the multiple bounding rectangle.

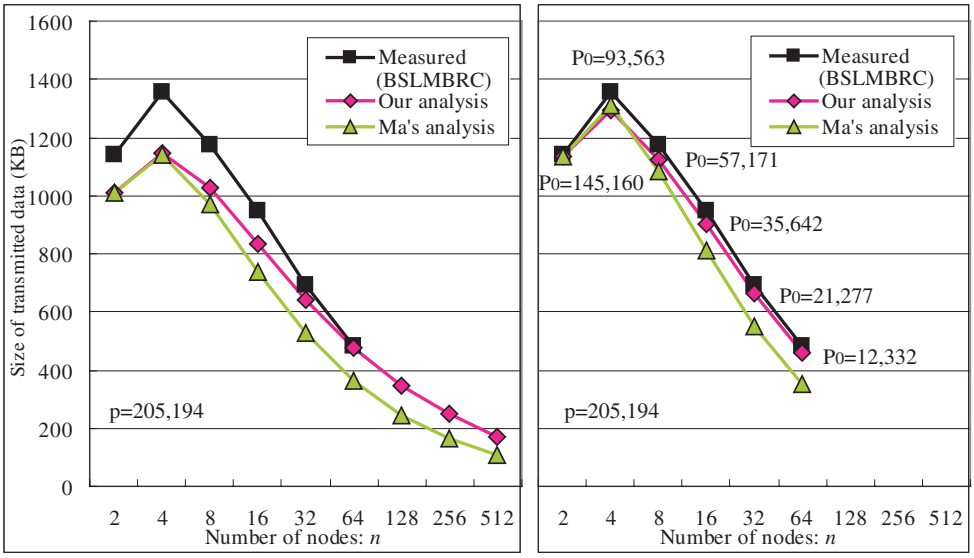
Fig. 10 shows two cases that can occur at the first compositing stage. One is the best case for the multiple bounding rectangle and the other is the worst. In the best case (Fig. 10(a)), the multiple bounding rectangle exploits almost all sparsity of the image, so that run-length encoding exploits a little. In the worst case (Fig. 10(b)), as same as the single bounding rectangle, the multiple bounding rectangle fails to exclude the blank pixels located between the non-blank regions. In this case, run-length encoding exploits further.

Summarizing the above discussions, all the three acceleration techniques are necessary for high performance compositing on high-speed network, and applying the multiple bounding rectangle before run-length encoding is necessary. Data compression is necessary especially for low-speed network such as Fast Ethernet.

### *4.4 Verification of theoretical analysis*

In this section, we verify our theoretical analysis presented in Section 3.3. To do this, we compare our analysis with the measured results and also with Ma's analysis [12].

Fig. 11 shows two comparisons between the theoretical size and the measured size of transmitted data. We used two values for $P_0$, one is $p \cdot n^{-2/3}$ in Eq. (2) and the other is the measured value. The theoretical number is calculated by $N_{\text{all}} \cdot 16$ bytes




(a) With $P_0 = p \cdot n^{-2/3}$ (b) With measured $P_0$

Fig. 11. Theoretical and Measured size of transmitted data for volume $VD_1$. Ma's analysis uses $N_{\text{all}} = P_0/2 \cdot \sum_{k=0}^{\log n-1} 2^{-k/3}$.

(12 bytes for RGB colors and 4 bytes for opacity).

When we use the measured value for $P_0$, the error of our analysis is at most 5%. On the other hand, the gap between Ma's analysis and the measured size spreads as $n$ increases and results in the maximum error of 38%. The difference between the analyses exists in how it reduces the total number of non-blank pixels at each compositing stage. In Ma's analysis, for any value of $n$, non-blank pixels reduce in half over three compositing stages. For large value of $n$, this analysis fails to decrease the reduction rate of non-blank pixels, which we have modeled as $1 - R_{k-1}$ in Eq. (2). Therefore, Ma's analysis estimates smaller number of pixels as $n$ increases.

## 5 Conclusions

We presented an image compositing method, which accelerates the BS method on distributed memory multiprocessors. Our BSLMBRC method has incorporated three acceleration techniques into the original BS method. The interleaved splitting ensures static load-balancing and reduces the significant costs for synchronization and finalization. Both the multiple bounding rectangle and run-length encoding exploits the sparsity of the image, and applying the multiple bounding rectangle before run-length encoding is required for achieving high performance compositing on high-speed network. By adding the above three techniques to the original BS method, our method accelerates its performance as the number of processors increases, and its speedup to the original reaches the maximum of 2.5.

## Acknowledgements

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (C)(2) (14580374), JSPS Research for the Future Program JSPS-RFTF99I00903, and Network Development Laboratories, NEC. We would like to thank Masaki Miyamoto at Medical Imaging Laboratory Co., Ltd., Yoshinobu Sato, Shinichi Tamura, and Takahiro Ochi at Osaka University for many useful discussions. We are also grateful to the anonymous reviewers for their valuable comments.

## References

[1] Alias|wavefront, Maya, `http://www.aliaswavefront.com/` (2003).

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W.-K. Su, Myrinet: A gigabit-per-second local-area network, IEEE Micro 15 (1) (1995) 29–36.

[3] R. Buyya (Ed.), High Performance Cluster Computing, Prentice Hall PTR, 1999.

[4] E. Camahort, I. Chakravarty, Integrating volume data analysis and rendering on distributed memory architectures, in: Proceedings of the 1993 Parallel Rendering Symposium (PRS'93), 1993, pp. 89–96.

[5] T. W. Crockett, An introduction to parallel rendering, Parallel Computing 23 (7) (1997) 819–843.

[6] R. A. Drebin, L. Carpenter, P. Hanrahan, Volume rendering, Computer Graphics (Proceedings of SIGGRAPH'88) 22 (3) (1988) 65–74.

[7] J.-L. Gailly, M. Adler, Zlib general purpose compression library, `http://www.gzip.org/zlib/` (2003).

[8] W. M. Hsu, Segmented ray casting for data parallel volume rendering, in: Proceedings of the 1993 Parallel Rendering Symposium (PRS'93), 1993, pp. 7–14.

[9] P. Lacroute, M. Levoy, Fast volume rendering using a shear-warp factorization of the viewing transformation, in: Proceedings of SIGGRAPH '94, 1994, pp. 451–458.

[10] T.-Y. Lee, C. 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.

[11] M. Levoy, Display of surfaces from volume data, IEEE Computer Graphics and Applications 8 (3) (1988) 29–37.

[12] K.-L. Ma, J. S. Painter, C. D. Hansen, M. F. Krogh, Parallel volume rendering using binary-swap compositing, IEEE Computer Graphics and Applications 14 (4) (1994) 59–68.

[13] Message Passing Interface Forum, MPI: A message-passing interface standard, International Journal of Supercomputing Applications 8 (3/4) (1994) 159–416.

[14] T. Mitra, T. Chiueh, Implementation and evaluation of the parallel mesa library, in: Proceedings of the 1998 International Conference on Parallel and Distributed Systems (ICPADS'98), 1998, pp. 84–91.

[15] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, A sorting classification of parallel rendering, IEEE Computer Graphics and Applications 14 (4) (1994) 23–32.

[16] K. Moreland, B. Wylie, C. Pavlakos, Sort-last parallel rendering for viewing extremely large data sets on tile displays, in: Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (PVG'01), 2001, pp. 85–92.

[17] U. Neumann, Parallel volume-rendering algorithm performance on mesh-connected multicomputers, in: Proceedings of the 1993 Parallel Rendering Symposium (PRS'93), 1993, pp. 97–104.

[18] F. O'Carroll, H. Tezuka, A. Hori, Y. Ishikawa, The design and implementation of zero copy MPI using commodity hardware with a high performance network, in: Proceedings of the ACM International Conference on Supercomputing (ICS'98), 1998, pp. 243–250, `http://www.pccluster.org/`.

[19] PIXAR, RenderMan, `http://renderman.pixar.com/` (2003).

[20] K. Sano, H. Kitajima, H. Kobayashi, T. Nakamura, Parallel processing of the shear-warp factorization with the binary-swap method on a distributed-memory multiprocessor system, in: Proceedings of the 1997 Parallel Rendering Symposium (PRS'97), 1997, pp. 87–94.

[21] C. T. Silva, A. E. Kaufman, C. Pavlakos, PVR: High-performance volume rendering, IEEE Computational Science and Engineering Winter 1996 3 (4) (1996) 18–28.

[22] R. N. Williams, An extremely fast ziv-lempel data compression algorithm, in: Proceedings of the 1991 Data Compression Conference (DCC'91), 1991, pp. 362–371.

[23] L. Westover, Footprint evaluation for volume rendering, Computer Graphics (Proceedings of SIGGRAPH'90) 24 (4) (1990) 367–376.

[24] B. Wylie, C. Pavlakos, V. Lewis, K. Moreland, Scalable rendering on PC clusters, IEEE Computer Graphics and Applications 21 (4) (2001) 62–70.

[25] D.-L. Yang, J.-C. Yu, Y.-C. Chung, Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers, The Journal of Supercomputing 18 (2) (2001) 201–220.