An Improvement on Binary-Swap Compositing for Sort-Last Parallel Rendering

Akira Takeuchi†, Fumihiko Ino*, and Kenichi Hagihara*

† Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
takeuti@ics.es.osaka-u.ac.jp

* Department of Computer Science, Graduate School of Information Science and Technology, Osaka University 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{ino, hagihara}@ist.osaka-u.ac.jp

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. Through the use of the three techniques, our method balances the compositing load 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.

Keywords
Image Compositing, Parallel Rendering, Sort-last, Binary-swap, Cluster Computing

1. INTRODUCTION

Volume rendering [7, 16] is an intuitive technique in understanding large amounts of three-dimensional 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. 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 serial volume rendering algorithms on distributed memory multiprocessors.

Many researchers have developed parallel volume rendering methods [2, 4, 5, 8, 12, 14, 15, 19]. Most of existing methods are sort-last [11] methods, which partition the entire volume into subvolumes and distribute them to processors. 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 [4] (SRC) and shear-warp factorization [5]. 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 [8] (BS) method is an efficient and simple compositing algorithm, which repeatedly splits the subimages and distributes them to the appropriate processors. Many sort-last systems [10, 14, 18, 19] have used this method, because it provides an efficient compositing with less implementation effort compared with others: projection [2], direct send [4, 12], and parallel pipeline [6].

In [19], 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 (BSLRC) method, failed to outperform 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 block regions. Second, to avoid more redundant computations, we restrict the image compositing to specific regions by the multiple bounding rectangle. Last,
to reduce communication costs, we transmit run-length encoded pixels like the BSBRC method, while the original optionally does LZRW1 [17] encoded pixels.

The rest of this paper is organized as follows. Section 2 briefly describes a 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 of 64 nodes. At last, Section 5 concludes this paper.

2. SORT-LAST PARALLEL RENDERING

Sort-last [11] parallel rendering, 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.

Figure 1 shows an overview of a 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 [16] or front-to-back [7] order.

Thus, in sort-last methods, exploiting the data parallelism in the compute-intensive rendering phase 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 \( n \).

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 [8] (BS) method merges all produced subimages into the final image as shown in Figure 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 \( \log n \) 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Load balancing</th>
<th>Bounding rectangle</th>
<th>Data compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS [8]</td>
<td>—</td>
<td>single</td>
<td>—/LZRW1</td>
</tr>
<tr>
<td>BSBRC [19]</td>
<td>—</td>
<td>single</td>
<td>RLE</td>
</tr>
<tr>
<td>BSLC [19]</td>
<td>static</td>
<td>—</td>
<td>RLE</td>
</tr>
<tr>
<td>BSLMBRC</td>
<td>static</td>
<td>multiple</td>
<td>RLE</td>
</tr>
</tbody>
</table>

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 to processors, while the original does into two-dimensional block regions (see Figure 2).

(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 (see Figure 3).
(3) Run-length encoding: To reduce communication costs, we transmit run-length encoded pixels, while the original optionally does LZRW1 encoded pixels.

In the following, we introduce why our method adopts the above three acceleration 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. Figure 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 \(B\) is 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 [19]. By using the interleaved splitting, the processor sends approximately \(B/2\) pixels.

Notice that such load-imbances 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-imbanced pair causes significant delay at the following stages, and the delay can spread among all processors. In Figure 4(a), we see that some horizontal bars adjoins their succeeding bars. That is, at that compositing stages, the pro-
cessors that show such bars perform few calculation and thereby can cause load-imbances 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 BS-LC method, which ensures static load-balancing but unuses the bounding rectangle, failed to outperform the BS-

BRC 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 Figure 3(c).

Given an \(A\) pixel square subimage, the multiple bounding rect-
angle takes the same \(O(A)\) 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 [19].

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 composed 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 BSLM-BRC method, \( T_{all} \). To do this, we first estimate the total transmitted pixels per processor, \( N_{all} \), then define \( T_{all} \) as a function of \( N_{all} \).

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 \( p \cdot n \cdot 2/3 \) non-blank pixels at the end of the rendering phase [12], so that \( R_0 = p \cdot n \cdot 2/3 \).

Let \( R_k \) also be the account rate of non-blank pixels in the split subimage at the end of \( k \)-th stage, where \( k \geq 0 \). That is, if the split 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 split image of \( p/2^k \) pixels, which has \( P_k \) non-blank pixels. Therefore,

\[
R_k = \frac{2^k}{p} \cdot P_k, \quad (1)
\]

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 are 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 Equation (1), we obtain:

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

Summing up half the number of pixels over all stage:

\[
N_{all} = \sum_{k=1}^{n} \frac{P_{k-1}}{2}. \quad (3)
\]

In Section 4.4, we show a verification of \( N_{all} \).

We next define \( T_{all} \) using \( N_{all} \). The compositing time of BSLM-BRC, \( T_{all} \), consists of five major costs as follows.

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

\[
T_{all} = T_{cpy} + T_{syn} + T_{com} + T_{cal} + T_{fin} = (T_{cpy} + T_{com} + T_{cal}) \cdot N_{all} + T_{syn} + T_{fin}, \quad (4)
\]

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

We used a PC cluster of 64 nodes for the experiments. Each node in the cluster has two Pentium III 1GHz processors and connects to a Myrinet-2000 [1] switch, which provides bandwidth of 2Gbps.

We have implemented all the BS based methods using C++ language and MPI-SCI [13] library, which is a fast implementation of Message Passing Interface [9] (MPI). We also have implemented the SRC [4] method for the rendering phase.

Figure 5 shows the rendered images from volume data sets used in our experiments. The belly volume \( V_D \) and the skull volume \( V_D \) are created by a X-ray CT scan, and the cube volume \( V_D \) is created by hand.

#### 4.2 Measured Compositing Time

We measured the compositing times of the BS based methods: BS (without LZW1 encoding), BSBRC, BSLC, and BSLMBRC methods. 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.

Figure 6 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 \( V_D \) and \( V_D \), 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 \( n \). 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 performance than the other BS based methods.

For the volume \( V_D \), which is small compared to the screen and sparse compared to the other volumes, the BSLC method is
Compositing time (ms)

15
20
25
30
35
40
45
50
55
60

0
5
10
15
20
25
30
35
40
45

2 4 8 16 32 64
Number of nodes: \(n\)

Figure 6: Measured compositing times on a PC cluster with Myrinet-2000 network.

(a) \(V D_1\): Belly 512 \(\times\) 512 \(\times\) 730
(b) \(V D_2\): Skull 512 \(\times\) 512 \(\times\) 448
(c) \(V D_3\): Cube 256 \(\times\) 256 \(\times\) 256

4.3 Discussion on BS Acceleration Techniques

We now discuss the effect of the three acceleration techniques. Table 2 shows the breakdowns of the compositing times measured for the volume \(V D_1\), where \(n = 64\). To illustrate the effect in clear, we measured additional three methods, BSBRC, BSLMBRC, and BSLMBR, which are the subsets of the BSLMBRC method.

First, the effect of static load-balancing appears at \(T_{syn}\) and \(T_{fin}\) in Table 2. The load-imbalanced methods, BS and BSBRC, show \(T_{syn} > 15\) and \(T_{fin} > 5\), while the other load-balanced methods show \(T_{syn} < 10\) and \(T_{fin} < 2\). That is, the interleaved splitting balances the compositing load among processors and decreases the synchronization and finalization costs. By comparing the BS and BSBRC, 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, we see that the multiple bounding rectangle decreases the compositing time by 30% (from 46.8 to 32.6 ms).

Last, the effect of data compression appears at all costs except \(T_{cpy}\). Run-length encoding decreases all costs except \(T_{cpy}\) but increases \(T_{cpy}\). 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.

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 the BSLBRC and BSLMBRC. The multiple bounding rectangle decreases \(T_{cpy}\). That is, in the BSLBRC, adding run-length encoding to the 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 the 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 compression technique as shown in the BSLC (14.4 ms).

However, more high-rate compression algorithms such as LZRW1 [17] and zlib [3] 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 $T_{cpy}$ increases to 380 ms, the number of transmitted pixels decreases by 30%, thereby shows better performance.

Figure 7 shows the maximum size of transmitted data among nodes and its standard deviation. At the first compositing stage, the BS and BSBRC, which use the interleaved splitting, exchange twice data compared to the others and show imbalanced-loads. At the second compositing stage, the BSLBR, which shows relatively good load-balancing at the first stage, exchange twice data like the BS and BSBRC. The BSLBR method unuse the data compression, so that many blank pixels inside the bounding rectangle are transmitted. Therefore, once a bounding rectangle enlarges as shown in Figure 3, the BSLBR method exchanges many blank pixels inside the bounding rectangle. Notice that the BSLMBR avoid this situation by using the multiple bounding rectangle.

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.

![Figure 7: Maximum size of transmitted data among nodes and its standard deviation.](image)

![Figure 8: Theoretical and Measured size of transmitted data for volume $VD_1$, Ma’s analysis uses $N_{all}=P_0/2\sum_{k=0}^{n-1} 2^{-k/3}$.](image)

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 [8].

Figure 8 shows two comparisons between the theoretical size and the measured size of transmitted data. We used two values for $P_0$, one is $p \cdot n^{-4/3}$ in Equation (2) and the other is the measured value. The theoretical number is calculated by $N_{all}=16$ bytes (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 max-

---

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Load balancing</th>
<th>Bounding rectangle</th>
<th>Data compression</th>
<th>$T_{cpy}$</th>
<th>$T_{syn}$</th>
<th>$T_{com}$</th>
<th>$T_{cie}$</th>
<th>$T_{fan}$</th>
<th>$T_{del}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS [8]</td>
<td>—</td>
<td>single</td>
<td>—</td>
<td>3.0</td>
<td>18.2</td>
<td>9.7</td>
<td>7.1</td>
<td>6.4</td>
<td>46.8</td>
</tr>
<tr>
<td>BSBRC [19]</td>
<td>—</td>
<td>single</td>
<td>RLE</td>
<td>4.4</td>
<td>16.4</td>
<td>5.7</td>
<td>3.8</td>
<td>5.1</td>
<td>38.0</td>
</tr>
<tr>
<td>BSLC [19]</td>
<td>static</td>
<td>—</td>
<td>RLE</td>
<td>14.4</td>
<td>5.9</td>
<td>5.7</td>
<td>3.4</td>
<td>1.0</td>
<td>32.5</td>
</tr>
<tr>
<td>BSLBR</td>
<td>static</td>
<td>single</td>
<td>RLE</td>
<td>2.7</td>
<td>9.0</td>
<td>9.7</td>
<td>6.8</td>
<td>1.8</td>
<td>32.6</td>
</tr>
<tr>
<td>BSLBRC</td>
<td>static</td>
<td>single</td>
<td>—</td>
<td>4.3</td>
<td>6.5</td>
<td>5.6</td>
<td>3.4</td>
<td>1.1</td>
<td>23.5</td>
</tr>
<tr>
<td>BSLMBR</td>
<td>static</td>
<td>multiple</td>
<td>RLE</td>
<td>2.8</td>
<td>6.4</td>
<td>6.2</td>
<td>3.9</td>
<td>1.2</td>
<td>23.1</td>
</tr>
<tr>
<td>BSLMBRC</td>
<td>static</td>
<td>multiple</td>
<td>—</td>
<td>3.1</td>
<td>6.2</td>
<td>5.6</td>
<td>3.6</td>
<td>1.1</td>
<td>22.0</td>
</tr>
</tbody>
</table>
imum 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 Equation (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 BSLM-BRC 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-RFTF99100903, Kayamori Foundation of Informational Science Advancement, and Development Laboratories, NEC Networks. We would like to thank Masaki Miyamoto at Medical Imaging Laboratory, Yosaihobu 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.

6. REFERENCES