Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing

Hongfeng Yu* Chaoli Wang* Kwan-Liu Ma*

Department of Computer Science
University of California at Davis

ABSTRACT

The ever-increasing amounts of simulation data produced by scientists demand high-end parallel visualization capability. However, image compositing, which requires interprocessor communication, is often the bottleneck stage for parallel rendering of large volume data sets. Existing image compositing solutions either incur a large number of messages exchanged among processors (such as the direct send method), or limit the number of processors that can be effectively utilized (such as the binary swap method). We introduce a new image compositing algorithm, called *2-3 swap*, which combines the flexibility of the direct send method and the optimality of the binary swap method. The 2-3 swap algorithm allows an arbitrary number of processors to be used for compositing, and fully utilizes all participating processors throughout the course of the compositing. We experiment with this image compositing solution on a supercomputer with thousands of processors, and demonstrate its great flexibility as well as scalability.

1. INTRODUCTION

Leveraging the power of supercomputers, scientists can now simulate many things from galaxy interaction to molecular dynamics in unprecedented scales and details, leading to new scientific discoveries. Scientific simulations often produce data that are volumetric, time-varying, and multi-variate. The data may contain thousands of time steps with each time step having billions of voxels and each voxel recording dozens of variables. The vast amounts of data produced by these simulations, ranging from tens to hundreds of terabytes, demand a high-resolution, high-quality, and high-performance visualization capability so that scientists are able to test their hypotheses and discover insights in a timely and successful manner. A single computer cannot render terabytes of data interactively. Parallel rendering, which utilizes a supercomputer or a cluster of PCs for rendering, provides a natural solution for effective visualization of very large data sets. As visualization calculations are shared among multiple processors in a divide-and-conquer manner, the I/O and rendering costs are split among processors. A parallel rendering solution that ensures a balanced workload across processors makes interactive visualization of large data sets not only possible, but also very practical.

A typical parallel rendering solution consists of several stages, including data partition, distribution, rendering, final image composition, and image delivery. Among the three basic parallel rendering approaches, namely, sort-first, sort-middle, and sort-last, defined by Molar et al. [9], sort-last parallel rendering has been widely used by visualization researchers [5, 1, 20, 19, 17, 16, 12, 2, 3] due to its simple task decomposition for achieving load balancing. With the sort-last approach, the rendering stage scales very well since no communication overhead is involved. It can be easily incorporated into existing visualization software or systems for high-performance parallel rendering. However, the final image compositing stage (i.e., back-to-front alpha blending of partial images) in sort-last rendering demands interprocessor communication, which could become very expensive because of the potentially large amount of messages exchanged. Therefore, image compositing could become a bottleneck that affects the efficiency of the sort-last parallel rendering pipeline.

Among different image compositing methods developed for sort-last parallel volume rendering, direct send [11] and binary swap [7, 8] are the two representative and most commonly used ones. Although simple and easy to implement, the direct send method requires $N \cdot (N-1)$ messages to be exchanged among N compositing processors in the worst case. The binary swap compositing method reduces the number of messages exchanged from $N \cdot (N-1)$ to $N \log N$ using a binary tree style compositing process. However, to fully utilize the parallelism, the number of processors used must be an exact power-of-two due to the nature of binary compositing.

In this paper, we introduce a new image compositing algorithm, called *2-3 swap*, for parallel volume rendering which uses an arbitrary number of processors. The advantage of the 2-3 swap compositing algorithm is that it offers benefits of both the direct send and binary swap algorithms. On one hand, the 2-3 swap compositing algorithm can utilize any number of processors available, thus is as flexible as the direct send method. On the other hand, the number of messages exchanged among processors is bounded by $O(N \log N)$, which is as good as the binary swap algorithm. Actually, as

*Department of Computer Science, University of California at Davis, One Shields Avenue, Davis, CA, 95616. {hfyu,chawang,klma}@ucdavis.edu

shown later in this paper, the binary swap algorithm can be considered as a special case of our 2-3 swap algorithm. Like the direct send and binary swap algorithms, the 2-3 swap algorithm is easy to understand and simple to implement. Another advantage of this algorithm is that it has the potential to be incorporated into simulation-time data visualization where the rendering is performed simultaneously with the simulation using the same supercomputer [18, 21]. Our algorithm scales very well in a massively parallel environment (i.e., thousands of processors on a supercomputer), which we demonstrate using parallel volume rendering of large data sets to generate high-resolution (up to 4096^2) images.

2. RELATED WORK

Parallel computing has been widely used in many areas of graphics and visualization, such as polygon rendering, isosurface extraction, particle tracing, and volume rendering. Since the early 1990s, several image compositing methods have been developed for parallel visualization tasks that require combining partial images generated from processors into a final image. The direct send method is the simplest compositing technique [4, 10, 11]. This method is flexible with network interconnect and the number of processors used in compositing. However, it introduces link contention due to the nature of all-to-all communication pattern. Ma et al. [7, 8] introduced the binary swap method which makes use of all processors in all stages of composition in a binary tree manner. The method perfectly balances the compositing workload among processors and reduces the number of messages exchanged. Finally, Lee et al. [5] introduced a parallel pipeline method which avoids link contention. This contention-free solution restricts each processor to sending only one message to its next processor in turn. The compositing takes $N-1$ stages to complete for N compositing processors, and the total number of messages exchanged remains the same as the direct send method.

Over the years, various techniques have been presented to optimize the aforementioned parallel image compositing algorithms. One optimization technique is to take advantage of the sparseness of the partial images (i.e., images with a large area of background pixels) on each processor to reduce the workload [7, 8]. To optimize direct send, Stompel et al. [15] introduced scheduled linear image compositing (SLIC) algorithm which computes a compositing schedule and classifies pixels into background pixels, pixels in the non-overlapping areas, and pixels in the overlapping areas. Only the pixels in the overlapping areas require compositing. Lee et al. [5] used multiple bounding boxes to skip the background pixels in their parallel pipeline method. The same solution was adopted by Takeuchi et al. [17] in a binary swap implementation. Strengert et al. [16] presented a compositing scheme that takes into account the footprints of volume bricks in the data hierarchy to minimize the costs for reading from framebuffer, network communication, and blending. The other optimization technique is to take a finer-grain image space partition for the compositing tasks [5, 20, 15, 17]. This technique achieves a more balanced workload by assigning finer-grain image partitions among processors in an interleaving fashion. The third optimization technique is to compress the pixel data before transmitting them to other processors. Several works [7, 8, 1, 20, 17, 15] reported the use of run-length encoding for lossless compression to accelerate pixel transferring and reduce communication cost. Finally, the last optimization technique is to overlap all stages of the process through pipelining, including reading, transferring, and blending pixel data. For example, Cavin et al. [2] presented a pipelined sort-last implementation of binary swap that uses multi-threads to overlap pixel reading, transferring, and blending. All these performance improvements, however, remain $O(N^2)$ in terms of message exchanged for the direct send method and do not remove the constraint of power-of-two processors for the binary swap algorithm.

Hardware-based image compositing equipment and solutions such as Lighting-2 [14] and Sepia-2 [6] have also been introduced. Although these methods achieve impressive compositing speed and high scalability, the cost for building a large-scale visualization system using such hardware devices could be prohibitively expensive. More recently, Nonaka et al. [12] presented a hybrid image composition method for sort-last parallel rendering on graphics clusters with the MPC image compositor, which offers high-performance image compositing at a reasonable cost. Pugmire et al. [13] presented a new solution that utilizes a network processing unit (NPU) for hardware-based image compositing in a distributed rendering system. Nevertheless, software image compositing is still widely used in most applications to date. Moreover, in a supercomputing environment with thousands of processors, employing dedicated graphics hardware for image compositing is not practical due to the costs of the hardware and the added efforts for integration, upgrading, and maintenance. Our software rendering solution is designed to use a supercomputer, which could be the same machine that runs the simulation. Therefore, our solution can be employed in simulation-time data visualization where the simulation and visualization calculations share the same machine. Such a solution may output visualization results instead of simulation data, which substantially reduces the I/O cost and obviates any post-processing.

3. 2-3 SWAP IMAGE COMPOSITING

In this section, we first discuss the pros and cons for the direct send and binary swap image compositing methods. Two straightforward extensions of the binary swap algorithm are also covered for comparison. Then, we describe our image compositing strategy and present the 2-3 swap image compositing method.

3.1 Direct Send and Binary Swap

All parallel image compositing algorithms require partitioning the image space and assigning the corresponding compositing tasks to processors. The assignment should be performed in a way so that the compositing workload is evenly distributed among processors. Each processor is assigned some image partitions and is responsible for the compositing tasks in those partitions. The final image is constructed by tiling all image partitions in order.

The direct send method is the simplest image compositing technique: each processor sends pixel data directly to the processor responsible for compositing that image partition. It requires only a single stage of communication. But, in the worst case, there are $N \cdot (N-1)$ messages to be exchanged among N compositing processors. In the communication network, link contention is likely to happen for the direct send method since there are multiple processors sending messages to the same processor at the same time.

The binary swap method utilizes a binary tree which pairs up processors in order of compositing. It requires $\log N$ stages of communication for N compositing processors. Unlike a typical binary tree compositing algorithm where half of the participating processors become idle after each stage of the composition, the binary swap algorithm exploits more parallelism in compositing so that every processor participates in all stages of the compositing process. The key idea is that at each stage, instead of having only one processor from each pair composite for the whole image space, the image space is divided into two partitions and each processor takes the responsibility for one of the two partitions. A swapping of the partitions between the two processors is needed in the algorithm, thus the name *binary swap* [7, 8]. Note that the image partition needed for compositing becomes smaller as we move up to the root of the binary tree. After $\log N$ stages, the compositing completes and each processor holds exactly $\frac{1}{N}$ of the final image. Binary swap can also take advantage of faster nearest neighbor communications in a tree or hypercube network: in early stages of the algorithm, processors exchange messages with their nearest neighbors, which is exactly when the amount of pixels transferred is the largest. This nice property makes binary swap much more scalable than other methods.

The only drawback of binary swap algorithm is that it requires the number of processors to be a power-of-two ($N = 2^K$). If the number of processors is not exactly a power-of-two (i.e., $2^{K-1} < N < 2^K$), a straightforward solution is to first send the images from $N - 2^{K-1}$ processors to the rest of 2^{K-1} processors and then perform binary swap on the 2^{K-1} processors directly. This *reduced binary swap* solution is convenient. However, in the worst case where $N = 2^K - 1$, about half of the processors will be idle during the entire compositing process. Moreover, the total image data exchanged will be twice as much as in the case where $N = 2^K$, which is clearly not desirable. Another solution is to still use the binary compositing tree with $K+1$ levels. In this scenario, the binary tree is complete (with 2^K leaf nodes) but not full. Additional complexity is required in the compositing partner computation. Nevertheless, in the worst case where $N = 2^{K-1} + 1$, the $(2^{K-1}+1)$th processor will be idle in all but the last compositing stages. In the last stage, the $(2^{K-1}+1)$th processor will communicate with all the other 2^{K-1} processors and be responsible for compositing half of the final image, introducing the most imbalanced workload. Therefore, both solutions described here are not optimal in terms of parallelism utilization and compositing efficiency.

3.2 Image Compositing Strategy

By pairing up processors in order of compositing, the binary swap method essentially uses the direct send method in each pair. As such, each processor only communicates with another processor at any stage of compositing. Given an arbitrary number of processors, our goal is to utilize all processors in all stages of the compositing process. Thus, no processors will be idle at any stage, and maximum parallelism is ensured. Similar to the binary swap algorithm, which partitions processors into pairs, we could partition the processors into groups with each group using the direct send method for compositing. At any stage of the compositing, a processor only communicates with other processors in its own group. The number of processors in each group should thus be as small as possible in the initial partition (which corresponds to the leaf nodes in the compositing tree) so that a processor only exchanges messages with a few other processors in its group. Care should be taken when considering how to partition the processors. Group size variation should be kept to a minimum, making the overall compositing procedure simple and possible to use an arbitrary number of processors. Furthermore, we must pay special attention to the choices of image data partition and assignment schemes in order to reduce the number of messages among processors, which is essential to ensure the scalability of our algorithm.

Algorithm 1 ConstructCompositingTree ($node, n, d$)

```
1: if d = 0 then
2:    node_list ⇐ procid {In node, node_list keeps the order of the
      indices of processors for image partition assignment; procid
      is a global variable, initialized as 1.}
3:    procid ⇐ procid + 1
4: else
5:    l ⇐ ⌊n/2⌋
6:    r ⇐ ⌈n/2⌉
7:    if r < 2^d then
8:       create two children, namely, node_l and node_r, for node
9:       ConstructCompositingTree (node_l, l, d − 1)
10:      ConstructCompositingTree (node_r, r, d − 1)
11:   else
12:      l ⇐ ⌊n/3⌋
13:      m ⇐ ⌊n/3⌋
14:      r ⇐ ⌈n/3⌉
15:      create three children, namely, node_l, node_m, and node_r,
         for node
16:      ConstructCompositingTree (node_l, l, d − 1)
17:      ConstructCompositingTree (node_m, m, d − 1)
18:      ConstructCompositingTree (node_r, r, d − 1)
19:   end if
20: end if
```

3.3 The Algorithm

The 2-3 swap image compositing algorithm is a generalization of binary swap to an arbitrary number of processors. The algorithm is based on an important observation that any integer number N ($N > 1$) can be decomposed into a summation of a list of twos and threes. Therefore, the initial partition of processors can be achieved using a combination of twos and threes. 2-3 swap follows a similar multi-stage image composition process as the binary swap algorithm. Actually, if the number of processors is a power-of-two, then our solution reduces to binary swap.

Given N processors, $2^{K-1} \leq N < 2^K$, we construct the compositing tree recursively using Algorithm 1 by calling ConstructCompositingTree ($root, N, K-1$). The tree constructed has K levels. Each non-leaf node in the compositing tree has either two or three children. Note that the structure of the compositing tree determines the groups of processors during each stage of image compositing.

We justify the correctness of Algorithm 1 by showing that the condition $2^d \leq n < 2^{d+1}$ holds in every iteration of the recursion. For the two children case, $(n \mod 2)$ is either 0 or 1, so we have $l + r = n$ and $2^{d-1} \leq l \leq r < 2^d$. Therefore, the condition holds. For the three children case, $(n \mod 3)$ can be 0, 1 or 2. When $(n \mod 3)$ is either 0 or 1, we have $l + m + r = n$ and $2^{d-1} \leq l = m \leq r < 2^d$, thus the condition holds. When $(n \mod 3) = 2$, we have $l + m + r = n - 1$, and then the condition does not hold. However, we can show that the case $(n \mod 3) = 2$ is impossible. For the three children case, the *if* statement returns false in Line 7, implying that

Algorithm 2 OrderAssignment (*node*)

```
1: if NumberOfChildren (node) = 0 then
2:    return
3: else
4:    if NumberOfChildren (node) = 2 then
5:       OrderAssignment (node_l)
6:       OrderAssignment (node_r)
7:       if SizeOfList (node_r) > SizeOfList (node_l) then
8:          node_list ⇐ merge the lists of node_r and node_l interleavingly
9:       else
10:         node_list ⇐ merge the lists of node_l and node_r interleavingly
11:      end if
12:   else
13:      OrderAssignment (node_l)
14:      OrderAssignment (node_m)
15:      OrderAssignment (node_r)
16:      if SizeOfList (node_r) > SizeOfList (node_l) then
17:         node_list ⇐ merge the lists of node_r, node_l, and node_m interleavingly
18:      else
19:         node_list ⇐ merge the lists of node_l, node_m, and node_r interleavingly
20:      end if
21:   end if
22: end if
```

$r = 2^d$ and $n = 2^{d+1} - 1$. Since 2 and 3 are relatively prime, 3 does not divide evenly into 2^{d+1}. Therefore, $(2^{d+1} \mod 3)$ is either 1 or 2, and $((2^{d+1} - 1) \mod 3)$ is either 0 or 1. That is, the case $(n \mod 3) = 2$ is impossible.

The question remaining is how to partition the image space and schedule the workload for each group of processors. To ensure a balanced workload, an image is represented as a 1D pixel array in a scanline order, which is partitioned evenly among M participating processors in a group. For example, at the leaf-node level of the compositing tree, $M = 2$ or 3 in a group. Thus, in the first stage, we divide the image space evenly into two or three partitions and assign image partitions to processors in the group. The order of assignment follows the order of the index of the processors.

In the following image compositing stages, the number of processors in a group increases as neighboring groups are merged together. At each intermediate stage of compositing, two or three groups are merged into a new group and we maintain a total order of all their processors for the new group. The total order is formed by interleaving the partial orders of the groups being merged. The order of the groups follows these two rules in order. First, a group having a larger number of processors (i.e., the processors in the group having smaller partitions assigned) in the previous stage gets its order first in the current stage. Second, if the groups have the same number of processors, the group with processors of smaller indices gets its order first. This process is sketched in Algorithm 2, where NumberOfChildren(*node*) returns the number of children of *node* and SizeOfList(*node*) returns the size of processor list of *node*. Calling OrderAssignment (*root*) generates the orders at all tree nodes.

For intuitive understanding, we illustrate our 2-3 swap image compositing algorithm with five, seven, and nine processors in Figure 1, 2, and 3, respectively. For simplicity, we use 2D region to represent 1D array partition of the image. Note that as the compositing progresses through different stages, the number of processors that each processor in a group communicates with remains fairly small, which we analyze next.

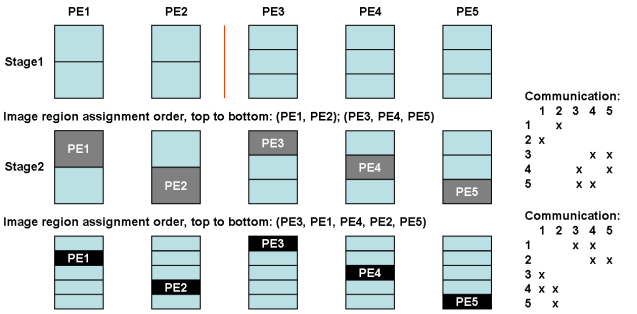


Figure 1: 2-3 swap image compositing with five processors. Image partition assignment order indicates which processor is responsible for which partition in a group, denoted by the parentheses. There are two compositing stages. From the communication matrix given on the right, we can see that at any stage, each processor communicates with up to two other processors.

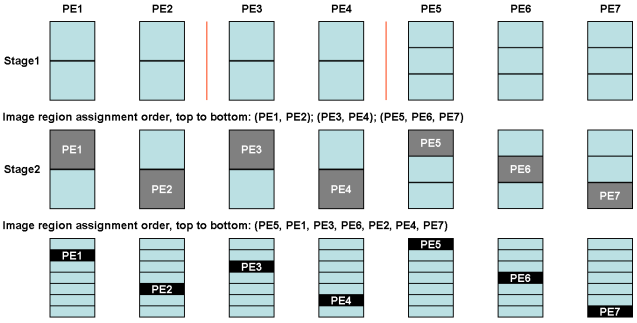


Figure 2: 2-3 swap image compositing with seven processors. There are two compositing stages. At any stage, each processor communicates with up to four other processors.

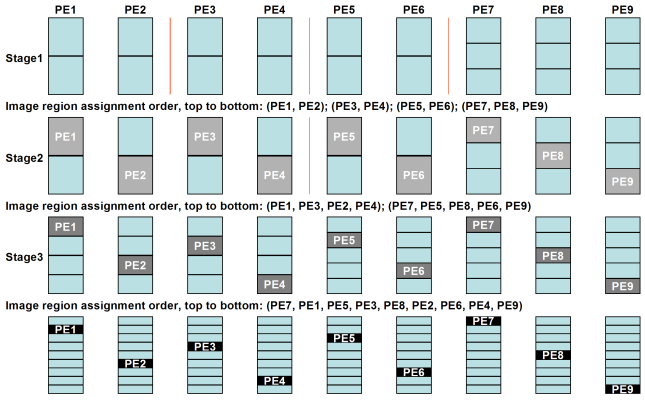


Figure 3: 2-3 swap image compositing with nine processors. There are three compositing stages. At any stage, each processor communicates with up to two other processors.

In a compositing tree with N processors, we know from Algorithm 1 that a non-leaf node either has two or three children. For the two children case, the number of processors assigned to child nodes could be either (M, M) or (M,

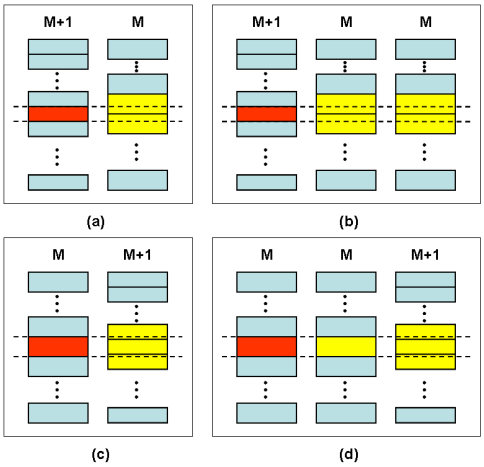


Figure 4: All possible cases of the image space partition that is not perfectly aligned for the compositing tree nodes. The figure shows the worst case scenarios with the maximum number of partition overlaps. A processor (denoted in red) communicates with other processors (denoted in yellow) in a group. Our further analysis shows that (c) and (d) are impossible cases. Therefore, a processor communicates up to four other processors in a group.

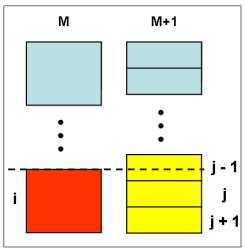


Figure 5: An impossible case of one partition with M processors overlapping with three partitions with $M+1$ processors.

$M+1$). For the three children case, the number of processors assigned to child nodes could be either (M, M, M) or (M, M, $M+1$)[1]. If the number of processors is evenly split in the child nodes, i.e., (M, M) or (M, M, M), then the image space partition is perfectly aligned for all nodes in a group and each processor communicates with one (for the two children case) or two (for the three children case) processors. However, the image space partition is not perfectly aligned for all nodes in a group if the number of processors is not evenly split in the child nodes, i.e., (M, $M+1$) or (M, M, $M+1$). These cases with the maximum number of partition overlaps are illustrated in Figure 4. In the figure, the processor denoted in red communicates with two, four, three, and four other processors denoted in yellow in (a), (b), (c), and (d) respectively.

A further analysis shows that Figure 4 (c) and (d) are actually impossible cases where one partition with M processors overlaps with three other processors in a partition with $M+1$ processors. Let us assume that such a case is possible, i.e., the ith partition (denoted in red) with M processors overlapping three partitions $j-1$, j, and $j+1$ (denoted in yellow) with $M+1$ processors. If $i=M$, then $j=M$. That is, the last partition with M processors overlaps the last three partitions, as well as covering the last two partitions, with $M+1$ processors, as shown in Figure 5. In this case, we have $\frac{1}{M} > \frac{2}{M+1}$, which leads to $1 > M$. Thus, this case is impossible. If $i < M$, then $j = i+1$. In this case, we have $\frac{i}{M} > \frac{i+1}{M+1} + \delta$, where δ is a portion of the $(j+1)$th partition with $M+1$ processors. This leads to $i - M > \delta(M+1)M$, which is impossible since $i < M$ and $\delta > 0$. Therefore, the number of processors that each processor in a group communicates with is bounded by four. This implies that the performance of our 2-3 swap image compositing algorithm would far surpasses that of the direct send method, which in worst case requires a processor communicating with all $N-1$ processors for N compositing processors.

4. COMPLEXITY ANALYSIS AND COMPARISON

In this section, we first formulate the running time of the direct send, binary swap, and 2-3 swap image compositing methods. Then we provide the complexity analysis and comparison of these three methods.

4.1 Running Time Formulation

The image compositing methods take S stages to complete using N compositing processors, where $S = 1$ for the direct send method, $S = \log_2 N$ for the binary swap method, and $S = \lfloor \log_2 N \rfloor$ for the 2-3 swap method. The total time for compositing is:

$$Tcomp = \sum_{i=1}^{S} Tcomp_i \qquad (1)$$

where $Tcomp$ is the total running time needed for a processor to complete its compositing, and $Tcomp_i$ is the time the processor spends at stage i.

At each stage, a processor first exchanges pixels with other processors, which takes $Texchange_i$ time. After pixel exchange, a processor then blends the pixels (which it is responsible for) in $Tblend_i$ time. Thus, the compositing time is:

$$Tcomp_i = Texchange_i + Tblend_i \qquad (2)$$

On the other hand, the total compositing time of a processor can be expressed as the summation of the total exchanging time $Texchange$ and the total blending time $Tblend$, i.e.,

$$\begin{aligned} Tcomp &= Texchange + Tblend \\ &= \sum_{i=1}^{S} Texchange_i + \sum_{i=1}^{S} Tblend_i \end{aligned} \qquad (3)$$

where $Texchange_i$ consists of the time for sending ($Tsend_i$) and the time for receiving ($Trecv_i$). Let the latencies for sending and receiving be $Lsend_i$ and $Lrecv_i$, respectively. We assume that a single send and a single receive take the same amount of latency, which is denoted as $l_{sendrecv}$. For the current generation of supercomputer, the network usually supports full-duplex send and receive operations. So, sending and receiving can be overlapped. Thus, we have:

$$Texchange_i = \max(Lsend_i + Tsend_i,\ Lrecv_i + Trecv_i) \qquad (4)$$

[1]The (M, $M+1$, $M+1$) case is impossible, because we have $l = m \le r$ for the three children case. Refer to Algorithm 1.

and

$$Lsend_i = Nsend_i \times l_{sendrecv}, \quad Lrecv_i = Nrecv_i \times l_{sendrecv} \tag{5}$$

$$Tsend_i = \frac{Psend_i}{Bsend_i}, \quad Trecv_i = \frac{Precv_i}{Brecv_i} \tag{6}$$

where $Nsend_i$ and $Nrecv_i$ are the numbers of send and receive operations issued at stage i, respectively. $Psend_i$ and $Precv_i$ are the size of pixel data that a processor sends and receives, respectively. Each pixel contains four channels RGBA. In our experiment, we use 32-bit floating point format for each channel to ensure high precision compositing results. $Bsend_i$ and $Brecv_i$ are the network bandwidth for sending and receiving data, respectively. Without considering link contention, we also assume these two terms are the same, which is denoted as $Bsendrecv$.

The time for blending is:

$$Tblend_i = \frac{Pblend_i}{Bblend_i} \tag{7}$$

where $Pblend_i$ is the size of pixel data that a processor is responsible for blending. $Bblend_i$ is the amount of pixel data a processor can blend per second, which is assumed as a constant and denoted as $Bblend$.

Therefore, the overall compositing time can be written as:

$$\begin{aligned}
Tcomp &= Texchange + Tblend = \sum_{i=1}^{S} Texchange_i + \sum_{i=1}^{S} Tblend_i \\
&= \sum_{i=1}^{S} \max(Lsend_i + \frac{Psend_i}{Bsendrecv},\ Lrecv_i + \frac{Precv_i}{Bsendrecv}) \\
&\quad + \sum_{i=1}^{S} \frac{Pblend_i}{Bblend} \\
&\leq \sum_{i=1}^{S} \max(Lsend_i,\ Lrecv_i) \\
&\quad + \sum_{i=1}^{S} \max(\frac{Psend_i}{Bsendrecv},\ \frac{Precv_i}{Bsendrecv}) \\
&\quad + \frac{1}{Bblend} \sum_{i=1}^{S} Pblend_i \\
&= l_{sendrecv} \times \sum_{i=1}^{S} \max(Nsend_i,\ Nrecv_i) \\
&\quad + \frac{1}{Bsendrecv} \times \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) \\
&\quad + \frac{1}{Bblend} \times \sum_{i=1}^{S} Pblend_i
\end{aligned} \tag{8}$$

We further denote:

$$\begin{aligned}
Nsendrecv &= \sum_{i=1}^{S} \max(Nsend_i,\ Nrecv_i) \\
Psendrecv &= \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) \\
Pblend &= \sum_{i=1}^{S} Pblend_i
\end{aligned} \tag{9}$$

From Equations (8) and (9), we have:

$$\begin{aligned}
Tcomp \leq\ & l_{sendrecv} \times Nsendrecv \\
& + \frac{1}{Bsendrecv} \times Psendrecv \\
& + \frac{1}{Bblend} \times Pblend
\end{aligned} \tag{10}$$

In the following, we analyze $Nsendrecv$, $Psendrecv$, and $Pblend$ for the direct send, binary swap, and 2-3 swap algorithms, respectively. We assume that the total number of pixels in the final image is P, and the number of compositing processors is N.

4.2 Direct Send

For the direct send algorithm, S is equal to 1, since it requires only one stage of all-to-all communication. Each processor sends $\frac{P}{N} \times (N-1)$ pixels to and receive $\frac{P}{N} \times (N-1)$ pixels from all the other $N-1$ processors. So we have:

$$Nsendrecv = \sum_{i=1}^{S} \max(Nsend_i,\ Nrecv_i) = N-1 \tag{11}$$

and

$$\begin{aligned}
Psendrecv &= \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) \\
&= \max(\frac{P}{N} \times (N-1),\ \frac{P}{N} \times (N-1)) \\
&= P \times (1 - \frac{1}{N})
\end{aligned} \tag{12}$$

On the other hand, each of the N processors is responsible for blending $\frac{P}{N}$ pixels of the final image. So the total number of pixels a processor needs to blend is:

$$Pblend = \sum_{i=1}^{S} Pblend_i = \frac{P}{N} + \frac{P}{N} \times (N-1) = P \tag{13}$$

4.3 Binary Swap

The binary swap algorithm takes $S = \log_2 N$ stages to complete. At the ith stage, each processor only needs to exchange pixels with another processor in its pair. The amount of pixel data for a processor to send and receive are the same, which is $\frac{P}{2^i}$. Thus, we have:

$$\begin{aligned}
Nsendrecv &= \sum_{i=1}^{S} \max(Nsend_i,\ Nrecv_i) = \sum_{i=1}^{\log_2 N} 1 \\
&= \log_2 N
\end{aligned} \tag{14}$$

and

$$\begin{aligned}
Psendrecv &= \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) = \sum_{i=1}^{\log_2 N} \frac{P}{2^i} \\
&= P \times (1 - \frac{1}{N})
\end{aligned} \tag{15}$$

At the ith stage, each processor in a pair is responsible for blending $\frac{P}{2^i}$ pixels of the final image. So the total number of pixels a processor needs to blend is:

$$\begin{aligned}
Pblend &= \sum_{i=1}^{S} Pblend_i = \sum_{i=1}^{\log_2 N} (\frac{P}{2^i} + \frac{P}{2^i}) \\
&= 2 \times P \times (1 - \frac{1}{N})
\end{aligned} \tag{16}$$

4.4 2-3 Swap

The 2-3 swap algorithm takes $S = \lfloor \log_2 N \rfloor$ stages to complete. Unlike the direct send and binary swap algorithms, the workload of all processors at each stage of the 2-3 swap could be imbalanced. For simplicity, we consider the maximum workload assigned to the processors at each stage as their workload. This corresponds to analysis of the worst case scenario.

From Section 3.3, we know that the number of processors that each processor in a group communicates with is bounded by four. Therefore,

$$\begin{aligned} Nsendrecv &= \sum_{i=1}^{S} \max(Nsend_i,\ Nrecv_i) = \sum_{i=1}^{\lfloor \log_2 N \rfloor} 4 \\ &= 4 \times \lfloor \log_2 N \rfloor \end{aligned} \tag{17}$$

We assume that at the $(i-1)$th stage, the image fraction assigned to a processor is $\frac{1}{M_{i-1}}$, where M_{i-1} is the number of processors in the group at the $(i-1)$th stage. At the ith stage, the image fraction assigned to the processor is $\frac{1}{M_i}$, where M_i is the number of the participating processors in the group at the ith stage.

Therefore, at the ith stage, the numbers of pixels each processor in the group needs to send and receive are:

$$Psend_i = (\frac{1}{M_{i-1}} - \frac{1}{M_i}) \times P \tag{18}$$

and

$$Precv_i = c \times \frac{1}{M_i} \times P \tag{19}$$

where $c = 1$ or 2, since each non-leaf node in the compositing tree only has two or three children.

If $c = 1$, M_{i-1} can be either $\lfloor \frac{M_i}{2} \rfloor$ or $\lceil \frac{M_i}{2} \rceil$. We have:

$$\frac{1}{\lceil \frac{M_i}{2} \rceil} - \frac{1}{M_i} \leq \frac{1}{M_i} \leq \frac{1}{\lfloor \frac{M_i}{2} \rfloor} - \frac{1}{M_i} \tag{20}$$

Thus, the sending dominates pixel exchange when $c = 1$. In the worst case, $M_i = 2^i + 1$ and $M_{i-1} = 2^{i-1}$, then

$$\begin{aligned} Psendrecv &= \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) = \sum_{i=1}^{S} Psend_i \\ &= P \times \sum_{i=1}^{S} (\frac{1}{M_{i-1}} - \frac{1}{M_i}) \\ &= P \times \sum_{i=1}^{S} (\frac{1}{2^{i-1}} - \frac{1}{2^i + 1}) \\ &\leq P \times (1 + \sum_{i=1}^{S} \frac{1}{2^i \times (2^i + 1)}) \\ &\leq P \times (1 + \sum_{i=1}^{S} \frac{1}{4^i}) < P \times (1 + \sum_{i=1}^{\infty} \frac{1}{4^i}) \\ &= \frac{4}{3} \times P \end{aligned} \tag{21}$$

If $c = 2$, M_{i-1} can be either $\lfloor \frac{M_i}{3} \rfloor$ or $\lceil \frac{M_i}{3} \rceil$. In this case, we also have:

$$\frac{1}{\lceil \frac{M_i}{3} \rceil} - \frac{1}{M_i} \leq \frac{2}{M_i} \leq \frac{1}{\lfloor \frac{M_i}{3} \rfloor} - \frac{1}{M_i} \tag{22}$$

That is, the sending also dominates pixel exchange when $c = 2$. In the worst case, $M_i = 2^{i+1} - 1$ and $M_{i-1} = \lfloor \frac{2^{i+1}-1}{3} \rfloor$, then

$$\begin{aligned} Psendrecv &= \sum_{i=1}^{S} \max(Psend_i,\ Precv_i) = \sum_{i=1}^{S} Psend_i \\ &= P \times \sum_{i=1}^{S} (\frac{1}{M_{i-1}} - \frac{1}{M_i}) \\ &= P \times \sum_{i=1}^{S} (\frac{1}{\lfloor \frac{2^{i+1}-1}{3} \rfloor} - \frac{1}{2^{i+1} - 1}) \\ &\leq P \times (1 + \sum_{i=2}^{S+1} (\frac{1}{2^i - 1})) \leq P \times (1 + \frac{1}{6} \times \sum_{i=0}^{S-1} \frac{1}{2^i}) \\ &< P \times (1 + \frac{1}{6} \times \sum_{i=0}^{\infty} \frac{1}{2^i}) = \frac{4}{3} \times P \end{aligned} \tag{23}$$

From Equations (21) and (23), we can see that *Psendrecv* is bounded by $\frac{4}{3} \times P$.

For the blending workload, we have:

$$Pblend_i = c \times \frac{1}{M_i} \times P \tag{24}$$

where $c = 2$ or 3, since each non-leaf node in the compositing tree only has two or three children. If $c = 2$, in the worst case, $Pblend_i$ becomes the largest when $M_i = 2^i$. Thus, we have:

$$\begin{aligned} Pblend &= \sum_{i=1}^{S} Pblend_i = \sum_{i=1}^{S} (c \times \frac{1}{M_i} \times P) \\ &= \sum_{i=1}^{\lfloor \log_2 N \rfloor} (2 \times \frac{1}{2^i} \times P) \leq 2 \times P \times (1 - \frac{1}{N}) \end{aligned} \tag{25}$$

If $c = 3$, then $M_i = 2^{i+1} - 1$. We have:

$$\begin{aligned} Pblend &= \sum_{i=1}^{S} Pblend_i = \sum_{i=1}^{S} (c \times \frac{1}{M_i} \times P) \\ &= \sum_{i=1}^{S} (3 \times \frac{1}{2^{i+1} - 1} \times P) = 3 \times P \times \sum_{i=2}^{S+1} \frac{1}{2^i - 1} \\ &\leq 3 \times P \times (\frac{1}{3} \times \sum_{i=0}^{S-1} \frac{1}{2^i}) < 3 \times P \times (\frac{1}{3} \times \sum_{i=0}^{\infty} \frac{1}{2^i}) \\ &= 3 \times P \times (\frac{1}{3} \times 2) = 2 \times P \end{aligned} \tag{26}$$

From Equations (25) and (26), we can see that *Pblend* is bounded by $2 \times P$.

4.5 Summary

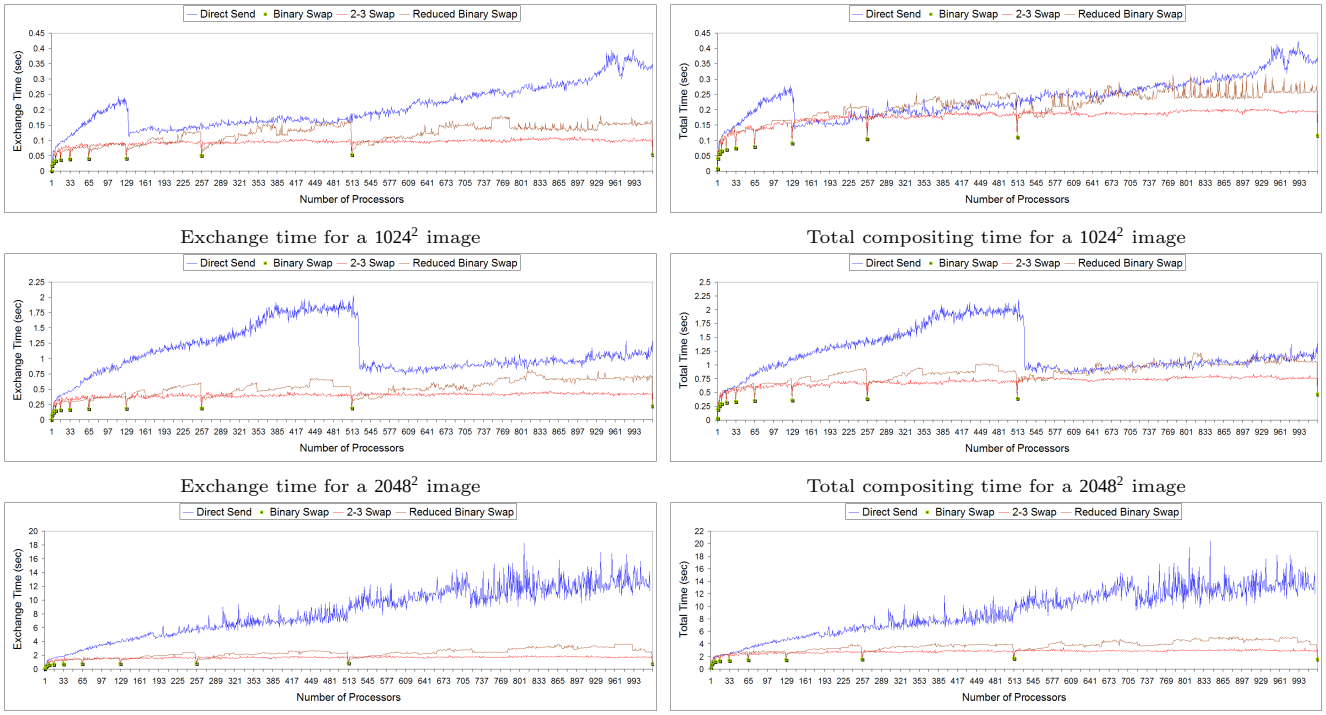
Table 1 summarizes the complexity comparison of the direct send, binary swap, reduced binary swap (Section 3.1), and 2-3 swap algorithms. Note that for simplicity, the complexity of the 2-3 swap algorithm derived in this section gives a loose upper bound. For example, in latency evaluation, chances are very slim for a processor to communicate with four other processors in a group. The actual performance based on the average case shows a better (tighter) bound, which we present in the following section.

5. RESULTS AND DISCUSSION

We experimented with our 2-3 swap algorithm on the Cray XT4 machine at the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science facility at Lawrence Berkeley National Laboratory. The NERSC

	Latency	Send & Recv	Blend
Direct Send	$l_{sendrecv} \times (N-1)$	$\frac{1}{B_{sendrecv}} \times P \times (1-\frac{1}{N})$	$\frac{1}{B_{blend}} \times P$
Binary Swap	$l_{sendrecv} \times \log_2 N$	$\frac{1}{B_{sendrecv}} \times P \times (1-\frac{1}{N})$	$\frac{1}{B_{blend}} \times 2 \times P \times (1-\frac{1}{N})$
Reduced Binary Swap	$l_{sendrecv} \times (\lfloor \log_2 N \rfloor + 1)$	$\frac{1}{B_{sendrecv}} \times P \times (2-\frac{1}{2^{\lfloor \log_2 N \rfloor}})$	$\frac{1}{B_{blend}} \times 2 \times P \times (2-\frac{1}{2^{\lfloor \log_2 N \rfloor}})$
2-3 Swap	$l_{sendrecv} \times 4 \times \lfloor \log_2 N \rfloor$	$\frac{1}{B_{sendrecv}} \times \frac{4}{3} \times P$	$\frac{1}{B_{blend}} \times 2 \times P$

Table 1: The summary of complexity of the direct send, binary swap, reduced binary swap, and 2-3 swap algorithms. The complexity for the reduced binary swap algorithm can be straightforwardly derived from the formulation of the binary swap algorithm (Section 4.3). Note that the complexity of the 2-3 swap algorithm shown here is the loosely-estimated worst case scenario. The actual performance results from our experiments show a tighter bound on the average.



Exchange time for a 1024^2 image

Total compositing time for a 1024^2 image

Exchange time for a 2048^2 image

Total compositing time for a 2048^2 image

Exchange time for a 4096^2 image

Total compositing time for a 4096^2 image

Figure 6: The timing results of the direct send, binary swap, 2-3 swap and reduced binary swap algorithms on any number of processors from 1 to 1024 with the output image resolutions of 1024^2 (top), 2048^2 (middle) and 4096^2 (bottom). The left column shows the exchange time, and the right column shows the total image compositing time. For the direct send method, the sudden drop of timing on 132 processors (top) and 525 processors (middle) is due to the message size setting. The timing for binary swap algorithm is the same at the 2-3 swap and reduced binary swap algorithms when the number of compositing processors is a power-of-two.

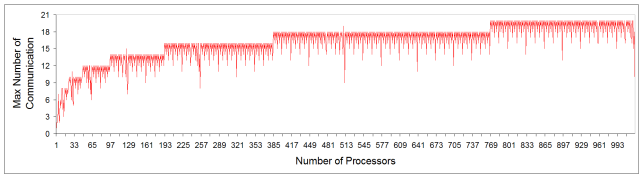
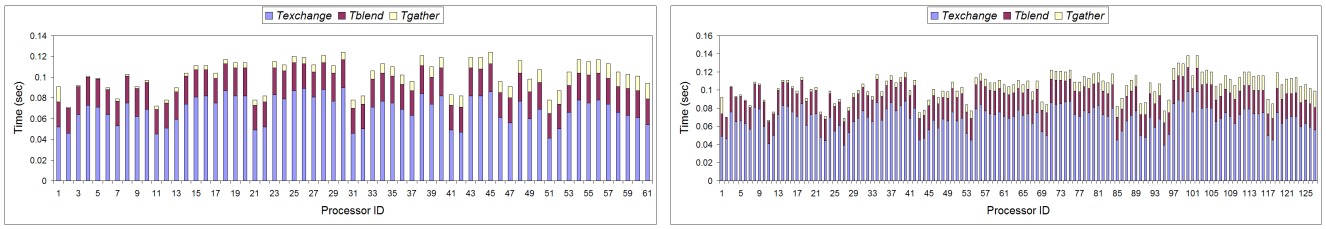


Figure 7: The maximum number of communication for the 2-3 swap algorithm with an output image resolution of 1024^2. The figure shows the results on any number of processors from 1 to 1024. The maximum number of communication is calculated as the summation of the maximum number of communication for all nodes at each level of the compositing tree.

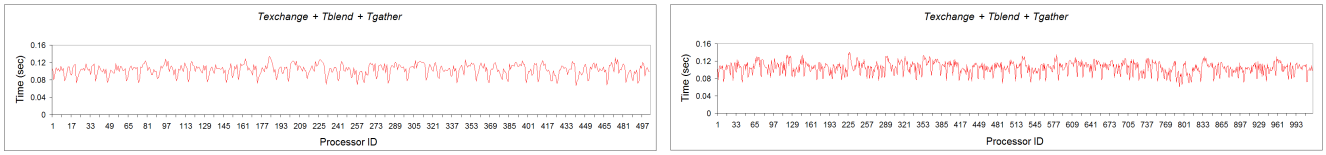
Cray XT4 system, named *Franklin*, is a massively parallel processing (MPP) system with 9660 compute nodes. Each node has dual cores (2.6GHz dual-core AMD Opteron processor) and 4GB of memory, and is connected to a dedicated SeaStar2 router through Hypertransport with a 3D torus topology. Such a topology is adopted to ensure high-performance, low-latency communication for MPI jobs.

We tested any number of processors from 1 to 1024 with three output image resolutions of 1024^2, 2048^2, and 4096^2. We compare the algorithms in the worst case, where all pixel data are considered for compositing. We did not implement particular optimization techniques. Nevertheless, removing blank pixels or not will not affect the comparison between the 2-3 swap algorithm and the other algorithms. The only change is that the number of total effective pixels becomes smaller. Such a number is the same regardless which image



(a) (b)

Figure 8: The figure shows the timing breakdown of 2-3 swap for each of the processors. The output image resolution is 1024^2. (a): 61 compositing processors. (b): 127 compositing processors.



(a) (b)

Figure 9: The figure shows the total compositing time of 2-3 swap for each of the processors. The output image resolution is 1024^2. (a): 503 compositing processors. (b): 1021 compositing processors.

compositing and/or compression methods are used.

Figure 6 shows the pixel data exchange time and the total image compositing time respectively for the direct send, binary swap, 2-3 swap, and reduced binary swap algorithms. As described in Section 3.1, the reduced binary swap algorithm uses only 2^{K-1} processors for image compositing when the given number of processors N is not a power-of-two, i.e., $2^{K-1} < N < 2^K$. For image resolution of 4096^2, we were not able to get results beyond 1021 processors for the direct send method (we changed the environment variables such as `MPICH_PTL_UNEXPECTED_EVENTS` and `MPICH_PTL_OTHER_EVENTS`, but still could not get a setting that works). It appears that for the machine we tested, the large size of data exchanged and the large number of all-to-all communication required are over the capability of MPI to handle.

From Figure 6, we can see that when the number of processors is a power-of-two, the timing for binary swap algorithm is the same as the 2-3 swap and reduced binary swap algorithms. The 2-3 swap algorithm maintains almost a constant performance and scales much better than the direct send method with the increase of processor number and/or the increase of output image resolution. Moreover, in terms of timing stability and efficiency, the 2-3 swap algorithm also outperforms the reduced binary swap algorithm. The reduced binary swap algorithm requires extra image data exchanged in the reduction stage (i.e., sending the images from $N - 2^{K-1}$ processors to the rest of 2^{K-1}, where $2^{K-1} < N < 2^K$) and nearly doubles the image blending workload for a large number of processors.

Note that for the direct send method, the sudden drops of timing in Figure 6 (top and middle) are due to the setting of `MPICH_MAX_SHORT_MSG_SIZE` in the Cray XT4 machine, which indicates the maximum size of a message in bytes that can be sent via the short (eager) protocol. The default value for the machine we used is 128000 bytes. With the 1024^2 (2048^2) output image resolution, i.e., RGBA and 32-bit for each channel, the machine started to use short messages when the number of compositing processors changes from 131 to 132 (from 524 to 525). On the other hand, as the special case for the 2-3 swap algorithm when the number of compositing processors is a power-of-two, the performance of the binary swap algorithm constitutes the lower bound for the 2-3 swap algorithm. Nevertheless, when the number of processors is not an exact power-of-two, the timing of the 2-3 swap algorithm is at most two times as those of the binary swap algorithm.

Figure 7 shows the maximum number of communication for the 2-3 swap algorithm with an output image resolution of 1024^2. We tested any number of processors from 1 to 1024. The maximum number of communication is calculated as the summation of the maximum number of communication for all nodes at each level of the compositing tree. The statistics show that on average, the number of communication is $1.98 \times \lfloor \log_2 N \rfloor$, which is much better than the upper bound, $4 \times \lfloor \log_2 N \rfloor$, that we estimate in Table 1. Among all possible maximum numbers (one to four) of communication for a processor at any stage of compositing, about 80% are the cases of one and two. On the other hand, the statistics from pixel data sending/receiving, and blending run on any number of processors from 1 to 1024 show that the 2-3 swap algorithm introduces a maximum $1.23 \times P$ for pixel sending/receiving, where P is the size of the final image, and a maximum $2.0 \times P$ for pixel blending. On average, it introduces $1.19 \times P$ for pixel sending/receiving and $1.80 \times P$ for pixel blending. The actual results show a tighter bound of 1.23 than $\frac{4}{3}$, which we estimate in Section 4.4.

We also provide the detail timing breakdown for the 2-3 swap algorithm: Figure 8 shows the timing breakdown for each of the processors when 61 and 127 processors are used in compositing, respectively. As formulated in Section 4.1, the compositing time includes the data exchange time and blending time. In the gathering stage, each processor sends its composited partial image to a host processor for tiling them to get the final image. In terms of the total compositing time, the difference ratio, defined as $(\max T - \min T)/\max T$, where $T = Texchange + Tblend + Tgather$, is 43% for Figure 8 (a) and 52% for Figure 8 (b). In Figure 9, we show the total compositing time for each of the proces-

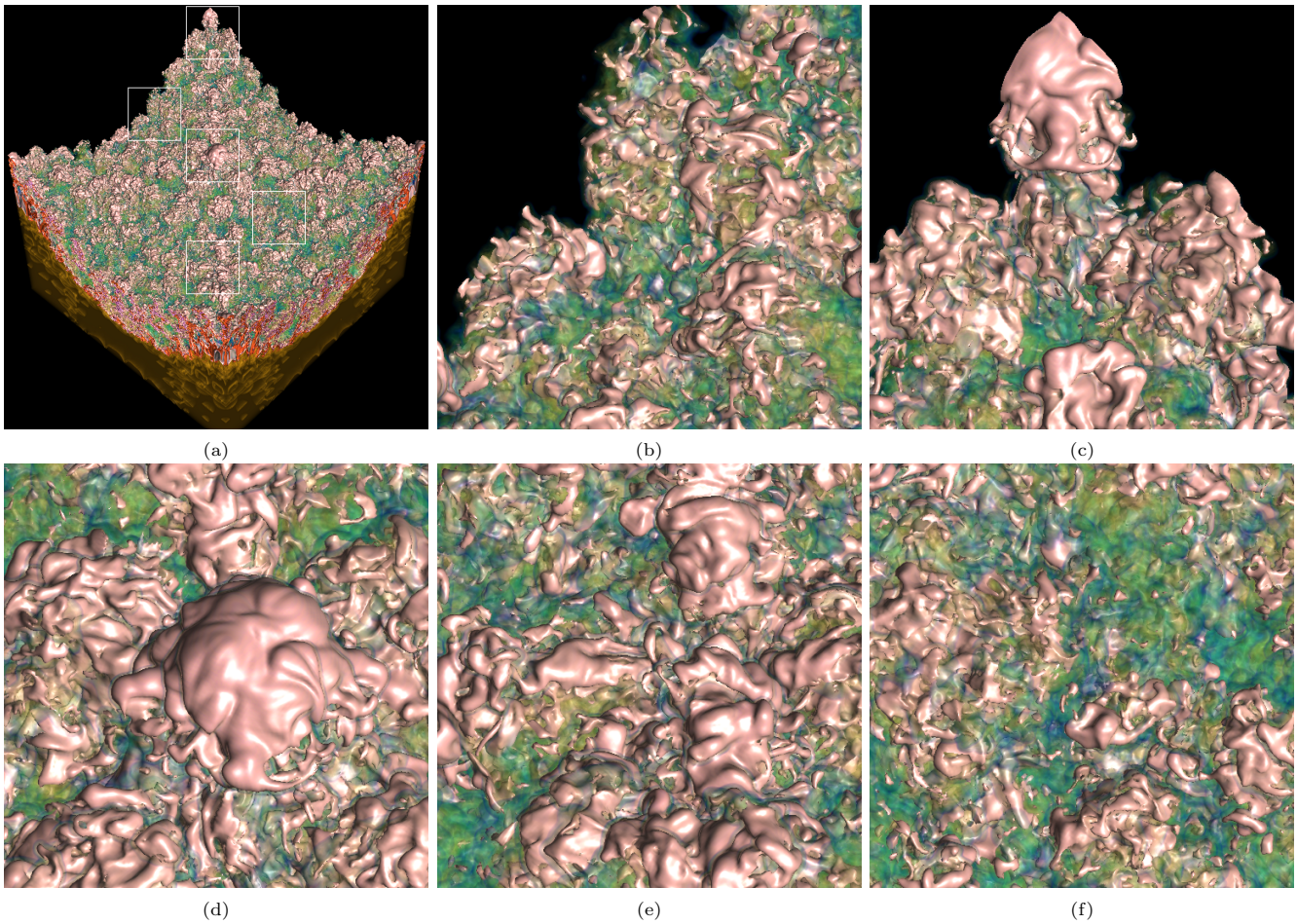


Figure 10: Rendering of a RMI data set. (a): an overview of the data. (b) - (f): five 512^2 zoom-in views cropped from a 4096^2 image output. The high-resolution, high-precision composited images allow clear observation of fine details in the large volume data set.

sors when 503 and 1021 processors are used in compositing. The difference ratios for the total compositing time are 50% and 57% for Figure 9 (a) and (b), respectively.

Our test data set came from the Richtmyer-Meshkov Instability (RMI) simulation, which was made available through the Lawrence Livermore National Laboratory. The RMI occurs when an interface between fluids of differing density is impulsively accelerated, e.g., by the passage of a shock wave. The time-varying RMI data set is over 2TB. It has 274 time steps, each with a spatial resolution of $2048 \times 2048 \times 1920$. Figure 10 shows an overview and several zoom-in views of the rendering of the last time step with an output image resolution of 4096^2. It can be seen that our high-resolution, high-precision parallel image compositing solution delivers high-quality visualization results that enable scientists to observe fine details from the vast amounts of data.

In summary, the 2-3 swap method shows its advantages over the direct send and reduced binary swap methods when image compositing involves a large number of processors and requires a high output image resolution. For large volume visualization, the rendering stage could be very expensive. Using a large number of processors for parallel rendering is preferred since it cuts down the overall cost of the rendering stage dramatically. 2-3 swap provides a natural solution that utilizes all processors participating in rendering for the following image compositing. 2-3 swap makes the maximum use of available computing resources and yields the best overall compositing efficiency.

6. SUMMARY

As we move into the era of petascale computing, it is imperative to have a high-resolution, high-quality, and high-performance solution for parallel rendering of large-scale scientific volume data. In this paper, we present 2-3 swap, a new parallel image compositing algorithm for large data visualization. Unlike the binary swap method, 2-3 swap is very flexible as it can utilize any number of processors for compositing. Every processor participates in all stages of compositing, thus a maximum utilization of parallelism is achieved. On the other hand, our solution is highly scalable compared with the straightforward direct send method. Moreover, the 2-3 swap algorithm is easy to understand and implement. We analyze the complexity of the 2-3 swap algorithm and compare it with the direct send and binary swap methods. We provide detailed performance comparisons and image results gathered with up to 1024 processors on a supercomputer. The experimental results confirm the flexibility and scalability of 2-3 swap. In the future, we plan

to employ optimization methods to reduce the amount of pixel data exchanged and to better balance the compositing calculations. We also would like to experiment with 2-3 swap on other graphics and visualization applications, such as rendering of massive polygon models.

Acknowledgements

This research was supported in part by the NSF through grants CCF-0634913, CNS-0551727, OCI-0325934, and CCF-9983641, and the DOE through the SciDAC program with Agreement No. DE-FC02-06ER25777, DOE-FC02-01ER41202, and DOE-FG02-05ER54817. We are grateful to the anonymous reviewers for their comments, and the NERSC for providing the supercomputing time. The Richtmyer-Meshkov Instability data set was provided by Mark A. Duchaineau at Lawrence Livermore National Laboratory.

7. REFERENCES

[1] J. P. Ahrens and J. S. Painter. Efficient Sort-Last Rendering Using Compression-Based Image Compositing. In *Proceedings of Eurographics Workshop on Parallel Graphics and Visualization 1998*, pages 145–151, 1998.

[2] X. Cavin, C. Mion, and A. Filbois. COTS Cluster-Based Sort-Last Rendering: Performance Evaluation and Pipelined Implementation. In *Proceedings of IEEE Visualization 2005 Conference*, pages 111–118, 2005.

[3] S. Eilemann and R. Pajarola. Direct Send Compositing for Parallel Sort-Last Rendering. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2007*, pages 29–36, 2007.

[4] W. M. Hsu. Segmented Ray Casting for Data Parallel Volume Rendering. In *Proceedings of IEEE Symposium on Parallel Rendering 1993*, pages 7–14, 1993.

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

[6] S. Lombeyda, L. Moll, M. Shand, D. Breen, and A. Heirich. Scalable Interactive Volume Rendering Using Off-the-Shelf Components. In *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2001*, pages 115–121, 2001.

[7] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering. In *Proceedings of Parallel Rendering Symposium 1993*, pages 15–22, 1993.

[8] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications*, 14(4):59–67, 1994.

[9] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[10] U. Neumann. Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers. In *Proceedings of IEEE Symposium on Parallel Rendering 1993*, pages 97–104, 1993.

[11] U. Neumann. Communication Costs for Parallel Volume-Rendering Algorithms. *IEEE Computer Graphics and Applications*, 14(4):49–58, 1994.

[12] J. Nonaka, N. Kukimoto, N. Sakamoto, H. Hazama, Y. Watashiba, X. Liu, M. Ogata, M. Kanazawa, and K. Koyamada. Hybrid Hardware-Accelerated Image Composition for Sort-Last Parallel Rendering on Graphics Clusters with Commodity Image Compositor. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics 2004*, pages 17–24, 2004.

[13] D. Pugmire, L. Monroe, C. C. Davenport, A. DuBois, D. DuBois, and S. Poole. NPU-Based Image Compositing in a Distributed Visualization System. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):798–809, 2007.

[14] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, M. Taveira, S. Hunt, and P. Hanrahan. Lightning-2: A High-Performance Display Subsystem for PC Clusters. In *Proceedings of ACM SIGGRAPH 2001 Conference*, pages 141–148, 2001.

[15] A. Stompel, K.-L. Ma, E. B. Lum, J. P. Ahrens, and J. Patchett. SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. In *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, pages 33–40, 2003.

[16] M. Strengert, M. Magallón, D. Weiskopf, and T. Ertl. Hierarchical Visualization and Compression of Large Volume Datasets Using GPU Clusters. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2004*, pages 41–48, 2004.

[17] A. Takeuchi, F. Ino, and K. Hagihara. An Improved Binary-Swap Compositing for Sort-Last Parallel Rendering on Distributed Memory Multiprocessors. *Parallel Computing*, 29(11–12):1745–1762, 2003.

[18] T. Tu, H. Yu, L. Ramirez-Guzmanz, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In *Proceedings of ACM/IEEE Supercomputing 2006 Conference*, 2006.

[19] B. Wylie, C. Pavlakos, V. Lewis, and K. Moreland. Scalable Rendering on PC Clusters. *IEEE Computer Graphics and Applications*, 21(4):62–70, 2001.

[20] D.-L. Yang, J.-C. Yu, and Y.-C. Chung. Efficient Compositing Methods for the Sort-Last-Sparse Parallel Volume Rendering System on Distributed Memory Multicomputers. In *Proceedings of International Conference on Parallel Processing 1999*, pages 200–207, 1999.

[21] H. Yu, T. Tu, J. Bielak, O. Ghattas, J. C. López, K.-L. Ma, D. R. O'Hallaron, L. Ramirez-Guzmanz, N. Stone, R. Taborda-Rios, and J. Urbanic. Remote Runtime Steering of Integrated Terascale Simulation and Visualization. *HPC Analytics Challenge, ACM/IEEE Supercomputing 2006 Conference*, 2006.