RESEARCH

# mSwap: A Large-Scale Image-Compositing Method with Optimal m-ary Tree

Min Hou[1], Chongke Bi[1*], Fang Wang[2], Liang Deng[2], Gang Zheng[3] and Xiangfei Meng[3]

[*]Correspondence:
bichongkex@tju.edu.cn
[1]College of Intelligence and Computing, Tianjin University, Tian Jin, China
Full list of author information is available at the end of the article
[†]Equal contributor

**Abstract**

With the increasing of computing ability, large-scale science simulations have been generating massive amounts of data in aerodynamics. Sort-last parallel rendering is a proven approach for large-scale science visualization. However, in the stage of image compositing, the sort-last method may suffer from scalability problem on large-scale processors. Existing image compositing algorithms tend to perform well in certain situations. For instance, Direct Send is well on small and medium scale; Radix-k gets well performance only when the $k$-value is appropriate and so on. In this paper, we propose a novel method named mSwap for science visualization in aerodynamics, which uses the best scale of processors to make sure its performance at the best. mSwap groups the processors that we can use with a $(m, k)$ table, which records the best combination of $m$ (the number of processors in subgroup of each group) and $k$ (the number of processors in each group). Then in each group, using a m-ary tree to composite the image for reducing the communication of processors. Finally, the image is composited between different groups to generate the final image. The performance and scalability of our mSwap method is demonstrated through experiments with thousands of processors.

**Keywords:** Parallel rendering; Sort-last; Image compositing; mSwap; m-ary Tree

## Introduction

With the development of HPC systems, science simulations could generate larger amounts of data than before in aerodynamics, which means more powerful technique should be adopted for data analyzing. Visualization aimed at aerodynamics has been playing an important role in scientific discovery, especially in massive data discovery. The increase of data also challenges the traditional methods of science visualization. Parallel rendering is a useful approach to improve the performance of visualization, which has been widely used in scientific visualization to make full use of the HPC systems. As defined by Molnar et al.[1], all parallel rendering approaches are classified into three categories: sort-first, sort-middle and sort-last based on where the sort from object-space to screen space occurs. Among these classifications, sort-first and sort-last are more appropriate for parallel systems than sort-middle, because of their entire rendering pipeline. Besides, sort-last parallel rendering has been widely used in parallel systems for its well performance in load balancing. Different from the other parallel rendering approaches, sort-last has the step of image compositing, that's where the bottleneck is. To settle this problem, several image composition algorithms have been proposed so far.

For better sort-last parallel rendering, image compositing algorithms are used to improve the utilization of processors. The fundamental image compositing algorithms can be divided into three categories according to [2]: Direct Send, Parallel Pipeline and Tree which includes all algorithms based on tree. Direct Send is simple to implement. Every processor will responsible of part of the dataset. However, its complex communication between processors lead to lower performance on massive scale. Parallel Pipeline is seldom found in practice for the same reason. The algorithms based on tree have been improved all these years, such as Binary Tree, Binary Swap, 2-3 Swap, Radix-k, 2-3-4 Decomposition and so on. All of these image compositing algorithms are implemented by an image compositing tree, which is designed using the tree structure. Although, parallel rendering has been studied for so many years, there are still some problems during image compositing.

Firstly, existing image compositing algorithms tend to perform well in some specific situations, which means in most cases, algorithms are not at the best point or even lower the average. The scale of processors is one of the most important factors. With the increasing of processors, the performance of parallel rendering has begun to raise because of the high parallelism. However, once the number of processors reaches a certain value, the performance won't improve again or even decline which is called performance degradation. That is the communication between processors affects the performance. Secondly, most algorithms prefer to keep processors busy to increase the utilization. However high utilization causes frequent communication between processors. Considering that parallel rendering is a communication-intensive approach, this case will lead to performance degradation. Besides that, more processors mean more pieces of image in some image compositing algorithms, which leads to such a phenomenon that each processor is responsible for a few pixels. That will waste the performance of processors. At last, there is an additional image collection step at the end of image composition in most image compositing algorithms. When the image is divided into multiple pieces, the image collection would take a lot of time besides the waste of performance according to [3], which is the next largest time-consume in parallel rendering.

In this paper, we introduce a new image compositing algorithm, called mSwap, for making full use of processors. On the one hand, the advantage of mSwap algorithm is that it can keep each group running in the best situation for avoiding performance degradation. Besides image compositing using m-ary tree could reduce the communication between processors effectively, which is the most time-consume step. Considering the cost of collection, Binary Tree is used at the end of mSwap, which replaces the collection step because of its no image split. On the other hand, the mSwap algorithm is easy to understand and implement, that makes it very practical. After experiment, the algorithm scales very well in a massively parallel environment.

Our algorithm also includes the following contributions:

- An optimal $(m, k)$ table, which indicates the highest performance combination of $m$ and $k$.
- A new image compositing algorithm named mSwap using m-ary tree to reduce the communication of processors.
- A new method to avoid the collection step at the end of image compositing algorithm.

- A new environment of Tianhe-3 prototype to prove the reliability of parallel rendering.

The remainder of this paper is organized as follows: Section  provides several image compositing algorithms. Section  introduces mSwap algorithm in detail. Followed by descriptions of mSwap, the environment and results of experimental in Section . Section  discusses existing technical challenges and proposes orientations for future research. Section  concludes the paper.

## Related work

Parallel technology has been widely used in many fields of scientific research, including visualization. Parallel rendering is a production of the combination of parallel technology and visualization. As mentioned before, parallel rendering is classified as sort-first, sort-middle and sort-last, where sort-last is the most popular for its well performance on HPC systems [4]. In sort-last, each processor loads part of the dataset and renders into a whole image, which means there is no communication between processors before image compositing. Given that each processor only has few blocks of the dataset, the time of loading and rendering could be ignored. On the contrast, during the image compositing each processor must communicate with other processors to produce the final image. Therefore, the cost of communication is very expensive in the image compositing stage. To solve this problem, several algorithms have been proposed for sort-last system.

### Direct Send

One of the most popular algorithms is Direct Send [5], which is intuitive and easy to implement. In sort-last parallel rendering, how to control every processor to communicate with the other processors is the main problem. The simplest idea is that each processor is responsible for a part of the whole image and that is what Direct Send done. In Direct Send, each processor is assigned a partial area of the final image before the image compositing. Then each processor requires to receive data belonging to itself from the other processors, and send data that it is not responsible to corresponding processor. With the completion of sending and receiving, each processor has the final image that it is responsible for. At last, Direct Send collects the image in order to produce the final whole image. Direct Send performs well on small scale or under some specific situation. For example, when the pixels are concentrated relatively, each processor owns the most pixels of its responsibility, that means there is no need to communicate with all of the other processors to get the final image. However, it still has the problem of communication between processors. In the worst situation, Direct Send needs $N(N-1)$ communications to finish the composition, where $N$ is the number of processors. To minimize the communication cost of Direct Send, SLIC [6] (Scheduled Linear Image Composition) algorithm uses a compositing schedule to control the image compositing step. Besides, pixels are classified into three categories: background pixels, pixels in the non-overlapping areas, and pixels in the overlapping areas. Only the second case requires image compositing. By pixels classifying, SLIC could reduce the number of pixels communicating between processors. So SLIC is essentially a highly optimized Direct Send algorithm. There is another reason that Direct Send is still keep alive nowadays. In Direct Send, the process of communication and computing could be parallel completely, which could reduce the time of image compositing further more.

### Parallel Pipeline

Parallel Pipeline [7] is another kind of image compositing algorithm, which is used seldom in practice. Different from Direct Send, processors are organized on a circular ring in Parallel Pipeline, and every processor will send data to the next processor, and receive data from the previous processor. Besides the whole image is divided into $N$ pieces, where $N$ is the number of processors. Each processor is responsible for one piece at a stage, but different stages mean different pieces. After round $N-1$, collection is also required just like Direct Send to produce the final image. Parallel Pipeline is also suitable for small scale of processors, because more processors lead to more communication. Based on Parallel Pipeline, there are several optimal algorithms proposed to improve the performance. PPB (Parallel Pipeline using a Bounding Box) selects active pixels by a bounding box to avoid sending inactive pixels. In sparse image, active pixels tend to stay together, which could be contained in a bounding box to reduce the number of pixels when communicating. DPF (Direct Pixel Forwarding) assigns fixed areas for each processor to avoid link contention just like Direct Send. The difference is DPF still organize processors as a circular ring to exchange data. DPFL (DPF with Static Load Balancing) is advanced optimization of DPF, which assigns horizontal lines among the processors in an interleaved fashion to keep tasks balanced. DPFS (DPF with Task Scheduling) avoids waiting among processors by adding a task schedule, which is caused by rendering out of sync. For the reason that all these algorithms only perform well on small scale, they are not widely used as the others.

### Tree

Last but not least, there are lots of algorithms based on tree, which are popular nowadays. Binary Tree is the basic image compositing algorithm in this category. In Binary Tree, all processors are grouped, and each group has two processors. The key point is one processor sends all data to the other one in the same group to composite in a stage, which means the processor sending data will be idle in the next stage. It could manage all processors better than before, but the idle processors lead to reduce performance to a certain extent. Binary Swap [8], as a well-known algorithm, solves this problem by changing the compositing step. In each stage, one processor sends half of the data to the other processor, and receive the rest from the other one. Then each processor will have half composited image. After $logN$ stages, each processor owns $1/N$ images and collect to one processor finally. By splitting data, Binary Swap improves the utilization of processors. Meanwhile the splitting data also leads to communication problem mentioned before. Considering that both in Binary Tree and Binary Swap, processors are grouped into two, the number of processors must be power-of-two. For supporting arbitrary number of processors, 2-3 Swap [9] groups processors into two or three instead of two only. It is based on that any integer can be expressed as the sum of a sequence of integers consisting of 2 and 3. What should be noted is that the image in processors is not aligned when image compositing between groups. Besides Direct Send is adopted to composite image within and between groups. Radix-k [10] is the result of the generalization of Binary Swap and Direct Send. The number of processors $N$ is factored in $r$ factors so that $k$ is a vector where $k = [k_1, k_2, \ldots, k_r]$. In each stage,

processors are grouped according to $k$ which is the number of processors in each group. However, $k$ is related to the network topology and to physical placement of processes onto that topology, which means there is no clear method to get the $k$ value [11]. The same as 2-3 Swap, Direct Send is used to composite image within and between processors. Given that Binary Swap performs well when the number of processors is power-of-two, 2-3-4 Decomposition [12, 13] is proposed to make sure the number of groups is power-of-two so that Binary Swap could be used between groups.

### The Others

What's more, there are still some image compositing algorithms, which are optimized in special environments based on the algorithms mentioned before.

All the image compositing algorithms perform well in the case of uniform pixels distribution, which is called dense image. However, there are all kinds of possibilities in reality. In some images, pixels may lie in a concentrated area called sparse image [14], which leads to not all of the pixels need to be composited. For example, if active pixels lie in the center of the whole image, it is wasting time to composite the other area of the whole image. To settle the problem, several solutions has been proposed. Bounding box [15] is one of them, which selects the active area using a bounding box to reduce the scale of composition. Interlacing algorithm [3] is another solution, which keeps the load-balancing by interlacing assignment.

Besides, Shift-based [16] image compositing method using shift permutation is based on Infiniband Fat-Tree interconnection network. Multi-Step [17] image compositing approach minimize undesirable performance degradation to achieve scalability. TOD-Tree [18] (Task-Overlapped Direct send Tree) image compositing algorithm combines Direct Send and k-ary Tree to improve the entire performance based on hybrid MPI parallelism. Larsen et al. [19] optimizes for multi-image sort-last parallel rendering. Aly et al. [20] proposes the Distributed kd-trees for retrieval. After that, Zhang et al. [21] designs dynamic load balancing based on constrained K-D tree for parallel particle tracing. Cavin et al. [22] designs a sort-last parallel rendering system based on COTS Cluster. There are also several applications implemented by sort-last parallel rendering [23, 24, 25, 26].

In this paper, we propose a new image compositing algorithm named mSwap, which is also based on tree using the best scale of processors to make sure its performance at the best. The details will be discussed in the next section.

## Methodology

In this section, we will introduce our mSwap algorithm in detail. At the beginning of the compositing step, each processor has an image rendered from partial blocks of the dataset, which has both RGBA value and depth information. All images will be composited into one final image according to the depth information with the image compositing algorithm. Before the description of mSwap, there are some definitions need to be known :

- $N$ : the number of processors.
- $k$ : the number of processors in each group.
- $m$ : the number of processors in subgroup of each group.

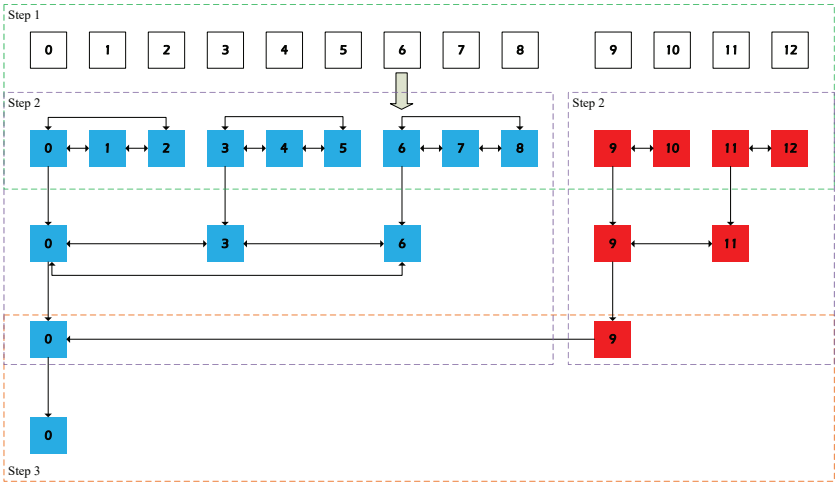



**Figure 1 mSwap contains three steps to finish the image compositing process.** Step 1 : all processors are grouped according to the optimal $(m, k)$ table. Step 2 : image compositing with m-ary tree in each group. Step 3 : image compositing with Binary Tree between groups.

For better description of mSwap, this section is divided into three sections according to the different steps of mSwap algorithm. Section  introduces the strategy of grouping. In Section  image compositing based on m-ary tree will be displayed in detail, and the last Section explains the method for avoiding image collection. The entire process of mSwap is shown in Figure 1.

### The strategy of grouping

As most of the existing image compositing algorithms do, grouping is also required in our algorithm. On the one hand, the independence of dataset makes it possible to group processors. On the other hand, grouping could reduce the scale of processors for easy management. In some way, the performance depends on well grouping. So in mSwap, we group processors using the optimal $(m, k)$ table.

In the classic Binary Swap algorithm, every two nodes are grouped together for data exchanging. However, it is different from that in mSwap. As mentioned before, existing algorithms always perform well in specific case, but in common their behavior is at the average or even under the average, which limits these algorithms greatly. For avoiding this problem, mSwap groups all processors according to the optimal $(m, k)$ table, which makes sure that each group could run at the best situation. There is a point that the performance of every image compositing algorithm is changed with the different scale of processors. At the beginning, with the increase of processors, the advantage of parallel rendering starts to reflect. But there is a threshold, once the number of processors beyond the threshold, the performance of algorithms will decline. Considering that parallel rendering is a communication intensive approach, the increasing communication between processors due to the increasing processors could be the main reason of performance degradation.

It is easy to understand that the performance will decline when the number of processors is too large. An extreme situation is that each processor only responsible for one pixel, but it needs to communicate with the other processors to produce the

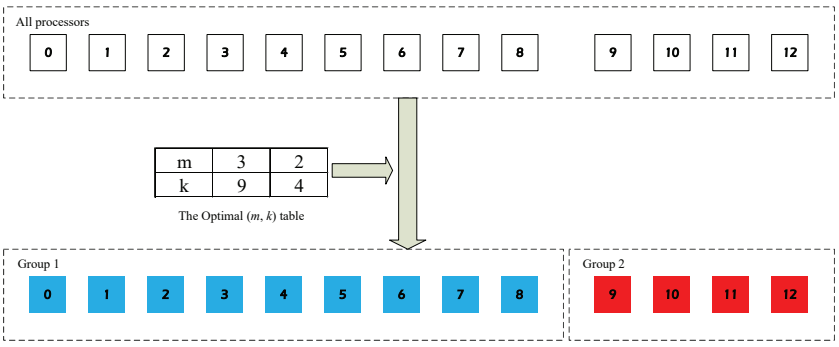



**Figure 2 Grouping according to the optimal $(m, k)$ table.** In the optimal $(m, k)$ table, the combination of $m = 3$, $k = 9$ is better than $m = 2$, $k = 4$. mSwap assigns processors 0 to 8 into the group 1 in priority, and the rest is grouped into group 2.

final image. In that case, so many processors lead to huge communication pressure instead of the performance improvement in parallel rendering. For avoiding performance degradation, mSwap limits the number of processors in each group, and that is what optimal $(m, k)$ table done.

The optimal $(m, k)$ table is the key of grouping in mSwap, which indicates the best scale of processors. In the table, $m$ is the number of processors in subgroup of each group. $k$ is the number of processors of each group, which is also the best scale of processors according to $m$. Since mSwap is based on m-ary tree, there is such a relationship between $m$ and $k$ :

$$n = log_m k \tag{1}$$

where $n$ is positive integer, and $m$ could be any positive integer. Given that in subgroup of each group Direct Send is adopted to composite image which will be introduced in the next section, the value of $m$ should be small to guarantee the performance. Especially when $m$ equals two, then k is the best point of Binary Tree.

In the optimal $(m, k)$ table, the performance of combination is decreasing gradually from left to right, which means the combination on the left is better than that on the right. So, we may consider using the combination in priority to composite image. However, if we keep assigning the previous combination of $m$ and $k$ in advance, the next combination of m and k may not be used forever when the previous $k$ is small. For avoiding this case, except for the priority need to be noted, we also consider the number of rest processors, which should be as small as possible to improve the performance of image compositing.

As shown in Figure 2, in mSwap, the combination on the left is chosen in priority, and then the next combination $(m, k)$ is considered. After traversing the whole table, almost all processors are grouped according to the best scale. Considering that there may be several processors left because of no enough processors for grouping, we assign the left processors to step three in mSwap.

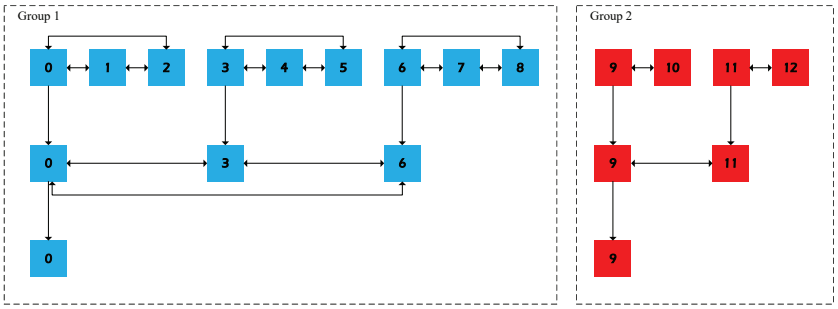



**Figure 3 Image compositing with m-ary tree.** In group 1, image compositing with 3-ary tree, and in group 2, image compositing with 2-ary tree, which is also called binary tree.

### Image compositing with m-ary tree

After grouping, each group will composite image by using m-ary tree to manage processors. The design of mSwap is little different from the existing algorithms that makes it reduce the communication between processors efficiently. And because of that, the performance is improved to a certain degree.

In each group, there is still a lot of processors need to be controlled for image compositing. Thus, the processors in the same group will be grouped again according to $m$, which is the number of processors in subgroup of each group. After the second grouping, each subgroup owns $m$ processors and Direct Send is adopted to composite a whole image among these processors. We still consider some the other image compositing algorithms, but as shown in [5], Direct Send performs better than others on small scale. Then the rest of processors continue to group according to $m$ until there is only one processor on m-ary tree. When step two is done, each group will keep one processor to store the compositing result of the entire group. Although after image compositing by m-ary tree, there is some processors out of computing, the communication between processors reduce efficiently, which is also adopted in other algorithms like 2-3-4 Decomposition.

mSwap is also based on tree to composite image, which makes it easy to manage processors. When $m = 2$, the grouping is just like Binary Tree. The reason that we choose Binary Tree instead of Binary Swap is that the number of pieces keep growing with the scale increasing in Binary Swap, which leads to two potential problems. One is that the greater number of pieces is the more time image collection will take. The other is the increasing pieces results in huge communication between processors, which is the most expensive cost in parallel rendering. For avoiding this, mSwap uses one processor to store the result of each subgroup image compositing, which means in the next stage, each node owns a whole image like the first stage. The details are displayed in Figure 3.

There is another reason that we use m-ary tree to composite image, instead of binary tree in Binary Swap. As for Binary Swap, there must be $log_2 N$ stages to finish image compositing. The same, in mSwap, the number of stages is $log_m N$. When $m > 2$, the number of stages in mSwap is less than that in Binary Swap, which could reduce the compositing time to a certain degree.

In step two of mSwap, m-ary tree is adopted to composite image, which means there will be $n$ m-ary trees, where $n$ is the number of groups in step one. Direct

Send could parallel the process of computing and communicating as mentioned before. However, image compositing algorithms based on tree couldn't parallel the both completely. The next stage of image compositing can only continue until the previous stage is finished. In mSwap, $n$ m-ary trees are mutually independent, which is better than based on one image compositing tree in the other algorithms.

### Binary Tree for avoiding image collection

At the last step of mSwap, there is $n$ processors need to be composited into one final image, where n is the number of groups in step one. Considering Binary Tree is the only one algorithm that does not need image collection at the end of image compositing, mSwap uses Binary Tree to avoid image collection.

In parallel rendering, the image composition time accounts for the vast majority, and the image collection time accounts for the majority of the rest [3]. So the total time of parallel rendering should be like this :

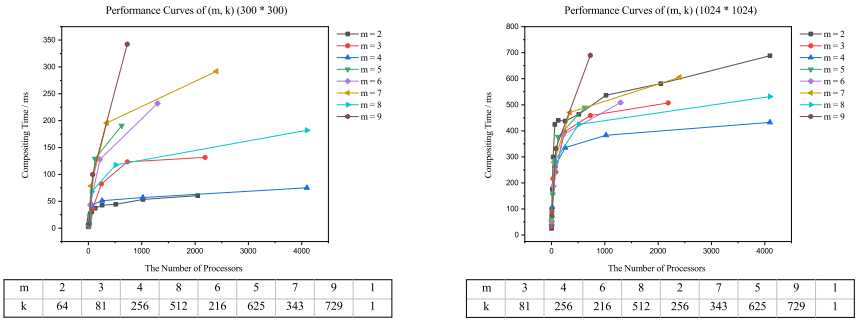
$$t_{total} = t_{load} + t_{composite} + t_{collect} + t_{write} \tag{2}$$

With the increasing of processors, the image collection time keep growing, which effects overall performance in parallel rendering. Considering that $t_{load}$ and $t_{write}$ are far less than $t_{composite}$ or $t_{collect}$, $t_{total}$ is mainly decided by $t_{composite}$ and $t_{collect}$. mSwap reduces the $t_{composite}$ by step two using m-ary tree, which is most of the total time. Besides mSwap reduces the $t_{collect}$ using Binary Tree in step three, which will further increase performance.

Binary Tree is the original image compositing algorithm based on tree, and it is the only algorithm that does not need image collection. In Binary Tree, processors are grouped like Binary Swap. Each group owns two processors. The difference is one processor will send all its data to the other one in the same group to produce the composite image. Although Binary Tree performs not good as the other image compositing algorithm on large scale for its lower processor utilization, it is superior on small scale for its no image collection. This is the reason that mSwap chooses Binary Tree finally. After the composition based on m-ary tree in step two, the number of nodes reduces to a small scale, which fit the characteristic of Binary Tree very well.

## Experimental results

We have implemented proposed mSwap image compositing algorithm using C++ programming language together with VTK [27, 28] (Visualization Toolkit) based on Tianhe-3 prototype. For better performance of mSwap algorithm, we obtain the optimal $(m, k)$ table by testing compositing time at different image resolutions. Besides we also compared our algorithm against Direct Send, Binary Swap, Radix-k and 2-3-4 Decomposition, which are also designed based on VTK. As for 2-3 Swap, considering that Radix-k is a more general result of 2-3 Swap, we choose Radix-k for testing.




| m | 2 | 3 | 4 | 8 | 6 | 5 | 7 | 9 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| k | 64 | 81 | 256 | 512 | 216 | 625 | 343 | 729 | 1 |

(a)

| m | 3 | 4 | 6 | 8 | 2 | 7 | 5 | 9 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| k | 81 | 256 | 216 | 512 | 256 | 343 | 625 | 729 | 1 |

(b)

**Figure 4 Performance curves of $(m, k)$ with a resolution of $300 * 300$ and $1024 * 1024$.** The performance curves of $(m, k)$ is used to locate the best point of m-ary tree. With the number of processors increasing, compositing time also rises until the number of processors reaches a threshold.

### Tianhe-3 prototype

Tianhe-3 prototype has a high-performance cluster environment, which is designed for high-performance computing and massive data processing. For better computing capability, it provides two kinds of CPUs, which includes Phytium MT2000+ and Phytium FT2000+. Tianhe-3 prototype owns 512 boards with three Phytium MT2000+ CPUs on each board, and 128 boards with four Phytium FT2000+ CPUs on each board. Each Phytium MT2000+ CPU is divided into four nodes, which owns 32 cores and 16GB RAM. Each Phytium FT2000+ CPU owns 64 cores and 64GB RAM. Besides, the floating-point computing performance of Tianhe-3 prototype could reach 3.146PFlops. The capacity of total parallel storage is 1PB, which could better meet the needs of users.

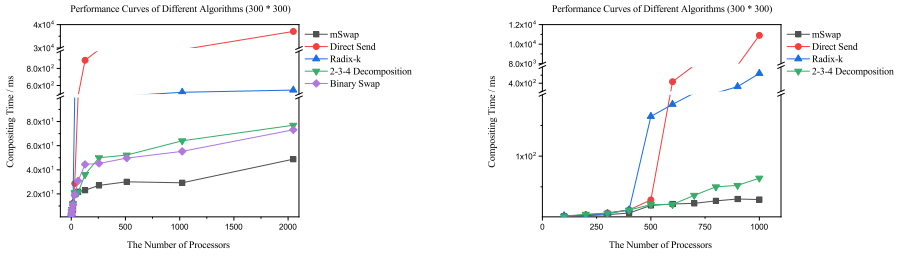
### The optimal $(m, k)$ table

In sort-last system, each processor owns a full image information before image compositing, which means the resolution of image decides the pressure of communication between processors. Therefore, different resolutions mean different optimal combination of $m$ and $k$. we firstly select the image resolution to $300 * 300$ to test the optimal combination, which is the default window size in VTK. Besides the resolution of image, the relationship between $m$ and $k$ also need to be considered. When testing, we must to make sure that $k$ is the power of $m$ as mentioned in section 3.

In order to get the optimal $(m, k)$ table, we did a lot of testing under different combination of $m$ and $k$. With the increasing of $k$, the time of image compositing keep growing, which is caused by communication between processors. However, once the $k$ value beyond a threshold, the performance of parallel rendering has begun to decline. For finding the threshold, we define the growth rate of image compositing time :

$$v = \delta t / \delta k \tag{3}$$

Where $\delta t$ is the change of time and $\delta k$ is the change of number of processors. As beginning, the time of image compositing grows quickly because of the need of




(a) (b)

**Figure 5 Performance curves of different algorithms with a resolution of $300 * 300$.** (a) is the compositing time when the number of processors is power-of-two. (b) is the compositing time when the number of processors is non power-of-two without Binary Swap.

parallel rendering, which means $v$ is a large value. After that, $v$ has begun to slow down, which means the time of rendering is also stable. To make sure the proper $k$ value, we select the smallest $v$ as the best point, which is also the best point of $k$.

There is another point that how to choose the $k$ value in Radix-k algorithm. In [10], it is mentioned that there is no exact way to get the appropriate $k$ value, but when fairly large radices such as 32 or 16 appears in the $k$ vector, the performance is better. So in Radix-k based on VTK, we prefer to choose 32 and 16 as the $k$ value.

Figure 4a shows the optimal $(m, k)$ table and the performance of each combination of $m$ and $k$. After that, the larger resolution of image need to be considered. So, we also test the performance when compositing images of sizes $1024 * 1024$. Figure 4b shows the optimal $(m, k)$ table under different image sizes.

The optimal $(m, k)$ table is the key to the mSwap algorithm, which indicates the process of grouping. Considering that Direct Send is used in each subgroup of each group. The value of $m$ is set from 1 to 9, where 1 is for the rest processors that are not grouped. In the best situation, the number of ungrouped processors is zero, which means all of the processors are grouped according to the best point. Although, it is possible that there are unassigned processors left, which is sent to step three for Binary Tree compositing, mSwap still performs well under most circumstances. After m-ary tree compositing in step two, the number of processors has declined to a smaller scale, which is suitable for Binary Tree. So, the performance of mSwap won't fall under any conditions.

### The performance of mSwap

We also execute our mSwap algorithm based on the optimal $(m, k)$ table mentioned before. Since existing algorithms were implemented based on VTK, we compared our algorithm with the other image compositing algorithms in the same situation.

There are two factors that affect the performance of image compositing. One is the resolution of image, which affects the optimal $(m, k)$ table directly. We use the default size of image $(300 * 300)$ to test firstly. After that, to prove the effectiveness on larger resolution, we also test the image size of $1024 * 1024$. The other is the number of processors. mSwap algorithm supports any number of processors, so we still compare mSwap with the other algorithms when the number of processor is

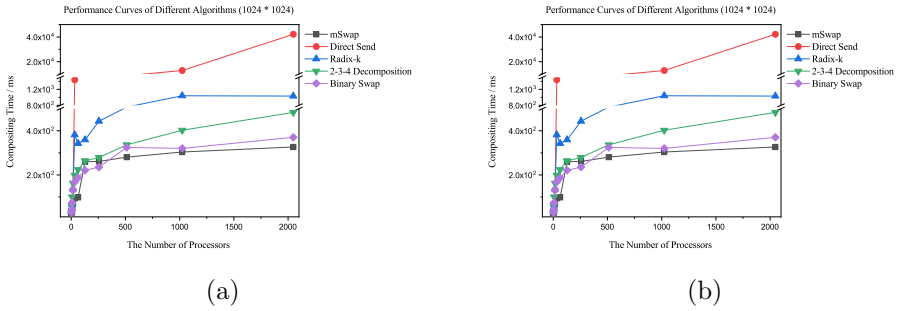



**Figure 6 Performance curves of different algorithms with a resolution of $1024 * 1024$.** (a) is the compositing time when the number of processors is power-of-two. (b) is the compositing time when the number of processors is non power-of-two without Binary Swap.

non power-of-two. However, Binary Swap is not tested because it does not support the case where the number of processors is non power-of two.

Figure 5 shows the different performance curves of different algorithms when the image size is $300 * 300$, and Figure 6 is the result with a resolution of $1024 * 1024$. From the result, we could see that mSwap is better than the other algorithms no matter the number of processors is power-of-two or not.

The performance of mSwap is well under most situations. When the number of processors has begun to grow, the compositing time increases quickly like the other image compositing algorithms. However, the upper limit of time in mSwap is lower than other algorithms for its well performance. Direct Send is popular for its simple and abandoned for its bad behavior on large scale. Thus, Direct Send is more appropriate for mixing with the other algorithms, which could provide better environment for Direct Send. Although Binary Swap has been proposed for so many years, it is also popular when the number of processors is power-of-two. It could make full use of processors to composite image, which leads to performance improvement. 2-3-4 Decomposition algorithm is based on Binary Swap, which add pre-processing to make sure any number of processors could use Binary Swap to composite image. Therefore, when the number of processors is power-of-two, its performance is slightly worse than Binary Swap on large scale because of the pre-processing progress. What's more, Radix-k is the combination of Direct Send and Binary Swap, which is affected by the value of $k$ greatly. Although we choose the better $k$ value, it is little worse than Binary Swap and 2-3-4 Decomposition. Compared to these image compositing algorithms, mSwap shows its advantages both on small or large scale. On the one hand, the optimal $(m, k)$ table provides the best situation when using m-ary tree to composite image, which could keep the processors at the best point or above the average. On the other hand, mSwap composites image by m-ary tree, which reduce the communication between processors. For example, if $m = 2$ is the best combination in the optimal $(m, k)$ table, mSwap prefer to use Binary Tree for image compositing. It means that mSwap improve the performance of each group by using the optimal $(m, k)$ table to perform better than other algorithms.

Besides the performance, the scalability of mSwap is better than other image compositing algorithms. When the number of processors keep growing, mSwap adjusts

the result of grouping according to the optimal $(m, k)$ table, which could resize the scale of each group for better performance. However, the other image compositing algorithms can not adjust grouping flexibly. In Direct Send, the communication between processors increases rapidly with the increasing number of processors, which causes the worse performance than the others. Binary Swap performs well when the number of processors is power-of-two, but it is not support for the other situations. As an improved algorithm of Binary Swap, 2-3-4 Decomposition shows its well scalability. By pre-processing, it makes sure the number of processors is power-of-two. Then Binary Swap is adopted to get high performance. Radix-k is a configurable image compositing algorithm, the value of $k$ is the key to the performance. Radix-k uses the vector $k$ to group in each stage, which is a bit like mSwap. The difference is Radix-k could only group according to each $k$ value, but mSwap could adjust the grouping according to the optimal $(m, k)$ table. It is like that Radix-k owns a dynamic vector $k$, which indicates the best grouping under different number of processors.

## Challenges

In this section, several possible research challenges will be introduced.

With the developing of HPC, there are various of environments both in hardware and software. Special optimization for different environments becomes a challenge gradually, such as Shift-based image composition, Multi-Step image composition, TOD-Tree image composition and so on. Considering that parallel rendering has a variety of application environments, it is important to improve image compositing algorithms based on the specific environments.

For better scientific analysis, the speed of visualization is very important, especially involved real-time data. To improve the speed of visualization, the utilization of processors needs to increase, the communication between processors need to reduce and the performance of processors needs to keep above the average at least. To solve this problem, one method named in-situ visualization was mentioned recently. By in-situ visualization, researchers will visualize the intermediate data in situ for avoiding data movement, which could improve the efficiency of scientific research in some way. However, there is still a lot of problems need to be done in in-situ visualization [29, 30].

## Conclusions

In this paper, we propose an image compositing algorithm called mSwap, which aims to find the best case of processors according to the performance curves.

In the past two decades, various of image compositing algorithms have been proposed to improve the performance of parallel rendering by increasing the utilization of processors. Most of them are designed based on tree for its high parallelism, and their sort-last system could provide better load-balancing. However, existing algorithms perform well only in some specific situation that means algorithms are at the average or even under the average in the most cases.

mSwap is proposed to settle this problem. Firstly, mSwap groups all the processors according to the optimal $(m, k)$ table for making full use of every processor. Considering that each algorithm has its own characteristic, we make sure processors

running at their best point by using the table containing the best scale of processors which is obtained by testing. Secondly, instead of using the existing image compositing algorithms, we composite image based on m-ary tree in group, which reduces the communication between processors in some way. Finally, Binary Tree is used to composite images in the third phase, because of its high performance on small scale and no image collection which is different from the other image compositing algorithms. Furthermore, it is proved that mSwap performs better than the other image compositing algorithms in a large-scale situation. Future works includes the further optimization for mSwap based on current challenges, which makes mSwap better adapt to special environment.

**Availability of data and materials**
The data used in this article is in tecplot format, any data in tecplot format can be used as the input of the source code.

**Competing interests**
Not applicable.

**Funding**
This work was partly supported by the National Natural Science Foundation of China under Grant No. 61702360.

**Author's contributions**
Min Hou and Chongke Bi designed and implemented the image compositing algorithm. Fang Wang and Liang Deng analyzed and extracted data from the simulation process. Gang Zheng and Xiangfei Meng provided assistance in cluster environment.

**Acknowledgements**
The work was carried out at National Supercomputer Center in Tianjin, and the calculations were performed on Tianhe 3 prototype.

**Author details**
[1]College of Intelligence and Computing, Tianjin University, Tian Jin, China. [2]Computational Aerodynamics Institute, China Aerodynamics Research and Development Center, Mian Yang, China. [3]National Supercomputer Center in Tianjin, Tian Jin, China.

**References**
1. Molnar, S., Cox, M., Ellsworth, D., Fuchs, H.: A sorting classification of parallel rendering. IEEE computer graphics and applications **14**(4), 23–32 (1994)
2. Bethel, E.W., Childs, H., Hansen, C.: High Performance Visualization: Enabling Extreme-scale Scientific Insight. CRC Press, New York (2012)
3. Moreland, K., Kendall, W., Peterka, T., Huang, J.: An image compositing solution at scale. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–10 (2011)
4. Nonaka, J., Ono, K., Miyachi, H.: Performance evaluation of large-scale parallel image compositing on a t2k open supercomputer. Information and Media Technologies **4**(4), 780–788 (2009)
5. Eilemann, S., Pajarola, R.: Direct send compositing for parallel sort-last rendering, 29–36 (2007)
6. Stompel, A., Ma, K.-L., Lum, E.B., Ahrens, J., Patchett, J.: Slic: scheduled linear image compositing for parallel volume rendering. In: IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003. PVG 2003., pp. 33–40 (2003). IEEE
7. Lee, T.-Y., Raghavendra, C.S., Nicholas, J.B.: Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. IEEE Transactions on Visualization and Computer Graphics **2**(3), 202–217 (1996)
8. Ma, K.-L., Painter, J.S., Hansen, C.D., Krogh, M.F.: Parallel volume rendering using binary-swap compositing. IEEE Computer Graphics and Applications **14**(4), 59–68 (1994)
9. Yu, H., Wang, C., Ma, K.-L.: Massively parallel volume rendering using 2–3 swap image compositing. In: SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 1–11 (2008). IEEE
10. Peterka, T., Goodell, D., Ross, R., Shen, H.-W., Thakur, R.: A configurable algorithm for parallel image-compositing applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–10 (2009). IEEE
11. Kendall, W., Peterka, T., Huang, J., Shen, H.-W., Ross, R.B.: Accelerating and benchmarking radix-k image compositing at large scale. EGPGV **10**, 101–110 (2010)
12. Nonaka, J., Bi, C., Fujita, M., Ono, K.: 2-3-4 decomposition method for large-scale parallel image composition with arbitrary number of nodes. In: SIMS 2014: International Conference on Systems Informatics, Modelling and Simulation, pp. 59–64 (2014)
13. Nonaka, J., Ono, K., Fujita, M.: 234compositor: A flexible parallel image compositing framework for massively parallel visualization environments. Future Generation Computer Systems **82**, 647–655 (2018)

14. Yang, D.-L., Yu, J.-C., Chung, Y.-C.: Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers. The Journal of Supercomputing **18**(2), 201–220 (2001)
15. Takeuchi, A., Ino, F., Hagihara, K.: An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. Parallel Computing **29**(11-12), 1745–1762 (2003)
16. Cavin, X., Demengeon, O.: Shift-based parallel image compositing on infiniband tm fat-trees, 129–138 (2012)
17. Nonaka, J., Fujita, M., Ono, K.: Multi-step image composition approach for sort-last massively parallel rendering. Journal of Advanced Simulation in Science and Engineering **2**(1), 108–125 (2015)
18. Grosset, A.P., Prasad, M., Christensen, C., Knoll, A., Hansen, C.D.: Tod-tree: Task-overlapped direct send tree image compositing for hybrid mpi parallelism. EGPGV **15**, 67–76 (2015)
19. Larsen, M., Moreland, K., Johnson, C.R., Childs, H.: Optimizing multi-image sort-last parallel rendering. In: 2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV), pp. 37–46 (2016). IEEE
20. Aly, M., Munich, M., Perona, P.: Distributed kd-trees for retrieval from very large image collections. In: Proceedings of the British Machine Vision Conference (BMVC), vol. 17 (2011)
21. Zhang, J., Guo, H., Hong, F., Yuan, X., Peterka, T.: Dynamic load balancing based on constrained kd tree decomposition for parallel particle tracing. IEEE transactions on visualization and computer graphics **24**(1), 954–963 (2017)
22. Cavin, X., Mion, C., Filbois, A.: Cots cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In: VIS 05. IEEE Visualization, 2005., pp. 111–118 (2005). IEEE
23. Muraki, S., Ogata, M., Ma, K.-L., Koshizuka, K., Kajihara, K., Liu, X., Nagano, Y., Shimokawa, K.: Next-generation visual supercomputing using pc clusters with volume graphics hardware devices. In: SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, pp. 44–44 (2001). IEEE
24. Moreland, K., Wylie, B., Pavlakos, C.: Sort-last parallel rendering for viewing extremely large data sets on tile displays. In: Proceedings IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (Cat. No. 01EX520), pp. 85–154 (2001). IEEE
25. Eilemann, S., Makhinya, M., Pajarola, R.: Equalizer: A scalable parallel rendering framework. IEEE transactions on visualization and computer graphics **15**(3), 436–452 (2009)
26. Biedert, T., Werner, K., Hentschel, B., Garth, C.: A task-based parallel rendering component for large-scale visualization applications. In: EGPGV, pp. 63–71 (2017)
27. Avila, L.S., Barre, S., Blue, R., Geveci, B., Henderson, A., Hoffman, W.A., King, B., Law, C.C., Martin, K.M., Schroeder, W.J.: The VTK User's Guide. Kitware New York, New York (2010)
28. Hanwell, M.D., Martin, K.M., Chaudhary, A., Avila, L.S.: The visualization toolkit (vtk): Rewriting the rendering code for modern graphics cards. SoftwareX **1**, 9–12 (2015)
29. Ma, K.-L.: In situ visualization at extreme scale: Challenges and opportunities. IEEE Computer Graphics and Applications **29**(6), 14–19 (2009)
30. Childs, H., Bennett, J.C., Garth, C., Hentschel, B., Rhyne, T.: In situ visualization for computational science. IEEE Computer Graphics and Applications **39**(6), 76–85 (2019)