

Chapter 5

Parallel Image Compositing Methods

Tom Peterka

Argonne National Laboratory

Kwan-Liu Ma

University of California at Davis

5.1	Introduction	13
5.2	Basic Concepts and Early Work in Compositing	14
	5.2.1 Definition of Image Composition	15
	5.2.2 Fundamental Image Composition Algorithms	17
	5.2.3 Image Compositing Hardware	19
5.3	Recent Advances	19
	5.3.1 2-3 Swap	20
	5.3.2 Radix-k	20
	5.3.3 Optimizations	22
5.4	Results	23
5.5	Discussion and Summary	26
	5.5.1 Conclusions	26
	5.5.2 Directions for Future Research	27
	Acknowledgment	28
	Bibliography	29

Image compositing is a fundamental part of high performance visualization on large-scale parallel machines. Aside from reading a dataset from storage, compositing is the most expensive part of the parallel rendering pipeline, because it requires communication among a large number of processes. On a modern supercomputer, compositing may generate literally hundreds of thousands of messages. Thus, developing compositing algorithms that scale with growing machine size is of crucial importance. Such algorithms have enabled, for example, wall-size images that are tens of megapixels in resolution to be composited at interactive frame rates from all of the nodes of some of the world's largest supercomputers and visualization clusters. We begin with a history of classic parallel image compositing algorithms: direct-send and binary-swap. From there we move to optimizations that have been proposed over the years, from scheduling to compression and load balancing. Advanced compositing on mod-

ern supercomputing architectures, however, is the main topic of this chapter, in particular 2-3 swap and radix-k for petascale HPC machines.

5.1 Introduction

The motivation for studying image compositing is the same as in most of this book: datasets consisting of trillions of grid points and thousands of time steps are being produced by machines with hundreds of thousands of cores, with hundreds of millions of cores expected by the end of this decade. Machine architectures are growing not only in size, but in complexity, with multidimensional topologies and specialized hardware such as smart network adaptors, direct memory access, and graphics accelerators. Against this backdrop, scientists in high-energy physics, climate, and other domains are demanding more of visualization: real-time, multivariate, time-varying methods that maximize data locality and minimize data movement.

Data size, problem complexity, system size, and science expectations all demand that visualization be done in parallel using sort-last parallel rendering, sometimes at the full scale of a high-performance supercomputer. This means that each parallel process generates a finished image of its subset of the data, and these images must be combined into one final result. The composition of these images is the critical last step in parallel rendering and the subject of this chapter.

Legacy image compositing algorithms were were invented for much smaller systems. Beginning with the mid 1990s, we will review the direct-send and binary-swap algorithms. Optimizations such as compression, identification of active pixels, and scheduling further improved performance. In the early 2000s, production compositing libraries implementing many of these features appeared, and they remain in use today.

New advances in the last five years have featured more architecture awareness of HPC systems. The late 2000s yielded compositing algorithms with higher degrees of concurrency, scalability, and flexibility in the form of 2-3 swap and radix-k algorithms, and the early 2010s showed continued optimization at very large scale and implementation of latest innovations in production. We will highlight some results of these recent advances with both theoretical and actual performance, and the chapter concludes with a look ahead to future directions in highly parallel image compositing.

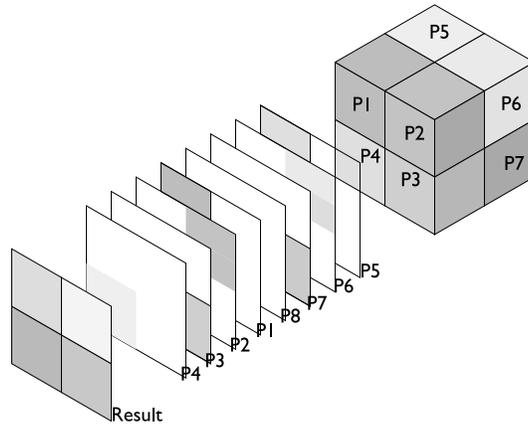


FIGURE 5.1: Sort-last image compositing

5.2 Basic Concepts and Early Work in Compositing

We begin with a review of parallel rendering followed by a formal definition of image compositing. Next, we review classical image compositing algorithms and optimizations. Throughout this chapter, we will use the term *process* to designate a task executed in parallel with other processes, where processes each have separate memory address spaces and communicate via passing messages.

The previous chapter classified parallel rendering according to when rasterized images are sorted [17]: sort-first, sort-middle, and sort-last. One way to understand the difference in these methods is to identify what is distributed and what is replicated among processes.

In sort-first rendering, the image pixels are usually distributed, and the dataset is typically replicated. HPC applications generate datasets many times larger than the memory capacity of a single node, so sort-first rendering is not frequently encountered in high-performance visualization. Sort-middle rendering attempts to combine the best of sort-first and sort-last with both image pixels and data voxels distributed among processes. This is difficult to implement and even harder to scale, because the portion of the data that a process needs to maintain depends on the viewpoint and requires redistribution when the view frustum changes.

This leaves sort-last rendering as the only practical approach to large-scale parallel rendering. Data are distributed while pixels are replicated, and each process is responsible for rendering a complete image of its subset of data. Of course, sort-last rendering comes at a cost, and that price is paid in compositing when the pixels at each process must be combined into one final image. The basic idea is illustrated in Figure 5.1. The dataset is partitioned

among processes; rendering occurs locally at each process, and the resulting images are composited at the end.

5.2.1 Definition of Image Composition

Image composition is the reduction of several images into one. It involves two subproblems: communicating two or more images to one location where the reduction can occur and the computation of the reduction itself. The formal definition of image composition, therefore, is a definition of each of these subproblems, beginning with the reduction operator.

Reduction is a binary operation, where one of the inputs is the current value of the pixel (p_{old}), and the other is an incoming value (p_{new}), accumulated with p_{old} to produce an updated value: $p = f(p_{old}, p_{new})$. The exact definition of f depends on the application. For example, f can select the pixel with the closer depth value or the greater intensity, or it can be a combination of the two pixel values. A common example is the *over* operator [23], a linear combination of p_{old} and p_{new} , where $p = (1.0 - \alpha_{old}) * p_{new} + p_{old}$. Here, p represents the pixel color and opacity components, each computed separately, and α is the pixel opacity only.

Because f often depends on the depth order of blended input pixels, f is assumed to be noncommutative for the general case of compositing semi-transparent pixels. In general, $f(p_{old}, p_{new}) \neq f(p_{new}, p_{old})$, and moreover, if the final value of f is derived from a sequence (functional composition) of individual operations, $f = f_1 \circ f_2 \circ \dots \circ f_n$, then the ordering of f_1 through f_n cannot be permuted without changing the value of f . Image composition is associative, however, so $(f_1 \circ f_2) \circ f_3 = f_1 \circ (f_2 \circ f_3)$. The remainder of this chapter addresses the communication subproblem, beginning with the problem definition.

A valid image composition requires that the final value of a pixel is derived from values of the same pixel on all processes. At first glance this may appear to be an all-to-all communication pattern, with p^2 messages sent and received among p number of processes. Because f is accumulated via individual f_i as shown above, however, each process can contribute to the final value f without communicating directly with every other process. Hence, a simple linear communication pattern with $p - 1$ messages suffices.

We can often do better by parallelizing this sequence with tree and pipeline communication patterns that trade fewer than $p - 1$ communication steps (or *rounds*) for a greater total number of messages, where each round involves multiple messages. The goal is for these multiple messages per round to be independent and for the network to support their concurrent transmission, thereby reducing the total communication cost to $O(\log p)$.

While the final image is often gathered to a single process, this is not strictly necessary, and a distributed result is not only acceptable but often desirable. In message-passing literature, this is an example of a reduce-scatter noncommutative collective, and many algorithms for such collectives have

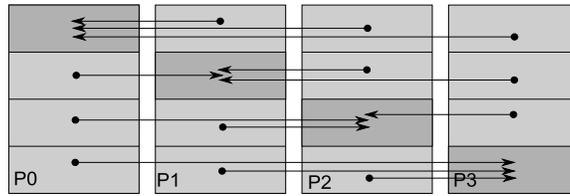


FIGURE 5.2: Example of the direct-send algorithm for 4 processes.

been published [4, 3, 31, 2, 30, 25, 8, 5, 12, 13]. As we will see later in this chapter, we can do even better with top-down knowledge of image composition and of current HPC architectures.

The representation of pixels in terms of number of bytes and whether to use integer or floating-point depends on the implementation, but because the value f can accumulate over many rounds, 4-byte floating-point precision for each channel is advisable. A depth value per pixel may also be needed in some circumstances.

5.2.2 Fundamental Image Composition Algorithms

Stompel et al. [28] surveyed methods for sort-last compositing, and Cavin et al. [6] analyzed relative theoretical performance of these methods. These overviews show that compositing algorithms usually fall into one of three categories: direct-send, tree, and parallel pipeline. Pipeline methods are seldom found in practice and are not covered here.

In direct-send, each of p processes takes ownership of $1/p$ of the final image, and all other processes send this subset of their images to the owner [10, 21]. Figure 5.2 shows an example of four processes executing a direct-send. The outermost rectangles represent each of the original images prior to compositing. The smaller highlighted rectangles represent the final image portions owned by each process after compositing. The arrows represent transmission of image pieces from sender to receiver. If process P_0 is the owner of the top $1/4$ of the image, then P_1 , P_2 , and P_3 each send the top $1/4$ of their image to P_0 to composite. The same thing happens at the other processes for the other three quarters of the image.

When p is large, direct-send can congest the network with many simultaneous messages. One remedy for relieving network contention is to use a tree for compositing. Figure 5.3 illustrates a hypothetical tree with a variety of group sizes. Tree methods mete out the work in levels of the tree, which we call rounds, and in each round, images are exchanged between a small number of processes. While direct-send tries to do as much work as quickly as possible by generating the maximum number of messages in a single round, tree methods strive for a measured approach that generates fewer simultaneous messages over more rounds. We will see later that trees can be designed

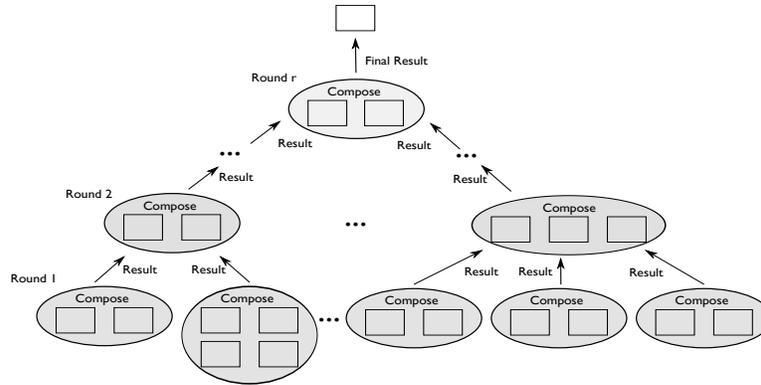


FIGURE 5.3: Tree-based compositing.

to span a broad range of this spectrum, but first let us consider perhaps the best-known binary tree compositing algorithm, binary-swap.

Ordinary tree compositing, as Figure 5.3 shows, causes many of the processes that were busy in early rounds to idle as execution proceeds to later rounds. Eventually, at the root of the tree, one process is performing the entire composition sequentially. To solve this bottleneck and keep all processes busy, Ma et al. [15] introduced the binary-swap algorithm. Figure 5.4 shows a binary-swap example of four processes and two rounds. Each process composes the incoming portion of the image with the same part of the image that it already owns. By continually swapping, all processes remain busy throughout all rounds. In each round, neighbors are chosen twice as far apart, and image portions exchanged are half as large as in the previous round. This is why binary-swap is also called a distance-doubling and vector-halving communication algorithm in some contexts.

Since the mid 1990s when direct-send and binary-swap were first published, numerous variations and optimizations to the basic algorithms have appeared. The basic ideas are to reduce active image regions using the spatial locality and sparseness present in many scientific visualization images, to better balance load after such reduction, and to keep network and computing

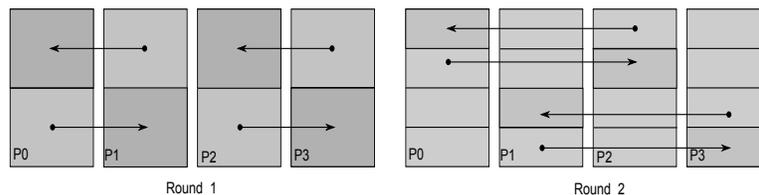


FIGURE 5.4: Example of the binary-swap algorithm for 4 processes and 2 rounds.

resources appropriately loaded through scheduling. We highlight examples of each of these ideas below.

Run-length encoding images achieves lossless compression [1], and using bounding boxes to identify the nonzero pixels is another way to reduce the active image size [15]. These optimizations can minimize both communication and computation costs.

Load balancing via scan line interleaving [29] assigns individual scan lines to processes such that each process is assigned numerous disconnected scan lines from the entire image space. In this way, active pixels are distributed more evenly among processes, and workload is better balanced. The drawback is that the resulting image must be rearranged once it is composited, which can be expensive for large images.

The SLIC [28] algorithm combines direct-send with active pixel encoding, scan-line interleaving, and scheduling of operations. Spans of compositing tasks are assigned to processes in an interleaved fashion. Another way to schedule processes is to assign them different tasks. This is the approach taken by Ramakrishnan et al. [26], who scheduled some processes to perform rendering while others were assigned to compositing. The authors presented an optimal linear-time algorithm to compute this schedule.

Image compositing has also been combined with parallel rendering for tiled displays. The IceT library, from Moreland et al. [19], performs sort-last rendering on a per-tile basis. Within each display tile, the processors that contributed image content to that tile perform either direct-send or binary-swap compositing. Although the tile feature of IceT is not used much in practice, IceT has become a production-quality library that offers a robust suite of image compositing algorithms to scientific visualization tools. Both ParaView [32] and VisIt [9] use the IceT library for image compositing.

5.2.3 Image Compositing Hardware

While this chapter primarily studies the evolution of software compositing algorithms, it is worth noting that hardware solutions to the image compositing problem exist as well. Some of these have been made commercially available on smaller clusters, but as the interconnects and graphics hardware on visualization and HPC machines have improved over time, it has become more cost-effective to use these general-purpose machines for parallel rendering and image compositing rather than purchasing dedicated hardware for these tasks.

Sepia [16] is one example of a parallel rendering system that included PCI-connected FPGA boards for image composition and display. Lightning-2 [27] is a hardware system that received images from the DVI outputs of graphics cards, composited the images, and mapped them to sections of a large tiled display. Muraki et al. [20] described an eight-node Linux cluster equipped with dedicated volume rendering and image composition hardware. In a more recent

system, the availability of programmable network processing units accelerated image compositing across 512 rendering nodes [24].

5.3 Recent Advances

Although the classic image composition algorithms and optimizations have been used for the past fifteen years, new processor and interconnect advances such as direct memory access and multidimensional network topology present new opportunities to improve the state of the art in image compositing.

We hinted earlier at the relationship between direct-send and binary-swap through a tree-based representation. We will explain that connection now and use it to develop more general algorithms that combine both techniques. Two recent algorithms will be presented as examples of more general communication patterns that can exploit new hardware: 2-3 swap and radix-k.

5.3.1 2-3 Swap

Yu et al. [33] extended binary-swap compositing to nonpower-of-two numbers of processors with an algorithm they called 2-3 swap. One goal of this algorithm is to combine the flexibility of direct-send with the scalability of binary-swap, and the authors accomplished this by recognizing that direct-send and binary-swap are related and can be combined into a single algorithm.

Any natural number greater than one can be expressed as the sum of twos and threes, and this property is used to construct the first-round group assignment in the 2-3 swap algorithm. The algorithm proceeds to execute a sequence of rounds with group sizes that are either between two and five. Each round can have multiple group sizes present within the same round. The number of rounds r is equal to the floor of $\log p$, where p is the number of processes and need not be a power of two, as is the case for binary-swap.

An example of 2-3 swap using 7 processes is shown in Figure 5.5. In the first round, shown on the left, processes form groups in either twos or threes, as the name 2-3 swap suggests, and execute a direct-send within each group. (Direct-send is the same as binary-swap when the group size is two.) In the second round, shown on the right, the image pieces are simply divided into a direct-send assignment, with each process owning $1/p$, or $1/7$ in this example, of the image. By assigning *which* $1/7$ each process owns, however, Yu et al. proved that the maximum number of processes in a group is five, avoiding the contention in ordinary direct-send. Figure 5.5 shows that indeed, process P_5 receives messages from four other processes, while all other processes receive messages from two or three others.

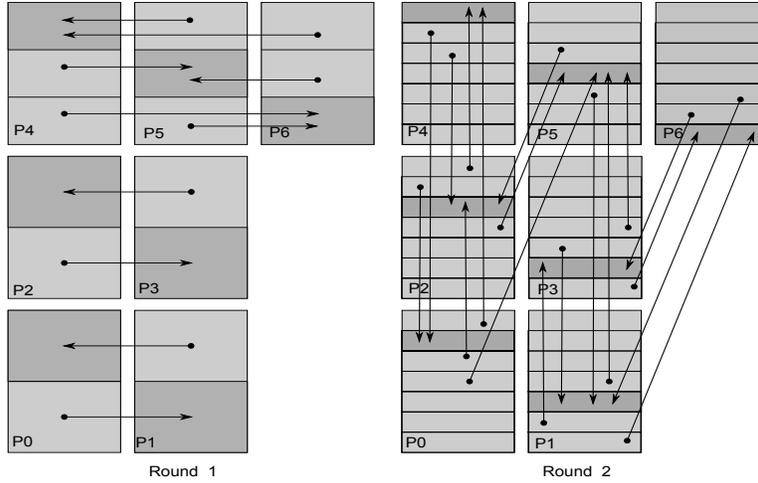


FIGURE 5.5: Example of the 2-3 swap algorithm for 7 processes in 2 rounds. Group size is 2 or 3 in the first round, and between 2 and 5 in the second round.

5.3.2 Radix-k

The next logical step in combining and generalizing direct-send and binary-swap is to allow more combinations of rounds and group sizes. To see how this is done, we begin by adopting the following terminology. Let k_i represent the number of processes in a communication group in round i . The k -values for all rounds can be written as the vector $\vec{k} = [k_1, k_2, \dots, k_r]$, where the number of rounds is denoted by r . Within each group, a direct-send is performed. The total number of processes is p .

More than a convenient notation, this terminology makes clear the relationship among the previous algorithms. We can now define direct-send as $r = 1$ and $\vec{k} = [p]$; and binary-swap as $r = \log p$ and $\vec{k} = [2, 2, 2, \dots]$. Just as 2-3 swap was an incremental step in combining direct-send and binary-swap by allowing k -values that are either 2 or 3, it is natural to ask whether other combinations of r and \vec{k} are possible. The radix- k algorithm [22] answers this question by allowing any factorization of p into $\prod_{i=1}^r k_i = p$. In radix- k , all groups in round i are the same size, k_i .

Figure 5.6 shows an example of radix- k for $p = 12$ and $\vec{k} = [4, 3]$. The processes are drawn in a 4×3 rectangular layout to identify the rounds and groups clearly. In this example, the rows on the left side form groups in the first round, while the columns on the right side form second-round groups. A convenient way to think about forming groups in each round is to envision the process space as an r -dimensional virtual lattice, where the size in each dimension is the k -value for that round. This is the convention followed in Figure 5.6 for two rounds drawn in two dimensions.

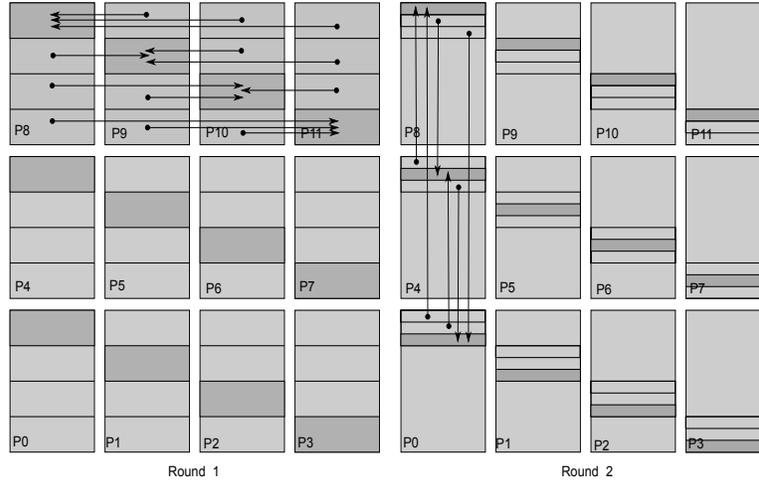


FIGURE 5.6: Example of the radix- k algorithm for 12 processes, factored into 2 rounds of $\vec{k} = [4, 3]$.

The outermost rectangles in the figure represent the image held by each process at the start of the algorithm. During the round i , the current image piece is further divided into k_i pieces, such that the image pieces grow smaller with each round. The image pieces are shown as highlighted boxes in each round.

Selecting different parameters can lead to many options; in the example above we could have chosen $\vec{k} = [12]$, $[6, 2]$, $[2, 6]$, $[3, 4]$, or $[2, 2, 3]$, to name a few. With judicious selection of \vec{k} , we can attain higher compositing rates when the underlying hardware offers support for multiple communication links and the ability to perform communication and computation simultaneously. Even when hardware support for increased parallelism is not available or the image size or number of processes dictates that binary-swap or direct-send is the best approach, those algorithms are valid radix- k configurations.

5.3.3 Optimizations

Kendall et al. [11] extended radix- k to include active pixel encoding and compression, and they showed that such optimizations benefit radix- k more than its predecessors, because the choice of k -values is configurable. Hence, when message size is decreased by active pixel identification and encoding, k -values can be increased and performance can be further enhanced. Their implementation encodes nonempty pixel regions, based on bounding box information, into two separate buffers: one for alternating counts of empty and nonempty pixels and the other for the actual pixel values. This way, new subsets of the image can be taken by reassigning pointers rather than copying

pixels, and images remain encoded throughout all of the compositing rounds. Nonoverlapping regions are copied from one round to the next without performing the blending operation.

A set of empirical tests was performed to determine a table of target k -values for different image sizes, number of processes, and architectures at both the Argonne and Oak Ridge Leadership Computing Facilities. Platforms tested included IBM Blue Gene/P, Cray XT, and two graphics clusters. With this table, radix- k can look up the closest entry for a given image size, system size, and architecture, and automatically factor the number of processes into k -values as close to the target value as possible.

Moreland et al. [18] deployed and evaluated optimizations in a production framework, rather than in isolated tests: IceT serves as both this test and production framework. The advantages of this approach are that tests represent real workloads, and improvements are ready for production use sooner. These improvements include minimizing pixel copying through compositing order and scan-line interleaving, and a new telescoping algorithm for nonpower-of-two number of processes that can further improve radix- k . A final advantage of using IceT for these improvements is that IceT provides unified and reproducible benchmarks that other researchers can repeat.

One of those improvements was devising a compositing order that minimizes pixel copying. The usual, accumulative order causes nonoverlapping pixels to be copied up to $k_i - 1$ times in round i , whereas tree methods only incur $\log k_i$ copy operations. Pixel copying can further be reduced while using a novel image interlacing algorithm. Rather than interleaving partitions according to scan-lines, as in [29], Moreland et al. realized that a desired property of the interleaving is that processes retain contiguous pixel regions after compositing, so as to not require further rearranging. The van der Corput sequence is one example of such an ordering.

While radix- k natively handles nonpower-of-two number of processes, it does not always do so gracefully. Some process counts, especially those with factorizations containing large prime numbers, can exhibit pathological performance. Moreland et al. [18] also devised a telescoping algorithm for nonpowers-of-two that continually looks for largest subgroups that are powers of two, and performs radix- k within each subgroup. Subgroups are further composited together afterwards.

5.4 Results

The theoretical communication and computation costs of direct-send, binary-swap, and radix- k are compared in Table 5.1 using the cost model in Chan et al. [7]. This model assumes p processors in a distributed-memory parallel architecture, and the original image size has n total pixels. The com-

TABLE 5.1: Theoretical Lower Bounds for Compositing Algorithms

Algorithm	Latency	Bandwidth	Computation
direct-send	$\alpha p/k$	$n\beta(p-1)/p$	$n\gamma(p-1)/p$
binary-swap	$\alpha \log_2 p$	$n\beta(p-1)/p$	$n\gamma(p-1)/p$
radix-k	$\alpha \log_k p$	$n\beta(p-1)/p$	$n\gamma(p-1)/p$

munication cost is $\alpha + n\beta$, where α is the latency per message and β is the transmission time per data item (reciprocal of the link bandwidth). The computation time to compose one pixel is γ , making the total time to transmit and reduce a message consisting of n data elements $\alpha + n\beta + n\gamma$. The model further assumes a fully connected network, where k messages can occupy the network without link contention and no overlap between communication and computation.

Some of those assumptions are not true in practice, but calculating the relative cost of communication algorithms is simplified under these conditions. In actual implementations, radix-k group sizes vary between 8 and 128, depending on the architecture and optimizations, and there is overlap between communication and computation on modern HPC systems that radix-k uses to further boost performance.

Nonetheless, Table 5.1 shows that in theory, radix-k should perform as well or better than other algorithms, a theory confirmed in Figure 5.7, which shows a test of the original algorithm in Peterka et al. [22] for a variety of process

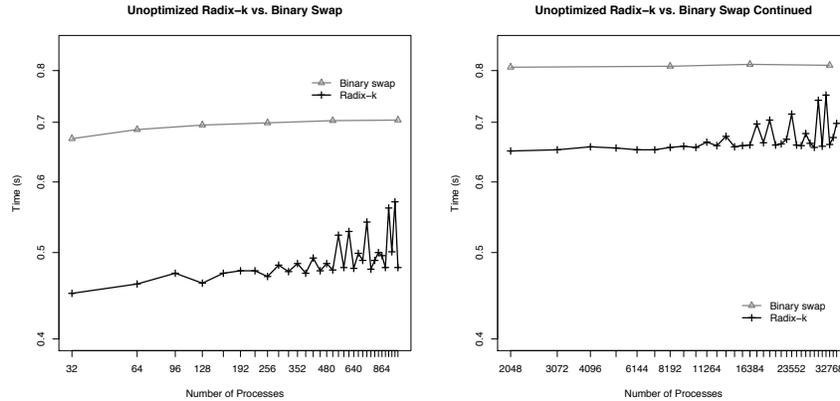


FIGURE 5.7: Performance comparing binary-swap with radix-k for an image size of 8 megapixels. Left: process counts from 32 to 1,024 in increments of 32. Right: the same test continued at larger scale in increments of 1,024 processes. No optimizations, such as compression or active pixel encoding, were applied to either algorithm for this test.

Table of target k-values for Intrepid BG/P and Jaguar XT5

Processes	Megapixels							
	Intrepid				Jaguar			
	4	8	16	32	4	8	16	32
8	8	8	8	8	4	4	4	4
16	16	16	16	16	4	4	4	4
32	32	32	32	32	16	8	16	16
64	64	64	64	64	16	16	16	16
128	64	128	128	128	8	8	8	8
256	64	128	128	128	16	8	8	8
512	64	128	128	128	16	32	8	8
1 K	64	128	128	128	64	32	32	8
2 K	32	128	128	128	8	32	32	32
4 K	32	32	32	32	8	16	32	64
8 K	32	32	32	32	4	8	32	64
16 K	32	32	32	32	4	32	32	8
32 K	32	32	32	32	16	16	64	8

FIGURE 5.8: Target k-values for two different machines are shown. Optimizations such as active pixel encoding enable the use of higher k-values than before. In the original algorithm, $k = 8$ was used, but the tables above show that with active pixel encoding, k-values as high as 128 are optimal, depending on the machine.

counts from 32 to 34,816 on the *Intrepid* Blue Gene/P supercomputer at Argonne National Laboratory. These graphs compare binary-swap and radix-k using k-values of 8 whenever possible, and include no other optimizations such as compression. In the left graph, process count increases by 32 at each

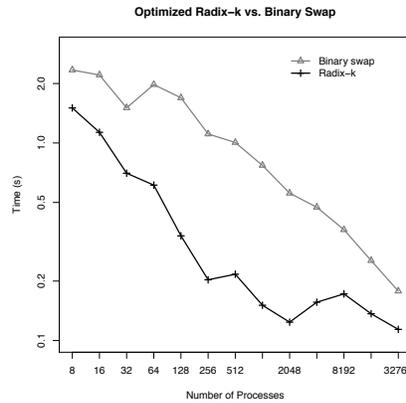


FIGURE 5.9: Performance comparing optimized binary-swap with radix-k shows overall improvement in volume rendering tests of core-collapse supernovae.

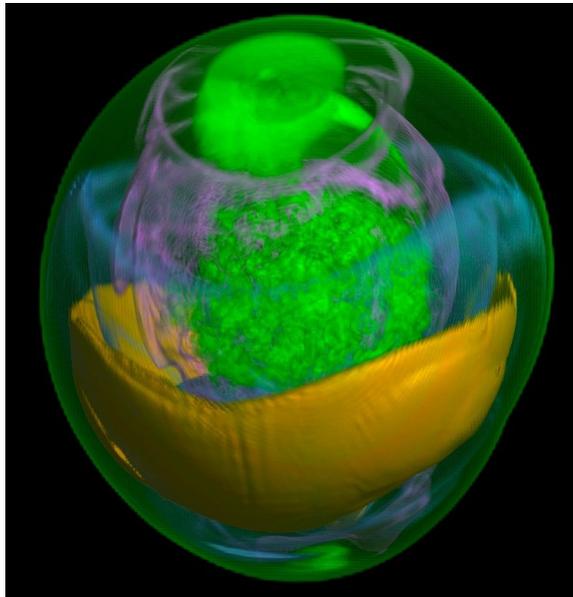


FIGURE 5.10: Core-collapse supernova volume rendered in parallel and composited using the radix-k algorithm.

data point up to 1024 processes, while the right graph is a continuation of the test by adding one Blue Gene rack (1,024 nodes) at each data point. On average, radix-k is approximately 20% to 40% faster than binary-swap.

The speedup increases when optimizations such as active pixel encoding are enabled. While the target k-value for Intrepid was 8 in the original algorithm of [22], Figure 5.8 shows that with optimizations, target k-values can be as high as 128. The reason is that smaller message sizes produced by active pixel encoding and compression allow more messages to be injected into the system without contention. The resulting performance is shown in the strong scaling study of Figure 5.9. With the optimizations of Kendall et al. [11], performance is up to five times better than binary-swap with the same optimizations. These tests were conducted while parallel volume rendering the core-collapse supernova shown in Figure 5.10. Scientists visualize such simulations in order to understand the physics of thermonuclear and hydrodynamic instabilities in the death of some of the most massive stars in the universe.

5.5 Discussion and Summary

5.5.1 Conclusions

The radix-k algorithm allows the number of message partners per round and the number of rounds to be adjusted to maximize performance for a given problem size and architecture. Algorithms such as radix-k and 2-3 swap build on the previous contributions of binary-swap and direct-send, generalizing and unifying two algorithms that previously were considered separately. Now, we understand that binary-swap and direct-send are just two points in an entire parameter space of configurations.

Optimizations to radix-k such as active pixel encoding enabled the use of higher k-values, up to 128 in practice, and overall performance can be improved over binary-swap by up to five times for identical optimizations. Such improvements enable strong scaling at the full scale of HPC machines such as IBM Blue Gene and Cray XT, compared to legacy algorithms whose performance bottomed-out at a few thousand processes.

It is now possible to compose large-size images interactively as well; sub-second compositing of images up to 64 megapixels has been demonstrated on actual volume rendering of core-collapse supernova datasets. Finally, all of these improvements are now available in IceT for production use in tools such as ParaView and VisIt. For example, the latest version of IceT in ParaView enabled up to seven times faster image compositing and the ability to scale to 36,000 processes, which was not possible with previous versions of the library.

Of course, these improvements are predicated on an underlying architecture that can support additional message concurrency. Today, HPC networks have multiple data paths and DMA access and thus can benefit from highly parallel algorithms such as radix-k. Because it is reasonable to expect more scientific visualizations to execute on supercomputers in the future, making efficient use of these architectures for compositing will be a critical part of the high-performance visualization pipeline.

Modern compositing algorithms have demonstrated a tangible impact for high-performance visualization and computational science. They allow scientists to render higher resolution images, at larger system scales, faster than before. Together, all of these benefits can result in increased understanding of scientific data, and they will be absolutely essential going forward toward exascale computing.

5.5.2 Directions for Future Research

Highly parallel image compositing offers several avenues for continued exploration. Load balance in both computation and communication can be further improved and combined with fine-grained delineation of active pixels. Other potential solutions to load balancing may be found in time-varying parallel rendering, where the additional time dimension poses new scheduling opportunities.

New architectures are creating the need for continued study. Network

topologies such as higher-dimensional torus networks are being investigated, as are clouds and distributed collections of heterogeneous processing elements [14]. With the ubiquity of multicore processors, a natural next step is to parallelize the computing of the compositing operator across several pixels. Hybrid parallelism that combines message passing with shared-memory threading is being studied in numerous algorithms; image compositing can also potentially benefit from the hybrid programming models in Chapter 12.

Algorithms such as radix-k require empirical experiments to find appropriate k-values for a particular architecture, but there exists a trend in communication algorithms toward self-tuning. Hence, another area for further study is automating parameter selection.

Acknowledgment

We gratefully acknowledge the use of the resources of the Argonne and Oak Ridge Leadership Computing Facilities at Argonne and Oak Ridge National Laboratories. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. Work is also supported by DOE with agreement No. DE-FC02-06ER25777.

Bibliography

- [1] James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. In *Proc. Eurographics Parallel Graphics and Visualization Symposium 1998*, Bristol, United Kingdom, 1998.
- [2] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, Robert Geijn, and Jerrell Watts. Interprocessor collective communication library (intercom). In *In Proceedings of the Scalable High Performance Computing Conference*, pages 357–364. IEEE Computer Society Press, 1994.
- [3] Mike Barnett, David G. Payne, Robert A. van de Geijn, and Jerrell Watts. Broadcasting on meshes with wormhole routing. *Journal of Parallel Distributed Computing*, 35(2):111–122, 1996.
- [4] Massimo Bernaschi and Giulio Iannello. Collective communication operations: Experimental results vs. theory. *Concurrency*, 10(5):359–386, 1998.
- [5] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multi-port message-passing systems. In *SPAA '94: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 298–309, New York, NY, USA, 1994. ACM.
- [6] Xavier Cavin, Christophe Mion, and Alain Fibois. Cots cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proceedings of IEEE Visualization 2005*, pages 111–118, 2005.
- [7] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(13):1749–1783, 2007.
- [8] Ernie Chan, Robert van de Geijn, William Gropp, and Rajeev Thakur. Collective communication on architectures that support simultaneous communication over multiple links. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 2–11, New York, NY, USA, 2006. ACM.
- [9] H. R. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. C. Miller, B. J. Whitlock, and N. L. Max. A contract based system for large data visualization. In *Proceedings of IEEE Visualization 2005*, pages 190–198, Minneapolis, MN, 2005.

- [10] William M. Hsu. Segmented ray casting for data parallel volume rendering. In *Proceedings of 1993 Parallel Rendering Symposium*, pages 7–14, San Jose, CA, 1993.
- [11] Wesley Kendall, Tom Peterka, Jian Huang, Han-Wei Shen, and Robert Ross. Accelerating and benchmarking radix-k image compositing at large scale. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization EG PGV'10*, Norrkoping, Sweden, 2010.
- [12] Sameer Kumar, Gabor Dozsa, Gheorghe Almasi, Philip Heidelberger, Dong Chen, Mark E. Giampapa, Michael Blocksome, Ahmad Faraj, Jeff Parker, Joseph Ratterman, Brian Smith, and Charles J. Archer. The deep computing messaging framework: generalized scalable message passing on the blue gene/p supercomputer. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 94–103, New York, NY, USA, 2008. ACM.
- [13] Sameer Kumar, Gabor Dozsa, Jeremy Berg, Bob Cernohous, Douglas Miller, Joseph Ratterman, Brian Smith, and Philip Heidelberger. Architecture of the component collective messaging interface. In *Euro PVM/MPI '08: Proceedings of the 15th annual European PVM/MPI users' group meeting*, pages 23–32, New York, NY, USA, 2008. Springer.
- [14] Kwan-Liu Ma and David M. Camp. High performance visualization of time-varying volume data over a wide-area network. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '00, Washington, DC, USA, 2000. IEEE Computer Society.
- [15] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(4):59–68, 1994.
- [16] Laurent Moll, Mark Shand, and Alan Heirich. Sepia: Scalable 3d compositing using pci pamette. In *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '99*, pages 146–, Washington, DC, USA, 1999. IEEE Computer Society.
- [17] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.
- [18] Ken Moreland, Wesley Kendall, Tom Peterka, and Jian Huang. An image compositing solution at scale. In *Proceedings of SC11*, Seattle, WA, 2011.
- [19] Kenneth Moreland, Brian Wylie, and Constantine Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and*

large-data visualization and graphics, pages 85–92, Piscataway, NJ, USA, 2001. IEEE Press.

- [20] Shigeru Muraki, Masato Ogata, Kwan-Liu Ma, Kenji Koshizuka, Kagenori Kajihara, Xuezheng Liu, Yasutada Nagano, and Kazuro Shimokawa. Next-generation visual supercomputing using pc clusters with volume graphics hardware devices. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 51–51, New York, NY, USA, 2001. ACM.
- [21] Ulrich Neumann. Communication costs for parallel volume-rendering algorithms. *IEEE Computer Graphics and Applications*, 14(4):49–58, 1994.
- [22] Tom Peterka, David Goodell, Robert Ross, Han-Wei Shen, and Rajeev Thakur. A configurable algorithm for parallel image-compositing applications. In *Proceedings of SC 09*, Portland OR, 2009.
- [23] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 253–259, 1984.
- [24] David Pugmire, Laura Monroe, Andrew DuBois, and David DuBois. Npu-based image compositing in a distributed visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):798–809, 2007. Member-Connor Davenport, Carolyn and Member-Poole, Stephen.
- [25] Rolf Rabenseifner. *New Optimized MPI Reduce Algorithm*. 2004. <http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html>.
- [26] C. R. Ramakrishnan and Claudio Silva. Optimal processor allocation for sort-last compositing under bsp-tree ordering. 1999.
- [27] Gordon Stoll, Matthew Eldridge, Dan Patterson, Art Webb, Steven Berman, Richard Levy, Chris Caywood, Milton Taveira, Stephen Hunt, and Pat Hanrahan. Lightning-2: a high-performance display subsystem for pc clusters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 141–148, New York, NY, USA, 2001. ACM.
- [28] Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett. Slic: Scheduled linear image compositing for parallel volume rendering. In *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 33–40, Seattle, WA, 2003.
- [29] Akira Takeuchi, Fumihiko Ino, and Kenichi Hagihara. An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. *Parallel Comput.*, 29(11-12):1745–1762, 2003.

- [30] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.
- [31] Jesper Larsson Traff, Andreas Ripke, Christian Siebert, Pavan Balaji, Rajeev Thakur, and William Gropp. A simple, pipelined algorithm for large, irregular all-gather problems. In *Proceedings of EuroPVM/MPI 2008*, Dublin, Ireland, 2008.
- [32] Brian Wylie, Constantine Pavlakos, Vasily Lewis, and Kenneth Moreland. Scalable rendering on pc clusters. *IEEE Computer Graphics and Applications*, 21(4):62–69, 2001.
- [33] Hongfeng Yu, Chaoli Wang, and Kwan-Liu Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

This next paragraph is required by ANL for submission but can be removed for final publication

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.