A Configurable Algorithm for Parallel Image-Compositing Applications

Tom Peterka
Argonne National Laboratory
tpeterka@mcs.anl.gov

David Goodell
Argonne National Laboratory
goodell@mcs.anl.gov

Robert Ross
Argonne National Laboratory
rross@mcs.anl.gov

Han-Wei Shen
The Ohio State University
hwshen@cse.ohio-state.edu

Rajeev Thakur
Argonne National Laboratory
thakur@mcs.anl.gov

ABSTRACT

Collective communication operations can dominate the cost of large-scale parallel algorithms. Image compositing in parallel scientific visualization is a reduction operation where this is the case. We present a new algorithm called *Radix-k* that in many cases performs better than existing compositing algorithms. It does so through a set of configurable parameters, the radices, that determine the number of communication partners in each message round. The algorithm embodies and unifies binary swap and direct-send, two of the best-known compositing methods, and enables numerous other configurations through appropriate choices of radices. While the algorithm is not tied to a particular computing architecture or network topology, the selection of radices allows Radix-k to take advantage of new supercomputer interconnect features such as multiporting. We show scalability across image size and system size, including both powers of two and nonpowers-of-two process counts.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Image compositing; Communication; Parallel scientific visualization

1. INTRODUCTION

Image compositing is the last stage in sort-last parallel visualization algorithms. In these applications, the dataset is partitioned into subdomains, and each process performs the visualization independently on its region; the compositing step blends these images into a final result. As image sizes and processor counts increase, the time to composite can dominate the cost of the entire visualization process. Just as scientific computations utilize parallelism to achieve new scales, parallelism is an important ingredient for analysis applications such as scientific visualization at large scale. An essential ingredient in many parallel problems is the merging of partial results; in parallel visualization, the results are images, and this merging step is image compositing.

November 14-20, 2009, Portland, Oregon, USA

1.1 An Improved Compositing Algorithm

Image compositing has been studied for over twenty years, but the best-known algorithms were invented when the computing landscape looked quite different from today. In developing a new algorithm, we strived to keep the best features of existing algorithm because, as we found, they perform their functions efficiently. At the same time, new hardware features such as multiported networks and network communication hardware that does not require CPU intervention present new opportunities to improve the state of the art in image compositing.

To this end, we developed a new algorithm called *Radix-k*. By configuring its parameters, we can enable the algorithm to encompass and unify binary swap and direct-send, well-known methods previously considered distinct. It offers many more combinations than those, however; and by judicious selection of its parameters (the radices or k-values), we can attain higher compositing rates than before, in particular when the underlying hardware offers support for multiple communication links and the ability to perform communication and computation simultaneously. Furthermore, Radix-k is not limited to processor counts that are powers of two; it exhibits good behavior over a variety of process counts.

The key to improved performance is to utilize available resources, both computational and networking. Like earlier algorithms, Radix-k keeps processors busy and load balanced in each message round. By using higher radices, however, it increases concurrency by exchanging messages in parallel and by computing in parallel with messaging. When hardware in support of increased parallelism is not available and the image size or number of processes dictates that binary swap or direct-send is the best approach, Radix-k accommodates those techniques as well.

1.2 Problem Definition

The image compositing task is a reduction problem. Each process communicates its results with all others, reducing

results along the way. It is not necessary for this result to end up at one process; a distributed answer is acceptable and often desirable. For example, writing the image to storage can be done more efficiently in parallel than from a single source. In MPI terminology, this is equivalent to a reduce-scatter problem. Because the particular composite operator depends on ordering, this reduce-scatter is noncommutative.

The problem can be formally defined by the following four postulates. (1) Each of p processes owns one vector x_p, its local image, where x has length n components, and each component is a pixel. (2) Let *over* [23] be a binary, component wise linear combination of two vectors. *Over* is associative but not commutative, and a correct ordering of operations is established by the visualization algorithm. In our tests, we impose process rank order to be the correct ordering of images. (3) All x_i are blended with all other x_j using the over operator. Because of the associative property, however, this does not require all-to-all communication. For example, if A, B, C are processes and $\oplus$ is the over operator, $A \oplus B$ suffices if B already contains the result of $B \oplus C$. A does not need to composite with C explicitly. (4) The resulting vector is distributed among one or more processes. Compositing has completed when each pixel has a final, correct value. Each process ends with a contiguous subset of all of the finished pixels.

The length of our vectors corresponds to image sizes encountered in high-quality scientific visualization. They range from a 1-megapixel image (for example, 1024 x 1024 pixels) to an 8-megapixel image (for example, 4096 x 2028 pixels). In terms of display devices, 2 megapixels is approximately high-definition television (HDTV) resolution; a 30-inch monitor has 4 megapixels, and a 4K projector used in digital cinema has 8 megapixels. We maintain a 4-byte floating-point value per channel, or 16 bytes per pixel, during the compositing process so that quantization errors do not accrue. Therefore, our starting images are 16 MB for 1 megapixel to 128 MB for 8 megapixels. (If only 4 bytes per pixel were used, our 128 MB image would contain 32 megapixels.) The image owned by a process may be sparse. Image compositing algorithms can be optimized to take advantage of this property, but we do not include these optimizations in our tests. Instead, we consider the worst-case scenario where all pixels are transmitted.

2. BACKGROUND

Relevant literature on this problem comes from the scientific visualization and from the high-performance computing communities. We survey each of these sources and conclude this section by evaluating the theoretical cost of various algorithms cited.

2.1 Image Compositing and Collective Communication Algorithms

Parallel rendering can be classified according to when rasterized images are sorted [18]; our work applies to sort-last rendering only. The dataset is partitioned among processors at the beginning of the process; rendering occurs locally at each processor, and the resulting images are depth-sorted (composited) at the end. Stompel et al. [27] survey methods for sort-last compositing, and Cavin et al. [7] analyze 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.

In direct-send, each process requests the subimages from all of those processes that have something to contribute to it [11, 21, 16]. Rather than sending individual point-to-point messages, tree methods exchange data between pairs of processes, building more complete subimages at each level of the compositing tree. To improve load balance by keeping more processes busy at higher levels on the tree, Ma et al. [17] introduced binary swap, a distance doubling and vector halving algorithm. Recently, Yu et al. [32] extended binary swap compositing to nonpower-of-two numbers of processors in 2-3 swap compositing. Pipeline methods are also published for image compositing, but their use is infrequent. Lee et al. [15] discuss a parallel pipeline compositing algorithm for polygon rendering.

Hybrid combinations of the above also have been studied. Nonaka et al. [22] combine binary swap with direct send and binary tree in two stages to improve performance. Nodes are partitioned into several groups and binary swap is executed in each of those groups. The results from each of the partitions are then combined by using either direct-send or a simple binary tree.

The above methods can be optimized by exploiting the spatial locality and sparseness in images resulting from scientific visualization. Run-length encoding images before transmitting among processes achieves lossless compression [2]. Using bounding boxes to identify the nonzero pixels is another way to reduce image size [17]. These optimizations can minimize both communication and computation costs. Takeuchi et al. [28] accelerate binary swap with bounding rectangles, interleaved splitting, and run-length encoding to mitigate any remaining sparseness. The Radix-k algorithm can likewise benefit from these optimizations, although in this paper, all of our tests are based on worst-case, full-size images.

Image compositing has also been combined with parallel rendering for tiled displays. The IceT library, based on Moreland et al. [19] is one example of a hybrid algorithm that 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. By limiting the composited image size to that of a physical display tile, the problem of compositing large images (tens of megapixels) arising from tiled display walls is alleviated. Humphreys et al.'s Chromium [12] is another system that supports compositing across cluster nodes and tiled displays. While its default compositor for sort-last rendering is a single node, the authors demonstrate that a binary-swap stream processing unit (SPU) can be built using Chromium. Radix-k is a general message-passing algorithm that in principle can serve as the compositing module in other libraries such as IceT, although we have not tested this. Our tests simulate volume rendered images but Radix-k can support polygon rendering (as in IceT and Chromium) by including a depth value per pixel and modifying the compositing operator. The only prerequisite for including Radix-k in other rendering libraries is the existence of MPI.

One may group collective communication algorithms into tree-order, linear or ring-order, and dimension-order. Tree-order algorithms are optimal for short messages where the message latency dominates total time, with $\log_2(p)$ number of steps, where p is the number of processes. Messages are sent between nodes of a minimum spanning tree, and the distance between nodes doubles while message length is divided by two. Bernaschi and Ianello [5] expand the idea to make it more flexible by controlling the depth and width of the spanning tree with an α parameter that changes the spanning tree from a single-level flat tree to a binomial tree ($\alpha = 0, \alpha = .5$, respectively).

Linear and ring-order algorithms perform better for long messages, especially when pipelined so that steps overlap. Barnett et al. [4] present a pipelined algorithm, and Traff et al. [31] show that dividing nonuniform message lengths into uniform size blocks and pipelining those blocks produces significant speedup in a linear ring all-gather. Dimension-order algorithms perform operations dimension by dimension, where the later dimensions use the results of the earlier. Barnett et al. [3] use a general-purpose scatter-collect template to implement a variety of collective primitives on a mesh, including broadcast, scatter, gather, collect, and reduce.

Thakur et al. [29] show that MPICH collectives can be optimized when message length is considered, so that latency is minimized for short messages and bandwidth is minimized for long messages. They perform tests in the context of switched clusters and concluded that for reduce operations, binomial tree (MST) is ideal for short messages < 2 KB and for user-defined reduce operations that may be difficult to divided into scatter-collect. For larger library operations, they recommend Rabenseifner's [25] dimensional order algorithm, similar to that of Barnett et al. [4].

Newer interconnects are multiported; based on this idea, Chan et al. [10] rewrote the tree and ring algorithms to communicate with multiple neighbors at each stage. Bruck [6] also studies multiporting in the context of all-to-all collectives. In recent work, Kumar et al. [13, 14] incorporate a number of these optimizations in collective interfaces that exist at different layers of the messaging stack. They show that performance saturates at different number of links, depending on the message protocol used.

Another new advance is the availability of programmable network processing units (NPUs). Pugmire et al. [24] have shown that image compositing can be accelerated using NPUs in fixed, tree configurations of up to 512 rendering nodes. Presently, the NPU is a standalone device, but programmable network adapters may eventually find their way into general-purpose compute nodes, enabling message communication patterns such as those in Radix-k to be accelerated in hardware.

2.2 Communication and Computational Cost

To model the communication and computation cost of various algorithms, we adopt a simple cost model as in Chan et al. [9]. The assumptions in this model are the following. (1) There are p processors indexed from 0 to $p-1$ in a distributed-memory parallel architecture. (2) There are n data items in the original vector size. In our application, a data item is one pixel, and the original image size has n total pixels. A pixel occupies 16 bytes. (3) The communication cost is $\alpha + n\beta$, where α is the latency per message and β is the transmission time per data item (reciprocal of single link bandwidth). (4) The computation time to reduce one data element is γ, making the total time to transmit and reduce a message consisting of n data elements $\alpha + n\beta + n\gamma$. (5) For cost calculations, we assume a fully connected network, nonoverlapping communication and computation, and zero link contention.

Table 1: Lower Bounds for Commutative Reduction Collectives

Collective	Latency	Bandwidth	Computation
Reduce	$\alpha \log_2(p)$	$n\beta$	$n\gamma(p-1)/p$
Allreduce	$\alpha \log_2(p)$	$2n\beta(p-1)/p$	$n\gamma(p-1)/p$
Reduce-scatter	$\alpha \log_2(p)$	$n\beta(p-1)/p$	$n\gamma(p-1)/p$

Parts of the last assumption are not always true, but calculating the relative cost of communication algorithms is simplified under these conditions. After computing the theoretical cost, we compare the actual cost of our algorithm with the predicted cost. Three MPI collective reduction operations, along with binary swap, form our baseline analysis: reduce, allreduce, and reduce-scatter. For our Blue Gene/P MPI implementation, Figure 1 shows how these compare.

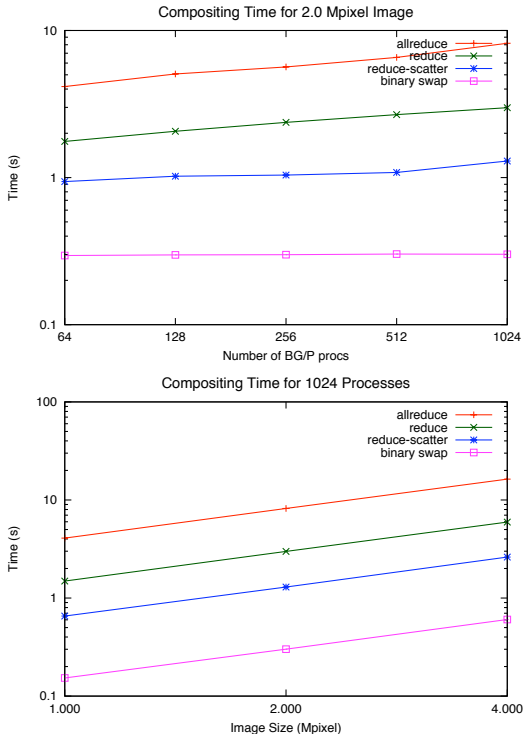

Figure 1: Top: Comparison of binary swap with built-in MPI collectives for image compositing. Binary swap performs faster and more consistently across number of processes. Bottom: All of the algorithms scale similarly with image size.

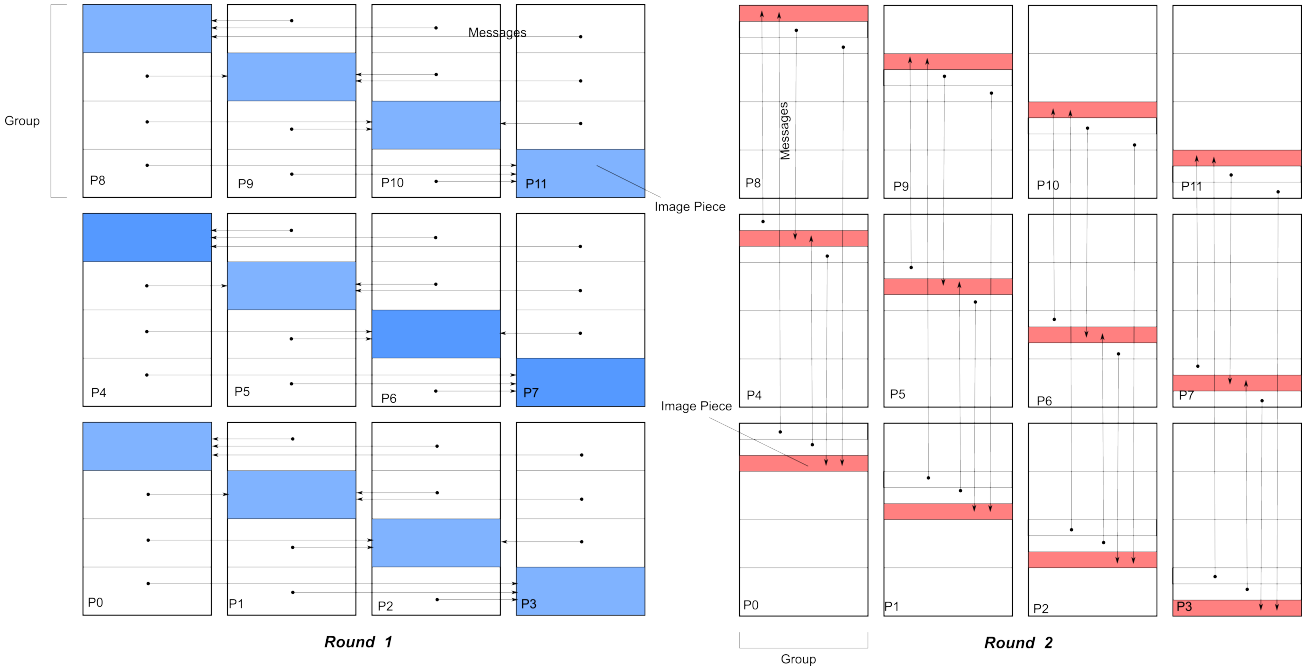

Figure 2: Example of the Radix-k algorithm for 12 processes, factored into 2 rounds of k = [4, 3].

Binary swap performs consistently across a wide range of processor counts and is approximately two times faster than its closest competitor, reduce-scatter. All of these algorithms scale similarly with image size.

In the following, we review the theoretical lower bounds for reduce, allreduce, and reduce-scatter. Then we compare these bounds to binary swap and direct-send. Chan et al. [9] define lower bounds on the latency, bandwidth, and computation terms (see Table 1) for commutative and associative operations. The latency term counts the number of *communication steps*, not the total number of messages; within each step p messages are exchanged simultaneously. The bandwidth and computation terms count the total number of data elements exchanged and computed, respectively.

In binary swap, the number of communication rounds is $\log_2(p)$, with one message sent and one message received by each process in each round. Since we assume at least one communication link in each direction, the latency component achieves the same lower bound as in Table 1. Ma et al. [17] compute the total number of pixels reduced; if inactive pixels are ignored, their result simplifies to $\sum_{i=1}^{\log_2 p} 2^{-i} n = n(p-1)/p$, matching the values in Table 1 for the bandwidth and computation terms.

This is the same lower bound reported by Neumann [20] for direct-send, again ignoring the active pixel optimization and assuming perfect load balancing. However, the average number of message steps in direct send is $p^{1/3}$, assuming an orthogonal view direction along one of the principal axes and all active pixels. Thus, at higher process numbers, direct-send leads to a higher latency for short messages and potential network congestion.

Binary swap incurs an added expense when the number of processors is not a power of two. Rabenseifner and Traff [26, 30] present a 3-2 elimination step in a hybrid butterfly algorithm. For an arbitrary number of processes, the latency term is at most $2\log_2 p+1$, while the bandwidth term is less than $3m$, and the compute term is less than $3/2m$. Yu et al. [32] also present an extension to binary swap for nonpower-of-two number of processes. Compared to Table 1, their 2-3 swap is four times greater in the latency term, 1.3 times greater in the bandwidth term, and two times greater in the computation term.

3. METHOD

In this section, we describe the operation of the Radix-k algorithm. We begin with a high-level description of message rounds, what processes communicate in each round, and what portion of the image is exchanged. Then, we compute the theoretical cost using the previous cost model and compare it to the optimal cost of the algorithms in the previous section. To show features of the algorithm that the cost model does not account for, such as overlapping operations, we log and profile message transfers and compositing calculations. By tracking communication and computation and operations and by viewing them with an information visualization tool, we can better understand the subtleties of the algorithm.

3.1 Algorithm Description

In this discussion, we refer periodically to Figure 2, which shows an example of 12 processes. To begin, the total number of processes, p, is factored into r factors. These factors constitute the vector $\mathbf{k} = [k_1, k_2, ...k_r]$ such that $p = \prod_{i=1}^{r} k_i$. There are r communication/compositing rounds. In each round i, there are p/k_i groups, and each group has k_i participants. Within a round, only the participants in

a group communicate with each other. In Figure 2, the 12 processes are factored into two rounds, with $\mathbf{k} = [4, 3]$. The processes are drawn in a 4×3 rectangular layout to identify the rounds more clearly. In this example, the rows of the grid form groups in the first round, and the columns form second-round groups. (Recall that the algorithm makes no assumptions about actual topology.) The outermost rectangles represent the image held by each process at the start of the algorithm.

During the current round i, the current image piece is further divided into k_i pieces. Each of the k_i group participants is responsible for compositing one of these pieces. The other members in the group send the appropriate piece to this member for compositing. For example, the first group member receives the first image piece from the other $k_i - 1$ members. In the next round $i + 1$, the image pieces are further subdivided by the new k_{i+1} number of group members, such that the image pieces grow smaller with each round. The original image size does not need to be a multiple of the number of processes, nor do current round image pieces need to be evenly divisible by the current number of group participants. In Figure 2, first-round messages travel horizontally, and second-round messages travel vertically. The image pieces are shown as colored boxes in each round.

As a group member receives image pieces from the other participants, it composites them with its own, one at a time, accumulating the current result. These messages may arrive out of order, in this case they are buffered until ready to be composited in the correct order. A message may be composited immediately if it is either directly over or directly under the accumulated result, or directly over or directly under another message that has already arrived. By *directly*, we mean that no other messages are between these two in the sequence.

At a new round, groups are formed from participants that are (in rank order) farther apart from each other than they were in the previous round. In the first round, the k members in a group are nearest neighbors in rank order; in the second round, each member is k apart; in round i, the rank-order distance between members is $\prod_{j=1}^{i-1} k_j$. 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. Imagine the processes are mapped onto that lattice in row-major order, extended to r dimensions. In the first round, groups are formed by taking rows in the lattice; the second-round groups are columns; and so on. This is the convention followed in Figure 2.

3.2 Computing Theoretical Cost

Using the same cost model as earlier, one can compute the cost of the Radix-k algorithm in terms of latency, bandwidth, and computation terms. The latency term is simply αr, because there are r rounds. Compared to binary swap and the latency terms in Table 1, this is less than $\alpha \log_2(p)$ provided that there exists some $k_i > 2$.

The bandwidth term is slightly more complex. There are r rounds, each consisting of one less message than k_i of the current round. Hence, the number of messages communicated by a process is $\sum_{i=1}^{r} k_i - 1$. The size of the message in round i is divided by a factor of the current k_i in each round, so the current message size at round i is $\prod_{j=1}^{i} 1/k_j$. Combining these expressions, we arrive at the bandwidth term: $n\beta \sum_{i=1}^{r} [(k_i - 1) \prod_{j=1}^{i} 1/k_j]$. The computation term is the same except that β is replaced with γ; every pixel that is transmitted also needs to be composited.

This expression can be simplified so that it can be compared with the optimal lower bound, $(p - 1)/p$. Converting the terms in the above summation to a common denominator yields $\sum_{i=1}^{r} [(k_i - 1) \prod_{j=1}^{i} 1/k_j \prod_{n=i+1}^{r} k_n/k_n]$. The denominator is the product of all of the k-values, or simply p. The numerator is $\sum_{i=1}^{r} [(k_i - 1) \prod_{n=i+1}^{r} k_n]$ or, expanded, $k_1k_2k_3...k_r - k_2k_3k_4...k_r + k_2k_3k_4...k_r + ... - k_{r-1}k_r + k_{r-1}k_r - k_r + k_r - 1$. Canceling all of the inner terms leaves $p - 1$ in the numerator.

Radix-k has the same theoretical communication and computation cost as does binary swap; in essence it combines multiple binary swap rounds into one round by using a higher radix than 2. It performs these higher-radix rounds using direct-send inside each round, which also has the same optimal bandwidth and computation cost. When all of the k-values are 2, the result is binary swap. When there is only a single round and $\mathbf{k} = [p]$, we have direct-send. Between these two extremes are multiple rounds of direct-sends in each group.

3.3 Profiling Actual Cost

The preceding cost computations make some simplifying assumptions, such as a fully connected network, nonoverlapped communication and computation, and the ability to receive multiple messages simultaneously. Depending on the hardware, some of these assumptions may not hold. We turn next to profiling tools to see these effects. By comparing the communication and computation profiles using MPE and Jumpshot [8] we can better understand how Radix-k compares, for example, to binary swap.

Figure 3 show traces of binary swap and Radix-k for $\mathbf{k} = [8, 8]$, using 64 processes on Blue Gene/P. In these figures, the horizontal axis measures time, and the vertical axis denotes process ranks. Only the first 32 processes are shown in the figure; the other 32 processes have similar patterns. The red and blue boxes denote computation of a pair of composited images. In binary swap, the green boxes indicate the time spent in communicating via MPI_Sendrecv. In Radix-k, the salmon-colored boxes indicate the time spent waiting for nonblocking messages to arrive (MPI_Waitany). In both diagrams, white arrows represent message transmission between processes.

The patterns are quite different for the two algorithms. Binary swap is entirely synchronous; the six rounds in this example are easy to see. Each is composed of a communication followed by a computation. The time for each round is one-half of the previous round because message size is cut by one-half in each round. The synchronous nature of binary swap does not permit any overlap between communication and computation.

Radix-k is designed to be asynchronous when the architec-

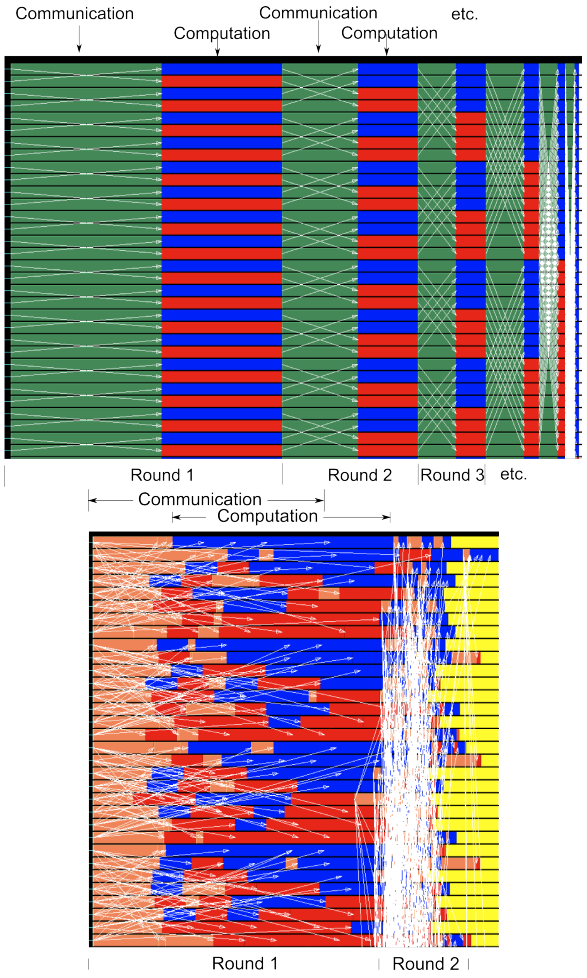

Figure 3: Top: Jumpshot profile of the binary swap algorithm for 64 processes. Bottom: Jumpshot profile of the Radix-k algorithm for 64 processes, factored into 2 rounds of k = [8, 8].

ture supports multiple messages arriving concurrently with computation. Not all architectures support these features, as our second case study below shows. When such hardware support does exist, however, Radix-k can exploit it. In the right side of Figure 3, computation blocks are drawn on top of the communication blocks, and the communication block extends until the last computation block in a round. Computation begins early, as blocks are composited as soon as possible. Rounds also begin asynchronously, as soon as a process has completed the previous round. This approach makes the boundary between rounds less defined in Figure 3.

4. RESULTS

Our tests were conducted on two platforms at Argonne National Laboratory. The IBM Blue Gene/P *Intrepid* is a 557-teraflop supercomputer currently ranked fifth on the Top 500 list. It consists of 40 racks, each rack containing 1,024 nodes, for a total of 40,960 nodes. Each node has four cores, for a grand total of 163,840 cores. The nodes are connected in a 3D torus topology. Our tests are conducted in *smp* mode, that is, one process per node. The second platform is a graphics cluster consisting of 100 compute nodes and 200 Quadro FX5600 graphics processing units. At 111 teraflops, *Eureka* is the world's largest NVIDIA Quadro Plex installation. The compute nodes are connected by a Myrinet switching fabric. Both machines are operated by the Argonne Leadership Computing Facility [1].

All tests were run multiple times to check for variability; mean times are reported. On both Intrepid and Eureka, standard deviation averaged 5 ms over all of our tests. The test images were synthetic checkerboard patterns, where each process contained a slightly different pattern offset from the previous one. Test results were cross-checked for correctness against a serial code that performed the image compositing. Images were composited in process rank order: process 0's image over process 1's, which in turn was over process 2's, and so forth.

In the following graphs, we compare our results with binary swap because, as we noted in the background section, it is the de facto standard in image compositing. We do not apply optimizations such as bounding the active pixels in either algorithm, but, instead, consider the worst case, where all pixels of all images are used. K-values are set manually based on initial tests of good values for a particular architecture.

4.1 Scalability

In order to evaluate scalability over a variety of system scales and problem sizes, Figure 4 shows four image sizes: 1, 2, 4, and 8 megapixels. At 16 bytes per pixel, starting message vectors range from 16 MB to 128 MB. Results are taken at power-of-two numbers of processes from 32 to 16384. In initial tests of what k-values work well for this architecture, we found 8 to be a good choice. Thus, in our selection of k-values, we favored 8 whenever possible in early rounds and then used 4 or 2 as needed in later rounds. The Radix-k results are never worse than binary swap and are up to 1.51 times faster. On average, Radix-k is 1.45 times faster across all of the data points in this test. The slight bump in some of the curves in Figure 4 is due to moving beyond a single rack to multiple Blue Gene racks. This is an artifact of the architecture, not of the algorithm.

4.2 Nonpower-of-two Numbers of Processes

Binary swap is designed to work on a power-of-two number of processes. Extensions such as 2-3 swap to handle the nonpower-of-two cases incur a performance penalty because they are not a part of the original algorithm design. Radix-k is not designed around a particular radix value, so in theory it should accommodate arbitrary numbers of processes more gracefully. In practice, certain numbers of processes will factor into k-values that map onto a particular architecture better than others.

Figure 5 shows the test results for a variety of process counts from 32 to 34,816. The left graph is a higher-resolution test, where the process count increases by 32 at each data point. In the right graph, above 1024 processes, the increment is one additional Blue Gene rack (1,024 nodes). The left graph exhibits considerable variability from one data point to the next, but there is not the same dependence on powers of

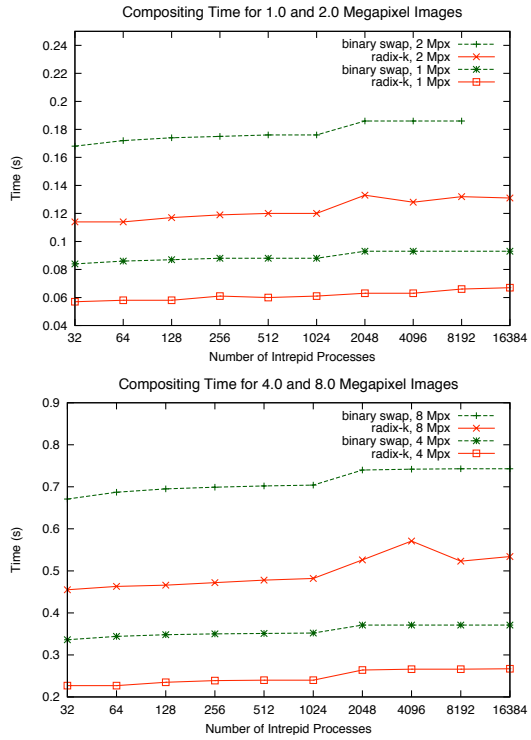

Figure 4: Scalability over a wide range of processor counts and image sizes. Process counts are powers of two from 32 to 16,384. Top: 1- and 2-megapixel image sizes. Bottom: 4- and 8-megapixel image sizes.

two as in other algorithms. In comparison, 2-3 compositing (see Fig. 6 of Yu et al.[32]) displays a constant slowdown of approximately two times between the data points that are powers of two and the rest.

4.3 Selecting the k Vector

Radix-k is configurable to various architectures, but one needs to be able to find what the optimal configurations are. This subsection studies how to select the best values for the vector **k**. These values do not depend on the image, but are related to the network topology (mesh or torus, for example) and to physical placement of processes onto that topology. The following tests are conducted on Intrepid, at 256 processes, but they are representative of larger numbers of processes. Image size is 2 megapixels.

The left panel of Figure 6 shows the performance for different combinations and permutations of k-values, listed along the horizontal axis. At the far left of this graph is binary swap, while direct-send is at the far right. The upper curve measures performance when a 3D mesh is selected; at 256-node partitions and below, the wrap-around links are unavailable making the network a mesh instead of a torus. By allocating a 512-node partition, however, even if 256 nodes are used, a true 3D torus exists. This is the lower curve. The availability of a torus increases performance, as the graph shows, and the effect is more dramatic toward the right side of the graph where the k-values are larger. The irregular spikes in both curves are not noisy data; rather, they are various configurations when nodes that need to communicate large messages are farther apart in the torus or mesh. That is, messages may need to make several hops or congregate in hot spots, causing congestion. Overall, the best k-values in the left graph occur when fairly large radices such as 16 or 32 appear in the **k** vector.

When using high-radix configurations, it is beneficial to control where processes land in the network topology, in order to avoid the hot spots mentioned above. Intrepid provides a mechanism for process mapping; the right panel of Figure 6 examines the effect of this process mapping on Radix-k performance. The top curve is identical to the left panel; it includes no mapping. The lower two curves show the effect of mapping increasing ranks to $2 \times 2 \times 2$ and $4 \times 4 \times 4$ physical blocks in the torus. For example, in the former case, the first 8 ranks would map to the first $2 \times 2 \times 2$ blocks, the next 8 ranks would map into the next adjacent block, and so forth. We tested a number of block sizes besides those shown. The right panel performance improved over that of the left panel, and the optimal settings have shifted to **k** vectors that include the radix 8, such as [8, 8, 2, 2] and [8, 8, 4]. Intuitively, having the early rounds communicate with radix 8 makes each $2 \times 2 \times 2$ block operate as a 3D hypercube, an efficient communication kernel for a multiported

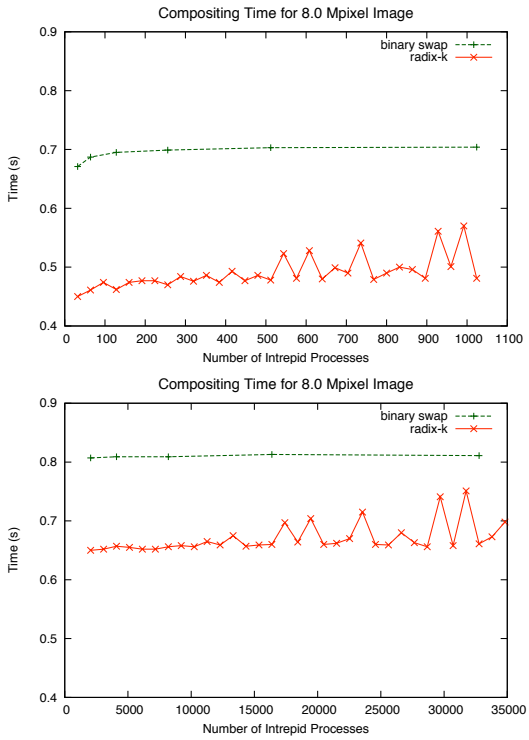

Figure 5: Performance for a variety of processor counts, both powers of two and primarily nonpowers-of-two counts, compared with binary swap at the processor counts that are powers of two. Top: process counts from 32 to 1,024 in increments of 32. Bottom: the same test is continued at larger scale, from 1,024 to 8,192 process in increments of 1,024.

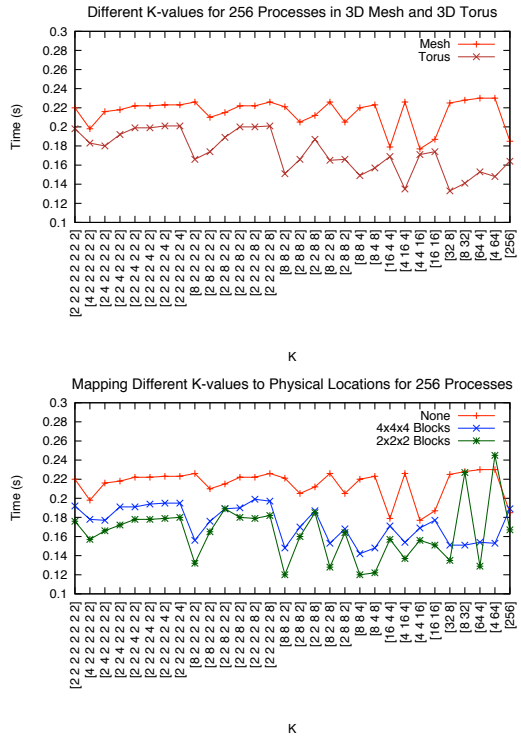

Figure 6: Performance of different k-values for 256 processes. K-values increase from left to right. Top: High-radix values such as 16 and 32 perform better than low values, and the presence of wrap-around torus links is significant. Bottom: Performance is further improved by mapping process ranks to physical torus locations. The "sweet spot" for this architecture occurs when k-values are biased toward radix-8 in conjunction with mapping successive process ranks into physical blocks of $2 \times 2 \times 2$.

3D topology such as the Blue Gene.

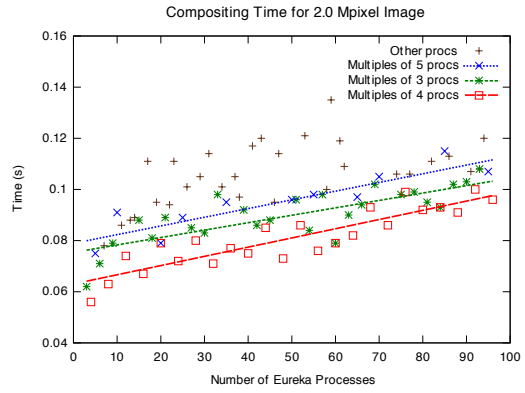

Figure 7: Performance on a cluster architecture can be grouped into regions. A best-fit line for three regions is shown. Process counts that are multiples of 4 perform slightly better than multiples of 3 and 5.

4.4 Cluster Performance

As data continue to grow in size, more analysis and visualization operations will be performed on machines such as Intrepid. Today, however, such machines are still inaccessible for many visualization applications, and these tasks are often performed on smaller clusters. In the next test, we analyze the performance of Radix-k on the 100-node Eureka cluster, with a Myrinet interconnect. One would expect some features of supercomputer interconnects such as multiporting and separate DMA engines for communication to eventually make their way to commodity interconnects, but for the time being, we were not surprised to see less performance gain from using Radix-k on this single-ported network. Nevertheless, Radix-k still provides a useful parameterization of combinations of binary swap and direct-send rounds for arbitrary numbers of processes.

Figure 7 shows these results for a 2-megapixel image size. We tested four image sizes again, from 1 to 8 megapixels; the graph for 2 megapixels is indicative of all the results. Process counts range from 4 to 96, in increments of 1. For our k-values, we factored the process count into prime numbers and arranged these factors in ascending order. Earlier rounds contained smaller factors. For process counts that are powers of two, this equates to a binary swap where $\mathbf{k} = [2, 2, ...]$.

The scatterplot in Figure 7 shows a broad spread of results; but by grouping the points into four categories, interesting patterns emerge. The points in the lower half of the graph cluster around three lines. The slopes of these lines is governed by the interconnect latency and bandwidth, not the algorithm. By comparison, the slope in the earlier Intrepid tests was much less.

Process counts that are multiples of four perform better than the rest. Those points are clustered around a line that ranges from 0.06 s at the left end to 0.08 s at the right. These points include the powers of two as well as other points. The next-best category is composed of process counts that are multiples of three. These points form a similar line, slightly offset from the first. Multiples of five have a similar pattern. The remaining points are scattered; some of these are prime numbers where Radix-k performs only one direct-send.

Users do not always choose arbitrary numbers of processes to run a job. Multiples of two, three, four, five, or ten are common. Figure 7 shows that for such configurations, Radix-k performs in approximately 33% of the optimal time across a variety of process counts, indicated by the distance between the upper and lower best-fit lines. While Radix-k does not have the same advantage on Eureka as on Intrepid, this result shows that Radix-k is a useful tool within a cluster environment as well as in a supercomputer environment.

5. SUMMARY

5.1 Conclusions

Radix-k trades the number of message partners with number of rounds, and it does so in a round-by-round, configurable manner. The Radix-k algorithm for image compositing builds on the previous contributions of binary swap and direct-send. By parameterizing the number of message part-

ners in a round, it unifies these two algorithms that previously were treated separately. By factoring the number of processes into a number of rounds with a separate radix for each round, the algorithm embodies binary swap, direct-send, and combinations in between.

By using higher radices, messages can occur in parallel with each other and with computation. Of course, this improves performance only when the underlying architecture can exploit the additional concurrency. As we saw in the tests performed on the Blue Gene/P, modern networks that are multiported and that have DMA access to messages can benefit Radix-k. The case study on Intrepid is relevant for two reasons. First, as simulations grow in size and scope, more analysis such as scientific visualization will need to occur on the same supercomputer as the simulation. This situation is true whether the analysis is performed after simulation or concurrent with it. Second, hardware innovations at the supercomputer scale tend to migrate to other architectures such as clusters.

In the cluster study, Radix-k did not exhibit the same performance gains as in the supercomputer study, since the interconnect and node hardware, together with (perhaps) the MPI implementation, saturate at small radices. By selecting more rounds of small k-values, however, as we did by choosing ascending prime factors, we still were able to achieve consistent performance over a variety of process counts.

5.2 Future Work

This research offers several avenues for continued exploration. In the experiments thus far, we did not cull inactive pixels. Instead, we considered the worst-case scenario when all pixels are used. Especially in the early rounds, this may not be the case in practice. It will be interesting to see how such an optimization compares between Radix-k and binary swap, for example. A smaller message size resulting from the active pixel optimization may favor the use of higher radices in early rounds. We plan to explore this hypothesis.

We have not done much to optimize the pixel compositing computation. With the ubiquity of multicore processors, a natural next step is to parallelize the computing of the *over* operator across several pixels. The amount of overlap between communication and computation depends on the relative rates of those two steps; maximum overlap occurs when they are approximately equal. Improving the speed of the computation on low-power compute nodes such as the Blue Gene can extend the effectiveness of the algorithm. Our work in profiling the algorithm with tools such as MPE and Jumpshot can prove useful as we measure the amount of overlap and look for ways to increase it.

There exists a trend in communication algorithms toward self-tuning. So far, we have performed small empirical experiments to select what we think are appropriate k-values for a particular architecture. Another area for further study is optimizing and automating this process. We foresee the algorithm's being able to select its own optimal set of parameters for a given set of initial conditions.

6. ACKNOWLEDGMENT

We gratefully acknowledge the use of the resources of the Argonne Leadership Computing Facility at Argonne National Laboratory. 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.

7. REFERENCES

[1] *Argonne Leadership Computing Facility*. 2009. http://www.alcf.anl.gov/.

[2] J. Ahrens and J. Painter. Efficient sort-last rendering using compression-based image compositing. In *Proc. Eurographics Parallel Graphics and Visualization Symposium 2008*, Bristol, United Kingdom, 1998.

[3] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. Geijn, and J. Watts. Interprocessor collective communication library (intercom. In *In Proceedings of the Scalable High Performance Computing Conference*, pages 357–364. IEEE Computer Society Press, 1994.

[4] M. Barnett, D. G. Payne, R. A. van de Geijn, and J. Watts. Broadcasting on meshes with wormhole routing. *Journal of Parallel Distributed Computing*, 35(2):111–122, 1996.

[5] M. Bernaschi and G. Iannello. Collective communication operations: Experimental results vs. theory. *Concurrency*, 10(5):359–386, 1998.

[6] J. Bruck, C.-T. Ho, S. Kipnis, and D. 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.

[7] X. Cavin, C. Mion, and A. Fibois. Cots cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proc. IEEE Visualization 2005*, pages 111–118, 2005.

[8] A. Chan, W. Gropp, and E. Lusk. An efficient format for nearly constant-time access to arbitrary time intervals in large trace files. *Scientific Programming*, 16(2-3):155–165, 2008.

[9] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. Collective communication: theory, practice, and experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(13):1749–1783, 2007.

[10] E. Chan, R. van de Geijn, W. Gropp, and R. 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.

[11] W. M. Hsu. Segmented ray casting for data parallel volume rendering. In *Proc. 1993 Parallel Rendering Symposium*, pages 7–14, San Jose, CA, 1993.

[12] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.

[13] S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, M. Blocksome, A. Faraj,

J. Parker, J. Ratterman, B. Smith, and C. 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.

[14] S. Kumar, G. Dozsa, J. Berg, B. Cernohous, D. Miller, J. Ratterman, B. Smith, and P. 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.

[15] 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.

[16] K.-L. Ma and V. Interrante. Extracting feature lines from 3d unstructured grids. In *Proc. IEEE Visualization 1997*, pages 285–292, Phoenix, AZ, 1997.

[17] 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):59–68, 1994.

[18] 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.

[19] K. Moreland, B. Wylie, and C. 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] U. Neumann. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *Proc. 1993 Parallel Rendering Symposium*, pages 97–104, San Jose, CA, 1993.

[21] U. Neumann. Communication costs for parallel volume-rendering algorithms. *IEEE Computer Graphics and Applications*, 14(4):49–58, 1994.

[22] J. Nonaka, K. Ono, and H. Miyachi. Theoretical and practical performance and scalability analyses of binary-swap image composition method on ibm blue gene/l. In *Proc. 2008 International Workshop on Super Visualization (unpublished manuscript)*, Kos, Greece, 2008.

[23] T. Porter and T. Duff. Compositing digital images. In *Proc. 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 253–259, 1984.

[24] D. Pugmire, L. Monroe, A. DuBois, and D. 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] R. Rabenseifner. *New Optimized MPI Reduce Algorithm*. 2004. http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html.

[26] R. Rabenseifner and J. L. Traff. More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems. In *Proc. EuroPVM/MPI 2004*, pages 36–46, Budapest, Hungary, 2004.

[27] A. Stompel, K.-L. Ma, E. B. Lum, J. Ahrens, and J. Patchett. Slic: Scheduled linear image compositing for parallel volume rendering. In *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 33–40, Seattle, WA, 2003.

[28] A. Takeuchi, F. Ino, and K. Hagihara. An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. *Parallel Comput.*, 29(11-12):1745–1762, 2003.

[29] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.

[30] J. L. Traff. An improved algorithm for (non-commutative) reduce-scatter with an application. In *Proc. EuroPVM/MPI 2005*, pages 129–137, Sorrento, Italy, 2005.

[31] J. L. Traff, A. Ripke, C. Siebert, P. Balaji, R. Thakur, and W. Gropp. A simple, pipelined algorithm for large, irregular all-gather problems. In *Proc. EuroPVM/MPI 2008*, Dublin, Ireland, 2008.

[32] H. Yu, C. Wang, and K.-L. 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.