Eurographics Symposium on Parallel Graphics and Visualization (2015)
C. Dachsbacher, P. Navrátil (Editors)

TOD-Tree: Task-Overlapped Direct send Tree Image Compositing for Hybrid MPI Parallelism

A.V.Pascal Grosset, Manasa Prasad, Cameron Christensen, Aaron Knoll & Charles Hansen

Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA

Abstract

Modern supercomputers have very powerful multi-core CPUs. The programming model on these supercomputer is switching from pure MPI to MPI for inter-node communication, and shared memory and threads for intra-node communication. Consequently the bottleneck in most systems is no longer computation but communication between nodes. In this paper, we present a new compositing algorithm for hybrid MPI parallelism that focuses on communication avoidance and overlapping communication with computation at the expense of evenly balancing the workload. The algorithm has three stages: a direct send stage where nodes are arranged in groups and exchange regions of an image, followed by a tree compositing stage and a gather stage. We compare our algorithm with radix-k and binary-swap from the IceT library in a hybrid OpenMP/MPI setting, show strong scaling results and explain how we generally achieve better performance than these two algorithms.

Categories and Subject Descriptors (according to ACM CCS):
I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

With the increasing availability of High Performance Computing (HPC), scientists are now running huge simulations producing massive datasets. To visualize these simulations, techniques like volume rendering are often used to render these datasets. Each process will render part of the data into an image and these images are assembled in the compositing stage. When few processes are available, the bottleneck is usually the rendering stage but as the number of processes increase, the bottleneck switches from rendering to compositing. Hence, having a fast compositing algorithm is essential if we want to be able to visualize big simulations quickly. This is especially important for in-situ visualizations where the cost of visualization should be minimal compared to simulation cost so as not to add overhead in terms of supercomputing time [YWG*10]. Also, with increasing monitor resolution, the size and quality of the images that can be displayed has increased. It is common for monitors to be of HD quality which means that we should be able to composite large images quickly.

Though the speed of CPUs is no longer doubling every 18-24 months, the power of CPUs is still increasing. This has been achieved though better parallelism [SDM11]; having more cores per chip and bigger registers that allows several operations to be executed for each clock cycle. It is quite common now to have about 20 cores on chip. With multi-core CPUs, Howison et al. [HBC10], [HBC12] found that using threads and shared memory inside a node and MPI for inter-node communication is much more efficient than using MPI for both inter-node and intra-node for visualization. Previous research by Mallon et al. and Rabenseifner et al. [MTT*09], [RHJ09], summarized by Howison et al. indicate that the hybrid MPI model results in fewer messages between nodes, less memory overhead and outperforms MPI only at every concurrency level. Using threads and shared memory allows us to better exploit the power of these new very powerful multi-core CPUs.

While CPUs have increased in power, network bandwidth has not improved as much, and one of the commonly cited challenges for exascale is to devise algorithms that avoid communication [ABC*10] as communication is quickly becoming the bottleneck. Yet the two most commonly used compositing algorithms, binary-swap and radix-k, are focused on distributing the workload. While this was very important in the past, the power of current multi-core CPUs means that load balancing is no longer as important. The

crucial aspect is now *minimizing communication*. Radix-k and binary-swap can be split into two stages: compositing and gathering. Moreland et al. [MKPH11] show that when the number of processes increase, the compositing time decrease but since the gathering time increases, the total overall time increases.

The key contribution of this paper is the introduction of Task Overlapped Direct send Tree, TOD-Tree, a new compositing algorithm for Hybrid/MPI parallelism that minimizes communication and focuses on overlapping communication with computation. There is less focus on balancing the workload and instead of many small messages, larger and fewer messages are used to keep the gathering time low as the number of nodes increases. We compare the performance of this algorithm with radix-k and binary-swap on an artificial and combustion dataset and show that we generally achieve better performance than these two algorithm in a hybrid setting.

The paper is organized as follows: in Section 2, different compositing algorithms that are commonly used are described. In Section 3, the TOD-Tree algorithm is presented and its theoretical cost described. Section 4 shows the results of strong scaling for an artificial dataset and a combustion simulation dataset, and the results obtained are explained; Section 5 discusses the conclusion and future work.

2. Previous Work

Distributed volume rendering is now commonly used in the scientific visualization community. Software like VisIt [CBB*05] and Paraview [HAL04] are extensively used by scientists for visualization on HPC systems. Parallel rendering algorithms can be generally classified as sort-first, sort-middle or sort-last [MCEF94] where sort-last is the most widely used. In sort-last, each process loads part of the data and renders it producing an image. These images are then blended together during the compositing stage to produce the final full image of the dataset. No communication is required in the loading and rendering stage but the compositing stage can require extensive communication. Thus different algorithms have been designed for sort-last compositing.

One of the oldest sort-last compositing algorithm is direct send. Direct send can refer to the case where all the processes send their data directly to the display process which blends the images, sometimes referred to as serial direct send, or to the case where each process takes responsibility for one section of the final image and gathers data for that section from all the other processes [Hsu93], [Neu94], sometimes referred to as parallel direct send. For parallel direct send, there is a gather section where the display node gathers the different sections from each process. The SLIC compositing by Stompel et al. [LMAP03] is essentially an optimized direct send. Pixels from the rendered image from each process are classified to determine if they can be sent directly to the display process (non overlapping pixels) or will require compositing. Then processes are assigned regions of the final image for which it has data and pixel exchanges are done through direct send. On GPUs though, parallel direct send is very popular because of its flexibility. Eilemann et al. [EP07] show that the performance of parallel direct send is comparable to binary swap and sometimes even better. Rizzi et al. [RHI*14] compare the performance of serial and parallel direct send for which they get very good results as GPUs are very fast. However, in both cases, the main bottleneck is the network performance which negatively impacts the performance of the algorithm.

In binary tree compositing techniques [SGS91], one of the leaves sends its data to the other leaf in the pair which does the compositing. The leaf which has sent its data is now idle. The main issue with this technique is half of the nodes go idle at each stage and this results in load imbalances. However, now that computation is reasonably cheap, this could again be a viable technique but tree compositing techniques also send full images at each stage making communication slow. Binary-swap by Ma et al. [MPHK93] improves the load balancing of binary tree compositing by keeping all the processes active in compositing until the end. The processes are grouped in pairs and initially each process in the pair takes responsibility for half of the image. Each process sends the half it does not own and blends the half it owns. In the next stage, processes that are authoritative on the same half exchange information in pairs again so that each is now responsible for a quarter of the image. Compositing proceeds in stages until each process has $1/p$ of the whole image where p is the number of process involved in the compositing. Once this is done, each process sends its section to the display process. Binary-swap has been subsequently extended by Yu et al. [YWM08] to deal with non power of 2 processes. In Radix-k, introduced by Peterka et al. [PGR*09], the number of processes p is factored in r factors so that k is a vector where $k = [k_1, k_2, ..., k_r]$. The processes are arranged into groups of size k_i and exchange information using direct send. At the end of a round, each process is authoritative on one section of the image in its group. In the next round, all the processes with the same authoritative partition are arranged in groups of size k_{i+1} and exchange information. This goes on for r rounds until each process is the only one authoritative on one section of the image. Both binary-swap and radix-k have a gather stage where the display process has to gather the data spread among the p processes. If the vector k has only one value which is equal to p, radix-k behaves like direct send. If each value of k is equal to 2, then it behaves likes binary-swap. Radix-k, binary-swap and direct send are all available in the IceT package [Mor11] which also adds several optimizations such as telescoping and compression which have been described in [MKPH11].

Also, there are algorithms like the Shift-Based Parallel Image Compositing on InfiniBand Fat-Trees [CD12] that focus on image compositing on specific infiniband networks.

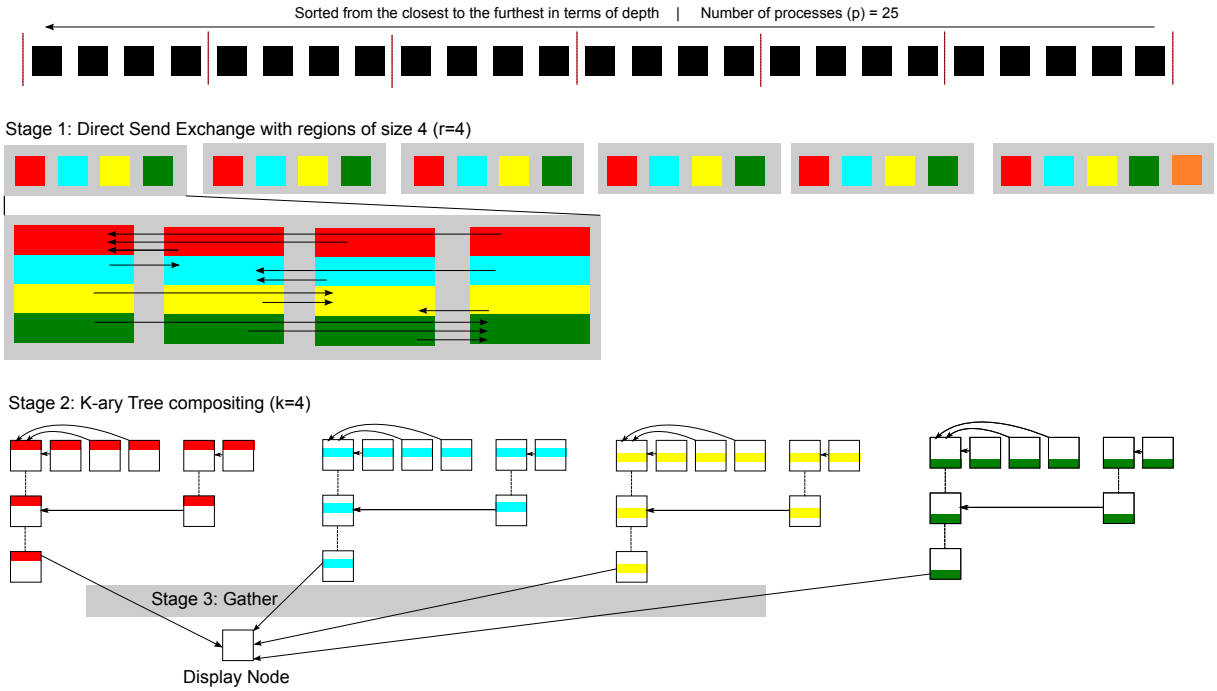

Figure 1: *The three stages of the compositing algorithm with r=4, k=4, and the number of nodes p=25. Red, blue, yellow and green represent the first, second, third an fourth quarter of the image.*

Other interconnects are common in the HPC world, such as are Crays and Blue Gene/Q systems [KBVH14]. It would be best for compositing algorithms to not be tied to particular network infrastructure. In this paper, we show the performance of our algorithm on both Cray and Infiniband networks.

Finally, the work by Howison et al. [HBC10], [HBC12], comparing volume rendering using only MPI versus using MPI and threads is the closest one to his paper and can be seen as a predecessor to this work. They clearly established that using MPI and threads is the way forward as it minimizes exchange of messages between nodes and results in faster volume rendering. However, for compositing, they only used MPI_Alltoallv but do mention in their future work the need for better compositing algorithm. Our work addresses that by presenting a new compositing algorithm for hybrid OpenMP/MPI.

3. Methodology

Since our algorithm has been tuned to work on hybrid MPI architectures, a process in our case is not a core but a node. At the start of the compositing phase, each node has an image that has been rendered from the part of the dataset it has loaded. Each image also has an associated depth from the viewpoint. Each node can know the depth of the images associated with other processes either through nodes sharing that information with each other or since a k-d tree is often used to determine which part of a dataset a node should load, the latter could determine the depth of every other node. Each node sorts nodes by depth to know the correct order in which blending should be done. If the correct order is not used, the final image will not be correct. Also from the extents of the dataset and the projection matrix used, it is easy to determine the height h, the width w and the number of pixels p in the final image.

3.1. Algorithm

The algorithm, TOD-Tree (Task-Overlapped Direct send Tree), has three stages. The first stage is a grouped direct send followed by a k-ary tree compositing in the second stage and the last stage is a gather to the display process. In all stages, asynchronous communication is used to overlap communication and computation. We will first start by describing the algorithm conceptually.

In the first stage, the nodes are arranged into groups of size r, which we will call a locality. Each node in a locality will be responsible for a region equivalent to $1/r$ of the final image. If r is equal to 4, there are 4 nodes in a locality and each is responsible for a quarter of the final image. This is shown in stage 1 of figure 1. The nodes in each locality exchange sections of the image in a direct send fashion so that at the end of stage 1, each node is authoritative on a dif-

ferent $1/r$ of the final image. The colors red, blue, yellow and green in figure 1 represent the first, second, third and fourth quarter of the final image that each node is authoritative on. Also in figure 1, there are 25 processes initially. In this case the last locality will have 5 instead of 4 nodes and the last node, colored orange in the figure, will send its regions to the first r node in its locality but will not receive any data. In the second stage, the aim is to have only one node that is authoritative on a specific $1/r$ region of the final image. The nodes having the same region at the end of stage 1 are arranged in groups of size k. Each node in a group sends its data to the first node in its group which blends the pixels. This is similar to a k-ary tree compositing [SGS91], [YWM08], [MPHK93]. If, as shown in stage 2 of figure 1, there are 6 processes that have the same quarter of the image, two rounds are required until there is only one node which is authoritative on a quarter of the image. Finally, these nodes blend their data with the background and send it to the root node which assembles the final image, stage 3 in the figure 1.

We will now describe in detail how we implement each stage of the algorithm, paying attention to the order of operation to maximize overlapping of communication with computation.

Algorithm 1: Stage 1 - Direct Send

```
Determine the nodes in its locality
Determine region of the image the node owns
Create a buffer for receiving images
Advertise the receive buffer using async MPI Recv
if node is in first half of locality then
    Send front to back using async MPI Send
else
    Send back to front using async MPI Send
Create a new image buffer
Initialize the buffer to 0
if node is in first half of region then
    Wait for images to come in front to back order
    Blend front to back
else
    Wait for images to come in back to front order
    Blend back to front
Deallocate receive buffer
```

Algorithm 1 shows the how we have set up the direct send. There are a few design decisions to make for this part. Clearly, asynchronous MPI send and receive is the way to go if we want to maximize overlapping of communication and computation. Posting the MPI receive before the send allows messages to be received directly in the target buffer instead of being copied in a temporary buffer upon being received and later copied to the target buffer thereby decreasing efficiency. To minimize link contention, not all nodes try to send to one node. Depending on where they are in the locality, the sending order is different. The buffer used as sending buffer is the original image that the node has. To minimize memory use, we have only one blending buffer and so we need the data to be available in the correct order to start blending. The alternative would have been to blend on the fly as images are received but this will require creating and initializing many new buffers which can have a very high memory cost when the image is large. In some tests that have been carried out, we saw that it did not significantly improve the performance to outweigh the cost of allocating that much memory. The blending buffer also needs to be initialized to 0 for blending and this is a somewhat slow operation. To amortize this cost, we do it after the MPI operations have been initialized so that receiving images and the initialization can proceed in parallel.

The second stage is a tree compositing shown in algorithm 2. Again, the receive buffer is advertised early to maximize efficiency. Another optimization that we have added is to blend with the background color in the last round while waiting for data to be received to overlap communication and computation. Also while the alpha is needed when compositing, it is not needed in the final image and so in the last step, we separate the alpha so that in the last stage, algorithm 3, we do not need to send 4 channels: red, green, blue and alpha but only red, green and blue. This allows the last send to be smaller and makes the gather faster.

Algorithm 2: Stage 2 - Tree Region

```
Determine if the node will be sending or receiving
Create a buffer for receiving images
for each round do
    if sending then
        Send data to destination node
    else
        Advertise the receive buffer using async MPI
        Recv
        if last round then
            Create an opaque image for blending
            receiving images
            Create an alpha buffer for blending
            transparency
            Blend current image with the background
            Receive images
            Blend in the opaque buffer
        else
            Receive images
            Blend in image buffer created in stage 1
Deallocate image buffer created in stage 1
Deallocate receive buffer
```

Finally, the last stage of the algorithm is a simple gather from the nodes that still have data. Since we already did the blending with the background in the previous stage, this is just a matter of receiving the image. At the end of the last stage, we also deallocate the send buffer that was being used in stage 1 to send images. If that is done in the earlier stages of the algorithm, it often involves having to wait for the im-

ages to have been sent but in stage 3, the images should have already been sent and so no waiting is required. This has been confirmed with some tests that we carried out.

Algorithm 3: Stage 3 - Gather

```
Create empty final image
if Node has data then
    Send opaque image to display node
else
    if display node then
        Advertise final image as receive buffer
Deallocate send buffer from stage 1
```

The two parameters to choose for the algorithm are the number of regions r and a value for k. r determines the number of regions that an image is split into and while doing so does load balancing. As we increase the number of nodes, increasing the value of r gives us better performance. k is used to control how many rounds should the tree compositing stage has. It is usually better to keep the number of rounds low.

3.2. Theoretical Cost

We are now going to analyze the theoretical cost of the algorithm using the cost model of Chan et al. [CHPvdG07] that has been used by Peterka et al. [PGR*09] and Cavin et al. [CD12]. Let the number of pixels in the final image be n, the number of processes be p, the time taken for blending one pixel be γ, the latency for one transfer be α and the time for transferring one pixel be β. Stage 1 is essentially several direct sends. The number of sends in a group of size r per process is $(r-1)$ and the number of compositings is $r-1$. Since each of the r group will do the same operation in parallel, the cost for stage 1 is: $(r-1)[(\alpha+\frac{n}{r}\beta)+\frac{n}{r}\gamma]$

The second stage is a k-ary tree compositing. There are r tree compositings going on in parallel. Each tree has p/r processes to composite. The number of rounds is $log_k(p/r)$. For each part of the tree, there are $k-1$ sends. The cost for the k-ary compositing is: $log_k\frac{p}{r}[(k-1)[(\alpha+\frac{n}{r}\beta)+\frac{n}{r}\gamma]]$

The cost for the final gather stage is: $r(\alpha+\frac{n}{r}\beta)$.

The final total cost would thus be:

$$(2r+(k-1)log_k\tfrac{p}{r}-1)(\alpha+\tfrac{n}{r}\beta)+(r+(k-1)log_k\tfrac{p}{r}-1)\tfrac{n}{r}\gamma$$

The cost for radix-k, binary swap and direct send is available in the work of Cavin et al. [CD12] and Peterka et al. [PGR*09].

These equations are useful but fail to capture the overlap of communication and computation. It is hard to predict how much overlap there will be as communication depends on the congestion in the network as well but from empirical observations, we saw that the equation acts as an upper bound for the time that the algorithm will take. For example, the total time taken for 64 nodes on Edison

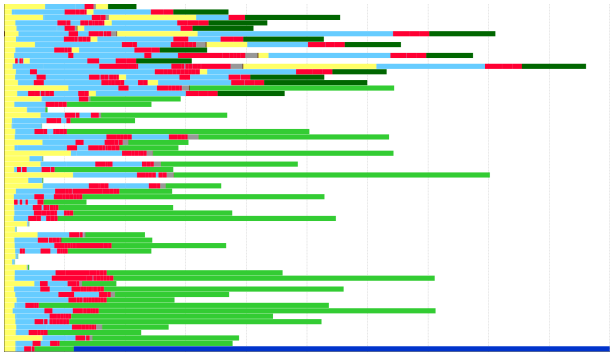

Figure 2: *Profile for 64 nodes for 2048x2048 (64MB) image on Edison at NERSC with r=16, k=8. Red: compositing, green: sending, light blue: receiving, dark blue: receiving on the display process. Total time: 0.012s.*

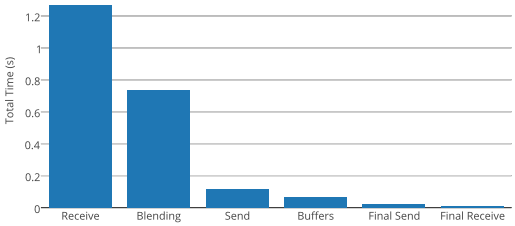

Figure 3: *Breakdown of different tasks in the algorithm.*

was 0.012s for a 2048x2048 image (64MB). Let's now calculate the time using the equation and performance values for Edison on the NERSC website [NER15], α is at least $0.25x10^{-6}s$, the network bandwidth is about 8GB/s, so for one pixel (4 channels each with a floating point of size 4 bytes) $\beta = 1.86x10^{-9}s$. The peak performance is 460.8 Gflops/node, so $\gamma = 8.1x10^{-12}s$. The theoretical time should be around 0.015s. So the model effectively gives a maximum upper bound for the operation but more importantly this calculation shows how much time we are saving by overlapping communication with computation. In the tests that we carried out, we never managed to get 8GB/s bandwidth; we always got less than 8GB/s and yet the theoretical value is still greater than the actual value we are measuring.

Figure 2 shows the profile for the algorithm using an internally developed profiling tool. All the processes start with setting up buffers and advertising their receive buffer which is shown colored yellow in the diagram. This is followed by a receive/waiting to receive section, colored blue and blending section colored in red. All receive communication is through asynchronous MPI receive while the sends for stage 1 is asynchronous and the rest are blocking sends. The dark green represents the final send to the display node and the dark blue indicates the final receive on the display node. As can be clearly seen, most of the time is being spend commu-

nicating or waiting for data to be received from other nodes. A breakdown of the total spent by 64 nodes on Edison is shown in figure 3.

As previously mentioned, the most time consuming operations are send and receive. This is one of the reasons why load balancing is not as important anymore, and using tree style compositing is not detrimental to our algorithm.

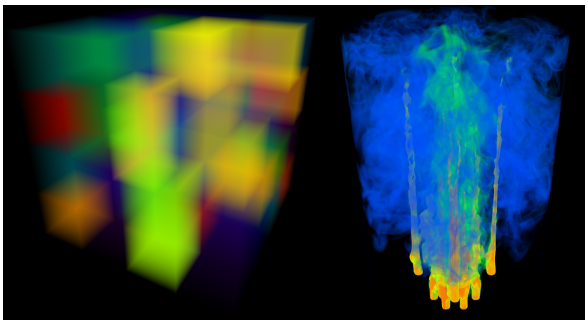

Figure 4: *Left: Synthetic dataset, Right: Combustion dataset.*

4. Testing and Results

We have compared our algorithm against radix-k and binary-swap from the IceT library [MKPH11]. We are using the latest version of the IceT library, from the IceT git repository (`http://public.kitware.com/IceT.git`), as it has a new function icetCompositeImage which compared to icetDrawFrame, takes in images directly and is thus faster when provided with pre-rendered images. This function should be available in future releases of IceT.

The two systems that have been used for testing are the Stampede supercomputer at TACC and the Edison supercomputer at NERSC. Stampede uses the Infiniband FDR network and has 6,400 compute nodes which are stored in 160 racks. Each compute node is an Intel SandyBridge processor which has 16 cores per node for peak performance of 346 GFLOPS/node [TAC15]. Since IceT has not been built to take advantage of threads, we did not build with OpenMP on Stampede. Both IceT and our algorithm will be compiled with g++ and O3 optimization. Edison is a Cray X30 supercomputer which uses the dragonfly topology for its interconnect network. The 5,576 nodes are arranged into 30 cabinets. Each node is an Intel IvyBridge processor with 24 cores and has a peak performance of 460.8 GFLOPS/node [NER15]. To fully utilize a CPU and be as close as possible to its peak performance, both threads and vectorization should be used. Both SandyBridge and IvyBridge processors have 256 bit wide registers which can hold up to eight 32 bit floating points; only when doing 8 floating point operations on all cores can we attain peak performance on one node. Crucially, IvyBridge processors offer the vector gather operation, which fetches data from memory and packs them directly into SIMD lanes. With newer compilers, this can improve performance dramatically. On Edison we fully exploit IvyBridge processors using OpenMP [DM98] and auto-vectorization with the Intel15 compiler.

The two datasets used for the tests are shown in figure 4. The artificial dataset is a square block where each node is assigned one sub block. The simulation dataset is a rectangular combustion dataset where the bottom right and left are empty. The artificial dataset is a volume of size 512x512x512 voxels and the images sizes for the test are 2048x2048 pixels (64MB), 4096x4096 pixels (256) and 8192x8192 pixels (1GB). The combustion dataset is a volume of size 416x663x416 voxels. For the image size, the width has been set to 2048, 4096 and 8192. The height are 2605, 5204 and 10418 pixels respectively.

On Edison at NERSC, we were able to get access to up to 4,096 nodes (98,304 cores) while on Stampede at TACC we have only been granted access to a maximum of 1,024 nodes (16,384 cores). So in the next section, we will show the performance for these two cases. Each experiment is run 10 times and the results are the average of these runs after some outliers have been eliminated.

4.1. Scalability on Stampede

When running on Stampede, threads are not being used for the TOD-Tree algorithm. Both IceT and our implementation are compiled with g++ and O3 optimization. This is done to keep the comparison fair and also to point to the fact that it is the overlapping of tasks rather than raw computing power that is the most important here. Also, we are not using any compression as most image sizes used by users are small enough that compression does not make a big difference. At 8192x8192 pixels, an image is now 1GB in size and having compression would likely further reduce communication.

Figure 5 shows the strong scaling results for artificial data on Stampede. The TOD-Tree algorithm performs better than binary-swap and radix-k. The staircase like appearance can be explained by the fact that we use the same value of r for pairs of time steps; r=16 for 32 and 64 nodes, r=32 for 128 and 256 and, r=64 for 512 and 1024 and only 1 round was used for the k-ary tree part of the algorithm. Thus with r=32, for 256 nodes, there are 8 groups of direct send while there are only 4 groups of direct send at 128 nodes. So the tree stage must now gather from 7 instead of from 3 processes and so the time taken increases. Also it means that instead of waiting for 3 nodes to complete their grouped direct send, now the wait is for 7 nodes. Increasing the value of r helps balance the workload in stage 1 of the algorithm and reduces the number of nodes that have to be involved in the tree compositing and hence decreases the sending.

For images of size 2048x2048 pixels, compositing is heavily communication bound. As we increase the number

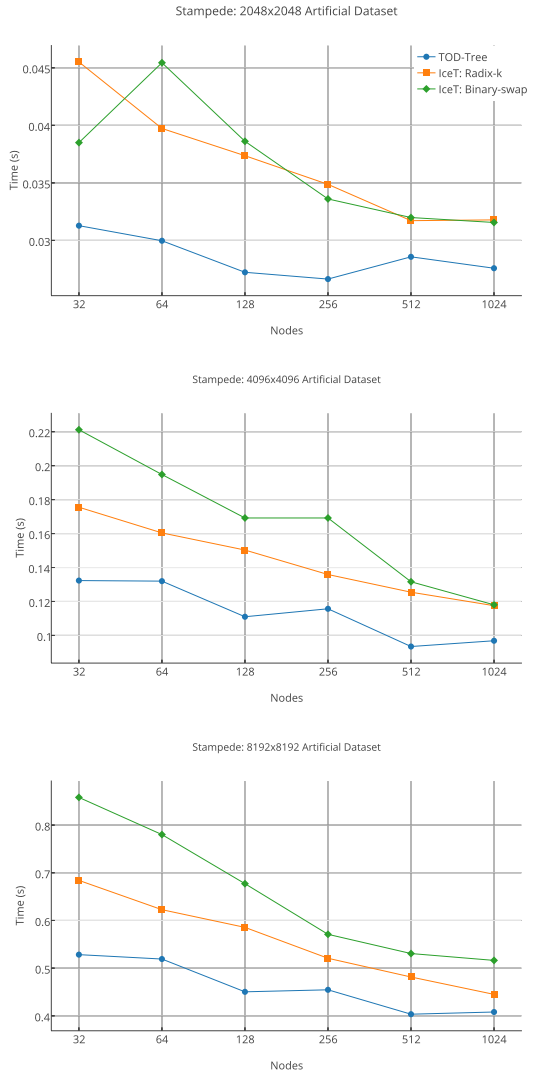

Figure 5: *Scaling for the artificial data on Stampede.*

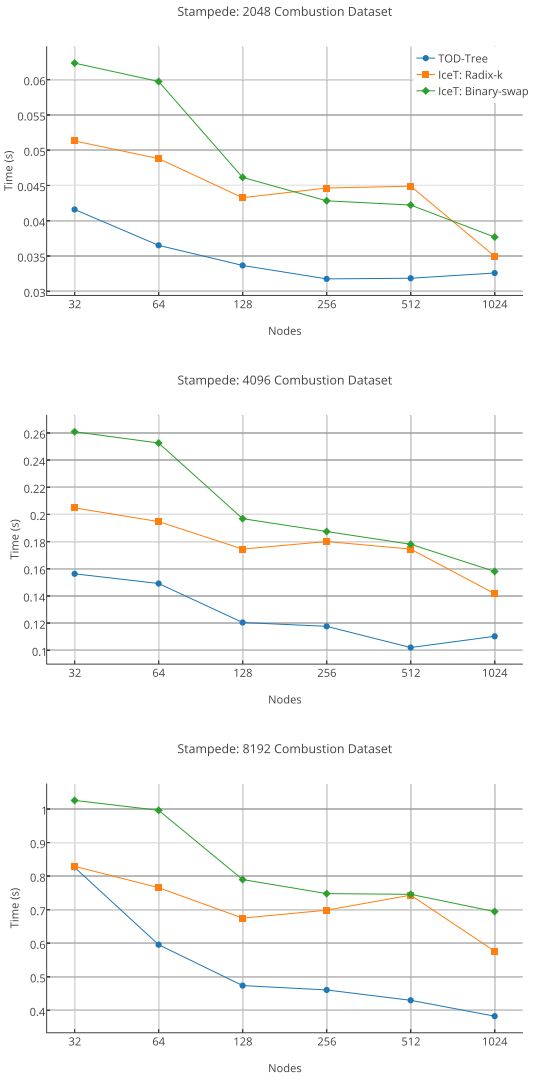

Figure 6: *Scaling for combustion data on Stampede.*

of nodes, each node has very little data and so all the 3 algorithms surveyed perform with less consistency as they become more communication bound and so more affected by load imbalance and networking issues. Communication is the main discriminating factor for small image sizes. For 8192x8192 images, there is less variation as it is more computation bound. Also, at that image size, IceT's radix-k comes close to matching the performance of our algorithm. On analyzing the results for TOD-Tree, we saw that the communication, especially in the gather stage, was quite expensive. While a 2048x2048 image is only 64 MB, a 8192x8192 image is 1GB and transferring such big sizes cost a lot without compression. This is where IceT's use of compression for all communication becomes useful.

In the test case above, we used only 1 round for the tree compositing. For large node counts, more rounds could be used. Figure 7 shows the impact of having different number of rounds for large node counts. For 256 nodes there is an improvement of 0.018 s while it is slower by 0.003 s for 512 nodes and 0.007 seconds for 1024 nodes. So having several rounds barely slows down the algorithm and can even speed up the results.

Figure 6 shows the results for the combustion dataset on Stampede. One of the key characteristics of this dataset is that at the bottom, there are empty regions. This creates load imbalances. Also, the dataset is rectangular and not as uniform as the artificial dataset but it resembles more closely

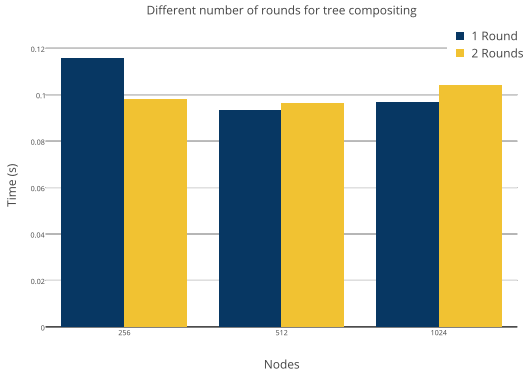

Figure 7: *Varying number of rounds for the artificial dataset for 4096x4096.*

what users are likely to be rendering. The load imbalance creates some different situations from the regular dataset which affect the IceT library a bit more than it affects the TOD-Tree compositing. This is because both binary-swap and radix-k give a greater importance to load balancing and if the data is not uniform, they are likely to suffer from more load imbalances. The TOD-Tree algorithm does not give that much importance to load balancing.

4.2. Scalability on Edison

On Edison, we managed to scale up to 4,096 nodes. The results for strong scaling are shown in Figure 8. The performance of IceT's binary-swap was quite irregular on Edison. For example, for the 4096x4096 image, it would suddenly jump to 0.49 seconds after being similar to radix-k for lower node counts (around 0.11 s). So we decided to exclude binary-swap from these scalings graphs. The staircase pattern is similar to what we see on Stampede for TOD-Tree. Both TOD-Tree and radix-k show less consistency on Edison compared to Stampede. On Edison for 8192x8192 images at 2048 and 4096 nodes are the only instances where radix-k performed better than the TOD-Tree algorithm. Again the main culprit was communication time and TOD-Tree not using compression. In the future, we plan to extend TOD-Tree to have compression for large image sizes as it is clearly not needed for commonly used image sizes.

4.3. Stampede v/s Edison

Figure 9 shows the result of TOD-Tree algorithm on Stampede and Edison. The values of r used are the same as on Stampede for up to 1024 nodes. For 2048 and 4096 nodes, we set r to be 128. As expected, the algorithm is faster on Edison than on Stampede: the interconnect is faster on Edison and the nodes have better peak flop performance. While on Stampede, we are not using threads, on Edison, we are

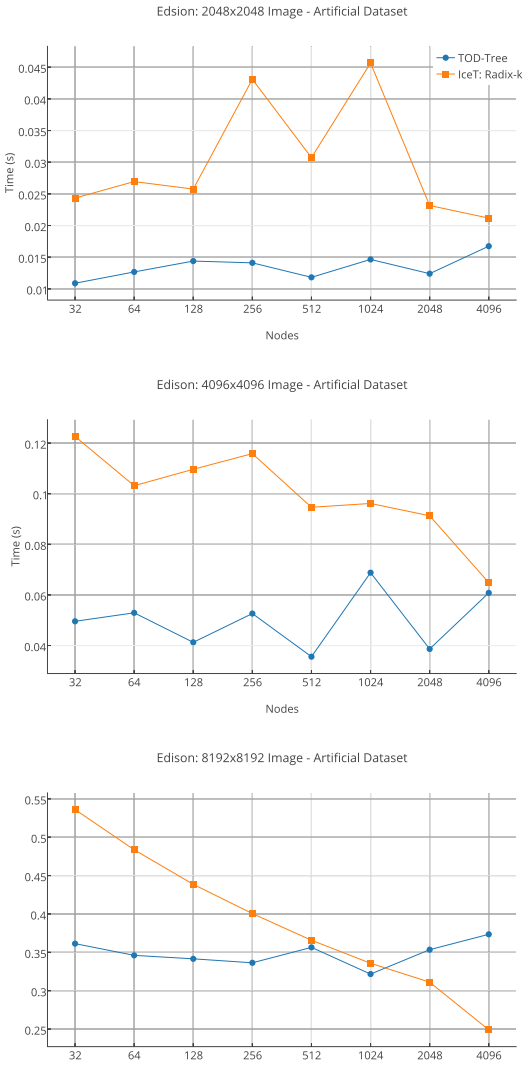

Figure 8: *Scaling for artificial dataset on Edison.*

using threads and vectorization. The gap between the performance is bigger for low node counts, as each node has a bigger chunk of the image to process when few nodes are involved and so a faster CPU makes quite a big difference. As the number of nodes increase, the data to process decreases and so the difference in computing power is less important as the compositing becomes communication bound. The staircase appearance is present in both but is amplified for Edison. On average we are still getting about 16 frames per second for a 256MB images (4096x4096). At 2048 nodes on Edison, the time taken for TOD-Tree decreases as can be seen in the middle chart of figure 8.

Figure 10 shows the equivalent comparison but

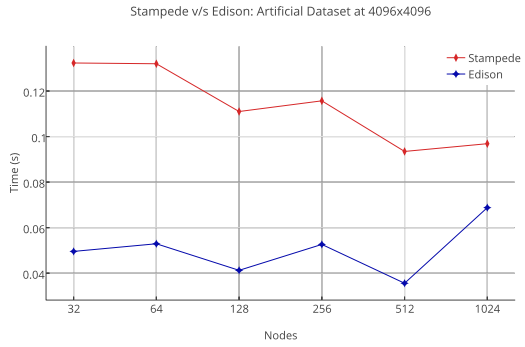

Figure 9: *Comparing Stampede and Edison for up to 1024 nodes for the artificial dataset at 4096x4096 resolution.*

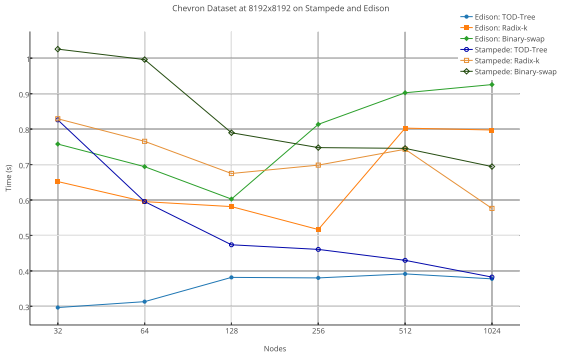

Figure 10: *Comparing Stampede and Edison for up to 1024 nodes for combustion at 8192x10418 resolution.*

8192x10418 images for the combustion dataset. It is interesting to note that the TOD-Tree algorithm on Stampede and Edison though initially have very different performance come closer as the number of nodes increase. This is again because initially there is a lot of computation required and so having a powerful CPU is beneficial but when there is less computation to do, the difference in computation power is no longer that important. IceT performs less consistently for this dataset probably because of the load imbalance inherent in the dataset.

5. Conclusion and Future Work

In this paper, we have introduced a new compositing algorithm for hybrid OpenMP/MPI Parallelism and shown that it generally performs better than the two leading compositing algorithms, binary-swap and radix-k, on the hybrid programming environment. When using the hybrid parallelism, there is a quite a large difference between the computation power available to one node compared to the speed of inter-node communication. Hence, the algorithm must pay much more attention to communication than to computation if we are to achieve better performance at scale.

As future work, we would like to add compression for large image sizes. A heuristic should also be added to determine when compression should be turned on or off based on the size of the data. While 8192x8192 image sizes are quite rare right now (since we lack the ability to display such images properly) it will likely be required in the future and so taking care of this will make the TOD-Tree algorithm more robust.

We would also like to extend out testing to Blue Gene/Q systems as well as this is the only major HPC platform on which the compositing algorithm has not been tested and eventually when they are introduced, the Intel Knights Landing. One of the limitations of this paper is the fact that we did not have enough resources to scale to more than 1024 nodes on Infiniband systems. This is something we would like to address. While we do understand that scaling to 2048, 4096 and above might be quite hard in terms of getting the resources for our runs (it would imply reserving three-quarters or even the full HPC), we would really like to see how our algorithm performs at such large numbers so as to be ready for the exascale era.

6. Acknowledgements

This research was supported by the DOE, NNSA, Award DE-NA0002375: (PSAAP) Carbon-Capture Multidisciplinary Simulation Center, the DOE SciDAC Institute of Scalable Data Management Analysis and Visualization DOE DE-SC0007446, NSF ACI-1339881, and NSF IIS-1162013.

The authors would like to thank the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing access to the clusters Stampede and Maverick, the National Energy Research Scientific Computing Center (NERSC) for providing access to the Edison cluster. We would also like to thank Kenneth Moreland for his help with using IceT.

References

[ABC*10] ASHBY S., BECKMAN P., CHEN J., COLELLA P., COLLINS B., CRAWFORD D., DONGARRA J., KOTHE D., LUSK R., MESSINA P., OTHERS: The opportunities and challenges of exascale computing. *summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science* (2010). 1

[CBB*05] CHILDS H., BRUGGER E. S., BONNELL K. S., MEREDITH J. S., MILLER M., WHITLOCK B. J., MAX N.: A contract-based system for large data visualization. In *Proceedings of IEEE Visualization 2005* (2005), pp. 190–198. 2

[CD12] CAVIN X., DEMENGEON O.: Shift-Based Parallel Image Compositing on InfiniBand TM Fat-Trees. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), Childs H., Kuhlen T., Marton F., (Eds.), The Eurographics Association. doi:10.2312/EGPGV/EGPGV12/129-138. 2, 5

[CHPvdG07] CHAN E., HEIMLICH M., PURKAYASTHA A., VAN DE GEIJN R.: Collective communication: Theory, practice, and experience: Research articles. *Concurr. Comput. : Pract. Exper. 19*, 13 (Sept. 2007), 1749–1783. URL: http://dx.doi.org/10.1002/cpe.v19:13, doi:10.1002/cpe.v19:13. 5

[DM98] DAGUM L., MENON R.: Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng. 5*, 1 (Jan. 1998), 46–55. URL: http://dx.doi.org/10.1109/99.660313, doi:10.1109/99.660313. 6

[EP07] EILEMANN S., PAJAROLA R.: Direct send compositing for parallel sort-last rendering. In *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2007), EG PGV'07, Eurographics Association, pp. 29–36. URL: http://dx.doi.org/10.2312/EGPGV/EGPGV07/029-036, doi:10.2312/EGPGV/EGPGV07/029-036. 2

[HAL04] HENDERSON A., AHRENS J., LAW C.: *The ParaView Guide*. Kitware Inc., Clifton Park, NY., 2004. 2

[HBC10] HOWISON M., BETHEL E. W., CHILDS H.: Mpi-hybrid parallelism for volume rendering on large, multi-core systems. In *Proceedings of the 10th Eurographics Conference on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2010), EG PGV'10, Eurographics Association, pp. 1–10. URL: http://dx.doi.org/10.2312/EGPGV/EGPGV10/001-010, doi:10.2312/EGPGV/EGPGV10/001-010. 1, 3

[HBC12] HOWISON M., BETHEL E., CHILDS H.: Hybrid parallelism for volume rendering on large-, multi-, and many-core systems. *Visualization and Computer Graphics, IEEE Transactions on 18*, 1 (Jan 2012), 17–29. doi:10.1109/TVCG.2011.24. 1, 3

[Hsu93] HSU W. M.: Segmented ray casting for data parallel volume rendering. In *Proceedings of the 1993 Symposium on Parallel Rendering* (New York, NY, USA, 1993), PRS '93, ACM, pp. 7–14. URL: http://doi.acm.org/10.1145/166181.166182, doi:10.1145/166181.166182. 2

[KBVH14] KERBYSON D. J., BARKER K. J., VISHNU A., HOISIE A.: A performance comparison of current hpc systems: Blue gene/q, cray xe6 and infiniband systems. *Future Gener. Comput. Syst. 30* (Jan. 2014), 291–304. URL: http://dx.doi.org/10.1016/j.future.2013.06.019, doi:10.1016/j.future.2013.06.019. 3

[LMAP03] LUM E., MA K.-L., AHRENS J., PATCHETT J.: Slic: Scheduled linear image compositing for parallel vollume rendering. Parallel Visualization and Graphics 2003, IEEE. 2

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *Computer Graphics and Applications, IEEE 14*, 4 (1994), 23–32. doi:10.1109/38.291528. 2

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An image compositing solution at scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 25:1–25:10. URL: http://doi.acm.org/10.1145/2063384.2063417, doi:10.1145/2063384.2063417. 2, 6

[Mor11] MORELAND K.: *IceT Users' Guide and Reference*. Tech. rep., Sandia National Lab, January 2011. 2

[MPHK93] MA K.-L., PAINTER J., HANSEN C., KROGH M.: A data distributed, parallel algorithm for ray-traced volume rendering. In *Parallel Rendering Symposium, 1993* (1993), pp. 15–22, 105. doi:10.1109/PRS.1993.586080. 2, 4

[MTT*09] MALLÓN D. A., TABOADA G. L., TEIJEIRO C., TOURIÑO J., FRAGUELA B. B., GÓMEZ A., DOALLO R., MOURIÑO J. C.: Performance evaluation of mpi, upc and openmp on multicore architectures. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 174–184. URL: http://dx.doi.org/10.1007/978-3-642-03770-2_24, doi:10.1007/978-3-642-03770-2_24. 1

[NER15] NERSC: Edison configuration, February 2015. URL: https://www.nersc.gov/users/computational-systems/edison/configuration/. 5, 6

[Neu94] NEUMANN U.: Communication costs for parallel volume-rendering algorithms. *IEEE Comput. Graph. Appl. 14*, 4 (July 1994), 49–58. URL: http://dx.doi.org/10.1109/38.291531, doi:10.1109/38.291531. 2

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A configurable algorithm for parallel image-compositing applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 4:1–4:10. URL: http://doi.acm.org/10.1145/1654059.1654064, doi:10.1145/1654059.1654064. 2, 5

[RHI*14] RIZZI S., HERELD M., INSLEY J., PAPKA M. E., URAM T., VISHWANATH V.: Performance Modeling of vl3 Volume Rendering on GPU-Based Clusters. In *Eurographics Symposium on Parallel Graphics and Visualization* (2014), Amor M., Hadwiger M., (Eds.), The Eurographics Association. doi:10.2312/pgv.20141086. 2

[RHJ09] RABENSEIFNER R., HAGER G., JOST G.: Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing* (Washington, DC, USA, 2009), PDP '09, IEEE Computer Society, pp. 427–436. URL: http://dx.doi.org/10.1109/PDP.2009.43, doi:10.1109/PDP.2009.43. 1

[SDM11] SHALF J., DOSANJH S., MORRISON J.: Exascale computing technology challenges. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science* (Berlin, Heidelberg, 2011), VECPAR'10, Springer-Verlag, pp. 1–25. URL: http://dl.acm.org/citation.cfm?id=1964238.1964240. 1

[SGS91] SHAW C. D., GREEN M., SCHAEFFER J.: Advances in computer graphics hardware iii. Springer-Verlag New York, Inc., New York, NY, USA, 1991, ch. A VLSI Architecture for Image Composition, pp. 183–199. URL: http://dl.acm.org/citation.cfm?id=108345.108358. 2, 4

[TAC15] TACC: Stampede user guide, February 2015. URL: https://portal.tacc.utexas.edu/user-guides/stampede [cited 04.02.2015]. 6

[YWG*10] YU H., WANG C., GROUT R. W., CHEN J. H., MA K.-L.: In situ visualization for large-scale combustion simulations. *IEEE Comput. Graph. Appl. 30*, 3 (May 2010), 45–57. URL: http://dx.doi.org/10.1109/MCG.2010.55, doi:10.1109/MCG.2010.55. 1

[YWM08] YU H., WANG C., MA K.-L.: Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Piscataway, NJ, USA, 2008), SC '08, IEEE Press, pp. 48:1–48:11. URL: http://dl.acm.org/citation.cfm?id=1413370.1413419. 2, 4