Eurographics Symposium on Parallel Graphics and Visualization (2008)
J. Favre, K. - L. Ma, and D. Weiskopf (Editors)

Parallel Volume Rendering on the IBM Blue Gene/P

Tom Peterka[1], Hongfeng Yu[2], Robert Ross[1], Kwan-Liu Ma[2]
[1]Argonne National Laboratory
[2]University of California at Davis

Abstract

Parallel volume rendering is implemented and tested on an IBM Blue Gene distributed-memory parallel architecture. The goal of studying the cost of parallel rendering on a new class of supercomputers such as the Blue Gene/P is not necessarily to achieve real-time rendering rates. It is to identify and understand the extent of bottlenecks and interactions between various components that affect the design of future visualization solutions on these machines, solutions that may offer alternatives to hardware-accelerated volume rendering, for example, when large volumes, large image sizes, and very high quality results are dictated by peta- and exascale data. As a step in that direction, this study presents data from experiments under a number of conditions, including dataset size, number of processors, low- and high-quality rendering, offline storage of results, and streaming of images for remote display. Performance is divided into three main sections of the algorithm: disk I/O, rendering, and compositing. The dynamic balance among these tasks varies with the number of processors and other conditions. Lessons learned from the work include understanding the balance between parallel I/O, computation, and communication within the context of visualization on supercomputers; recommendations for tuning and optimization; and opportunities for further scaling. Extrapolating these results to very large data and image sizes suggests that a distributed-memory high-performance computing architecture such as the Blue Gene is a viable platform for some types of visualization at very large scales.

Categories and Subject Descriptors (according to ACM CCS): I3.1 [Hardware Architecture]: Parallel processing, I3.2 [Graphics Systems]: Distributed / network graphics, I3.7 [Three-Dimensional Graphics and Realism]: Raytracing, I3.8 [Applications]

1. Introduction

As data sizes and supercomputer architectures grow toward the petascale and beyond, an attractive alternative to rendering on graphics clusters may be to perform software-based visualization directly on parallel supercomputers. Benefits include the elimination of data movement between computation and visualization architectures; the economies of large-scale, tightly coupled parallelism; and the possibility of visualizing a simulation in situ [MWY*07]. This paper examines the second benefit, large numbers of tightly connected processor nodes, within the context of a parallel ray casting volume rendering algorithm implemented on the IBM Blue Gene/P (BG/P) architecture at Argonne National Laboratory.

Volume rendering and parallel volume rendering on supercomputers have been published extensively in the literature, but this is the first such study conducted on BG/P. This research profiles and identifies bottlenecks in the rendering pipeline and suggests modifications to the parallel rendering algorithm to achieve scalability. The study, moreover, is intentionally system-wide and measures end-to-end frame time that includes disk I/O during the visualization of a time-varying dataset. Only by studying the entire visualization pipeline can one get a glimpse of the optimal balance between I/O, computation, communication, and interactivity requirements in the setting of parallel volume rendering on the BG/P.

The experiments include several test conditions, including small- to medium-sized data sets, real-time streaming of output images and offline storage of results, and both low- and high-quality renderings. From the results, one can draw conclusions about how to best leverage the strengths of this architecture in visualization applications. Although the results are specific to a particular algorithm and architecture, the lessons learned

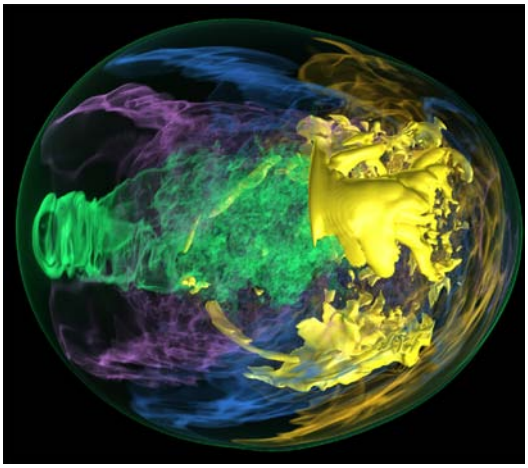

Figure 1: *Visualization of the early stages of supernova collapse.*

Direct correspondence to *tpeterka@anl.gov*

submitted to *Eurographics Symposium on Parallel Graphics and Visualization (2008)*

can potentially apply more broadly to other supercomputer architectures that share some of the same characteristics as the Blue Gene, and to other parallel rendering algorithms as well.

Thus far, we have successfully scaled up to 4096 cores. Remote streaming of a small, time-varying dataset at subsecond frame times was demonstrated. For the data sizes that we currently have available, performance is not faster than other methods and architectures, but we expect the benefits to be apparent at still larger scales. Rather than competing for real-time frame rate with graphics processor (GPU) accelerated rendering in small to moderate scales, parallel supercomputer rendering offers one solution to the peta- and exascale challenges of data sizes that are beyond the scalability of existing methods. For example, when the data size is on the order of billions of voxels and image resolution is on the order of millions of pixels, thousands or even tens of thousands of processors may be justified. Research at these scales is ongoing.

2. Background

Dataset

The dataset shown in Figure 1 is one time-step from a supernova simulation, made available by John Blondin at the North Carolina State University and Anthony Mezzacappa of Oak Ridge National Laboratory [BMD03], through the U.S. Department of Energy's SciDAC Institute for Ultrascale Visualization [SCI07]. The model seeks to discover the mechanism behind the core collapse supernova mechanism, which is the violent death of short-lived, massive stars. A spherical accretion shock instability, SASI, is driven by the response of an initially spherical shock wave to global acoustic modes trapped in the interior.

Visualization plays a key role in understanding the origin of this instability of the supernova shock wave. By manipulating the transparency of the rendered data, scientists can quickly visualize different combinations of variables or isolate features. In this dataset, a single scalar variable angular momentum is stored at 864^3 uniform, structured grid locations. Each of 200 time-steps of time-varying data is stored in a separate file. Files are stored in raw binary format, in 32-bit floating point.

Algorithm

Parallel volume rendering algorithms have been well documented in the literature. Beginning with Levoy's classic ray casting in 1988 [Lev88] and optimizations in 1990 [Lev90], parallel versions began to appear in 1993 with [MPHK93] and [Nue93]. Parker et al. demonstrated efficient parallelism of ray casting in 1999 [PPL*99] for isosurfacing and maximal intensity volume rendering on shared memory architecture. Ma and Camp demonstrated overlapped I/O, rendering, compression and transmission in the context of remote visualization [MC00]. More recently, Yu demonstrated that parallel volume rendering performance can be further improved by overlapping simulation with visualization [YMW04]. Parallel volume rendering has also been studied in the context of cluster computing [CMF05] and in standard visualization toolkits such as VTK [BGM*07], ParaView [MAF07], and VisIt [CBB*05, CDM06].

Our implementation uses post-classification after trilinear interpolation, optionally includes lighting [DCH88, Max95], and does not incorporate hierarchical levels of detail. Sort-last parallelization occurs both in object space and in image space. The dataset is divided into n approximately equal size partitions, where n is the number of processes. Each process computes a completed subimage corresponding to its local data, including local front-to-back compositing of samples along each ray of its local subimage using the "over" operator [PD84] and early ray termination. This standard technique terminates compositing along a ray once the accumulated opacity exceeds a predetermined threshold because further samples along the ray would be occluded.

Stompel et al. [SML*03] provide an overview of various methods for sort-last compositing of the n subimages, and Cavin et al. [CMF05] analyze relative theoretical performance of these methods. These overviews show that compositing algorithms usually fall into one of the following categories: plain or optimized direct send, plain or optimized tree, and parallel pipeline. The direct send approach is easiest to understand; each process requests the subimages from all of those processes that have something to contribute to it [Hsu93, Neu94, MI97]. Since the possibility for network contention is high in direct send, the SLIC [SML*03] optimization attempts to schedule communication. For simplicity and a high degree of parallelism, we use the direct send compositing approach.

Rather than sending compositing data monolithically, tree methods exchange data between pairs of processes, building larger completed subimages at each level of the compositing tree. To keep more processes busy at higher levels on the tree, Ma et al. introduced the binary swap optimization [MPHK94]. Lee et al. discuss a parallel pipeline compositing algorithm in [LRN96] for polygon rendering, although this seldom appears in the context of parallel volume rendering.

We define the time that a frame takes to complete, t_{frame}, as the time from the start of reading the time-step from disk to the time that the final image is ready at the root process. This frame time has three distinct components, and for a given data size, the relative contribution of each component to the total time depends on the number of processes.

$$t_{frame} = t_{io} + t_{render} + t_{composite} \qquad (1)$$

The I/O time, t_{io}, is the length of time required by a collective reading of the time-step data file by all processes simultaneously. The rendering time, t_{render}, is the time that it takes for all processes to complete their local subimage rendering. The compositing time, $t_{composite}$, is the time to composite all subimages into a single image on a single process. The following section describes the implementation of each component in more detail.

Before the execution of the first frame, a one-time initialization step allocates data structures and determines partitioning parameters; the time for this setup is on the order of tens of seconds and because it occurs only once, we omit it from the frame time. During the setup time, data cells are partitioned into block-shaped regions and allocated to processes. This static load balancing scheme

implies that the uniform data distribution can cause an uneven rendering workload when the view matrix or transfer function changes the visibility of subvolumes, as shown by Marchesin et al. [MMD06]. This is not a problem in the compositing step, where all processes participate equally, irrespective of projection area.

Blue Gene Architecture

The Blue Gene/L and Blue Gene/P systems at Argonne National Laboratory provide ample opportunities to experiment with parallel rendering. This work began with 2,048 cores of the BG/L system and, so far, has scaled to 4,096 cores on the BG/P system. The current single rack of BG/P is for testing and development; but in the near future, Argonne's BG/P system will contain 128K cores. In the interest of space, we highlight below just a small sample of relevant features but online documentation from IBM can be found at [IBM07]; the reader is directed there for specifications and configuration diagrams. For our purposes, the key differences between the older BG/L and the new BG/P are that BG/P provides twice as many cores per node, twice the memory footprint, approximately a 2X faster interconnect network, a 1.2X faster clock speed per core, and once completed, many times more available nodes and cores.

Processor cores are grouped together into nodes; the BG/P has four cores per node. Within a node, the cores can operate together to execute one user process, in pairs for two processes, or independently for four user processes, depending on the selected mode. Each BG/P core is a PowerPC 450 850 MHz processor that contains two parallel floating-point units that can execute certain pairs of identical floating-point operations in parallel (SIMD vectorization).

Application processes execute on top of a microkernel that provides basic OS services. The Blue Gene architecture has two separate interconnection networks – a 3D torus for inter-process point-to-point communication and a tree network for collective operations as well as for communicating with I/O nodes. BG/P has one I/O node for every 64 compute nodes. At the front end, the machine has four login nodes that support full Linux functionality.

3. Implementation

I/O

Our volume rendering application is written by using MPI for both communication and I/O and executes with one MPI process on each core. MPI-2 [GGH*96] (a.k.a. MPI-IO) collective file read calls perform data staging, t_{io} in equation 1, allowing each process to read its own portion of the volume in parallel with all of the other processes [YMW04, YM05]. This approach is more efficient than having a single master process read the entire dataset and distribute it to slave processes. More important, for large datasets it does not require a single process to be able to fit the entire dataset into its memory.

For example, the largest dataset tested to date in this work consists of 864^3 voxels, or approximately 2.5 GB per time-step. This is problematic for most workstations; even the BG/P has only 2 GB of memory per node. With collective I/O, however, the total memory footprint of the entire machine - not just of one node - is the upper bound on the maximum data size that can be processed in-core. This memory limit on the current single-rack BG/P is 2 TB, but will grow to 64 TB when the system is complete.

Underlying the MPI-2 collective I/O interface is a parallel file system such as GPFS or PVFS [CLRT00]. By striping data across multiple volumes controlled by a number of file servers, application programs can access noncontiguous regions of a file in parallel. Performance varies depending on whether reads or writes are executed (reads in our case), on the number of I/O nodes being used, and on the size of the partition that each process reads.

Because BG/P is a new system undergoing development, the parallel I/O system is untuned and I/O throughput is expected to increase dramatically in the future. In the meantime, the performance tests use both GPFS and PVFS, depending on which system is currently available. In performance tests, it is important to realize that a parallel file system is shared between all jobs and the login nodes. During timing measurements, we have taken care to restrict others' file system usage and confirmed results over multiple trials.

Rendering

The computation of local subimages, t_{render} in equation 1, is highly parallel and requires no interprocess communication. Its per-core performance is a function of the efficiency of the Blue Gene's compute node: clock speed, pipeline architecture, cache coherence, and the extent to which the code is tuned to optimize these features. Compiler and code optimizations thus far have netted 2X performance gains in t_{render}.

We are currently evaluating low-level performance counters to gauge the use of BG/P's dual floating-point pipeline, and estimate its use to be approximately 5%. Our tests have shown a correlation between this value and the rendering time. However, even with appropriate directives and flags, the compiler still may not be able to vectorize floating point operations, especially when loops contain control flow or function calls.

Profiling tools identify where the code spends the majority of time, and the IBM compiler reports the

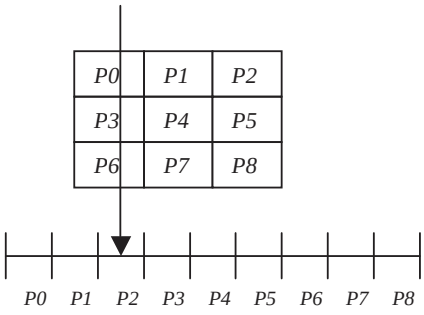

Figure 2: *Direct send compositing divides both the object space and image space among processes.*

locations within that critical kernel where vectorization failed, along with reason for failure. To increase SIMD vectorization within loops, function calls can be replaced by inline functions or macros, and control flow can be replaced by data flow, but these substitutions can be non-trivial in actual code. Tuning the rendering kernel specifically to the BG/P processor architecture is ongoing research.

Compositing

Compositing of parallel volume rendered subimages, $t_{composite}$ in equation 1, is implemented with direct send as follows. At the start of compositing, each of the n processes owns a completed subimage of its portion of the dataset. Next, each of the n processes is assigned responsibility for $1/n$ of the final image area as well. For example, the final image can be divided into n scan lines or rectangles, without any spatial correspondence between the completed subimage from the rendering step and the region of the final image during the compositing step.

For example, consider process *P2* in the 9-process 2D example in Figure 2. The squares represent the 9 subvolumes, and the line along the bottom represents the image divided into 9 regions. (The image need not be aligned with the subvolume axes.) Through a global data structure that all processes share, *P2* knows that it must get the subimages from *P6*, *P3*, and *P0*. It composites the images in front-to-back order according to Equations 2 and 3 to recursively compute color and opacity,

$$i = (1.0 - a_{old}) * i_{new} + i_{old} \quad (2)$$
$$a = (1.0 - a_{old}) * a_{new} + a_{old} \quad (3)$$

Where i represents the intensity (r,g,b) premultiplied by its associated alpha-value, and a represents the accumulated alpha-value or opacity.

The last step is for processes *P1* through *P8* to send their final results to process *P0*, which tessellates them together into one image. The average communication complexity of $t_{composite}$ is $O(n^{4/3} + n)$. The first term, $n^{4/3}$, is because on average, $n^{1/3}$ messages must be sent to each of n recipients in order for the n processes to composite their portion of the final image. The second term, n, represents the gathering of final subimages at the root process.

Streaming and Prefetching

When resulting images are streamed to a remote display device, rather than being stored on disk, the path requires several steps. The reason is that the Blue Gene connects to the outside world only through the front-end login nodes. Therefore, to send an image from one of the compute nodes, it first passes via a socket to the IP address of one of the login nodes. Physically, it actually travels from the compute node to the I/O node assigned to that compute node, and from the I/O node to the login node, but the connection between compute node and associated I/O node is transparent to the programmer. Finally, a daemon running on the login node forwards the data stream to the remote display via a separate socket connection. The connectivity is diagrammed in Figure 3.

Prefetching of time-steps can hide the I/O time when the total number of cores available is sufficient. A multi-pipe application structure, as in Figure 4, is one way to accomplish this. Each of the four pipelines in this example functions independently to process four time steps in parallel. This is not the only way to prefetch data; however, it maps well to the BG/P architecture and to our goals of studying real-time, end-to-end visualization performance that mitigates I/O cost without ignoring it altogether from the equation. Results from this method will appear in a future paper.

4. Performance Data

In November 2007, real-time streaming of the volume rendering application from BG/L was demonstrated, generating and streaming a series of 200 time-steps repeatedly from Argonne in Chicago, Illinois, to the Supercomputing conference exhibit floor in Reno, Nevada. A single time-step is 103 MB; during the one-hour demo, approximately 500 GB of data were processed in real time. The optimal setting for this data size was 512 cores.

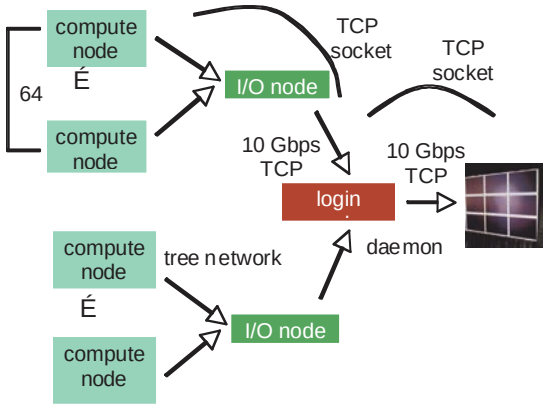

Figure 3: *Connecting a compute node to a remote display is a multi-step process.*

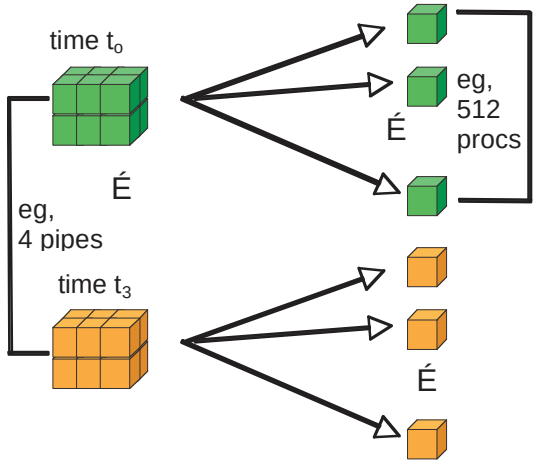

Figure 4: *Processing several time steps simultaneously can extend the degree of parallelism.*

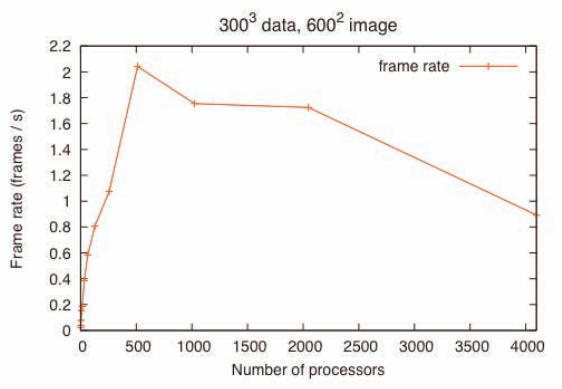

Figure 5: *Total frame rate of BG/P on the 300^3 downsampled dataset is plotted on a logarithmic scale.*

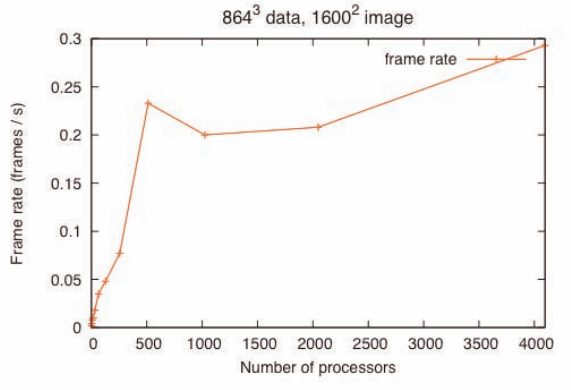

Figure 6: *Total frame rate of BG/P on the full 864^3 dataset is plotted on a logarithmic scale.*

Figure 5 shows updated tests of the same dataset on BG/P, out to 4K processes. The plot shows a peak performance still at 512 cores of approximately 2 frames per second. Performance decreases slightly to 1.75 frames / s through 2048 cores, and drops below 1 frame / s at 4096 cores. This is expected because the total file size divided by a large number of cores results in inefficient I/O and poor compositing behavior. In fact, at 4096 cores, 72% of the frame time is spent in I/O; compositing accounts for an additional 25% while the rendering portion is only 3%. In order to optimize performance, one may either allocate fewer cores or visualize a larger dataset.

In the next test, the full 864^3 dataset is scaled from 2 cores up to 4096 cores, and the result appears in Figure 6. Strong scaling, while still not ideal, improves using this 2.5 GB per time-step data. The full BG/P rack of 4096 cores produces a frame time of approximately 3.5 seconds. I/O performance still dominates: at 4096 cores the breakdown of time is t_{io} = 77%, t_{render} = 10% rendering, and $t_{composite}$ = 13%. However, because the file size is larger, I/O is more efficient at this scale and 4096 cores provides the best overall frame rate.

Figures 5 and 6 appear quite similar in shape up to 2048 cores. For example, the slope of the curve from 256 cores to 512 is steeper than from 128 to 256 cores and 512 cores outperforms 1024 cores in both figures as well. We are currently investigating cache usage as well as I/O and communication patterns in order to explain the similarities in scalability for two different data sizes.

BG/P is capable of executing one, two, or four processes per node. In IBM terminology, these are called *smp* mode, *dual* mode, *vn* mode, respectively. In smp mode, one core performs computation while the other cores idle, with the exception of low-level OS tasks. The total memory footprint of 2GB per node is shared among the four cores in smp mode.

Our tests show approximately 20 - 30% slower performance in dual and vn modes compared to smp mode. The largest increase is in t_{io}, because the number of I/O nodes assigned to a job is a fraction of the number of compute *nodes*, not compute *cores*. On the BG/P, this number is 64 compute nodes to 1 I/O node. Using more compute nodes allocates more I/O nodes available for t_{io}. Therefore, in these tests smp mode is used through 1024 cores; dual mode is used for 2048 cores (since the total number of nodes is 1024) and only 4096 cores employ vn mode.

Figure 7 compares the contribution to t_{frame} of each of t_{io}, t_{render}, and $t_{composite}$ for the same 864^3 dataset. At smaller numbers of processes, rendering time dominates the frame time, but I/O cost dominates at 1024 processes and beyond. This result underlines the need to further optimize parallel

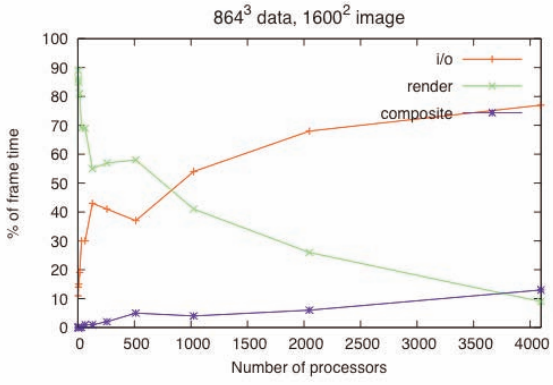

Figure 7: *Relative contribution to t_{frame} of each of t_{io}, t_{render}, and $t_{composite}$ is shown.*

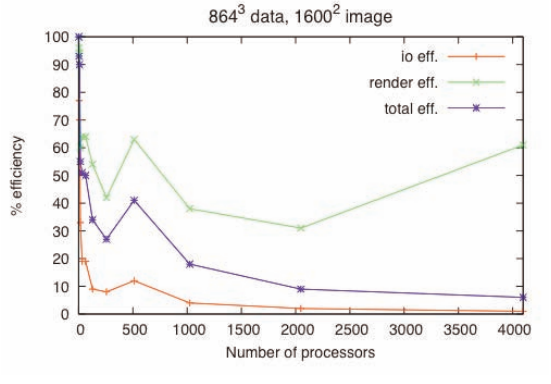

Figure 8: *Efficiency of t_{io}, t_{render}, and t_{frame} is plotted as a function of the number of processors.*

I/O operation on BG/P. Compositing time is still a relatively small fraction of the total time, reaching a maximum of 14% and usually less than 10%. However, Figure 7 clearly shows its relative contribution steadily increasing, and surpassing rendering time by 4096 cores. Hence, compositing time cannot be ignored indefinitely, especially if one expects to scale to tens of thousands of processes. Note that because Figure 7 displays relative percentages, features in one curve may be the result of another. For example, the dip in compositing contribution at 1024 cores is caused by an increase in I/O cost, not by a decrease in compositing itself.

Even when the parallel file system is optimized on BG/P, some configurations may be more efficient than others. For example, all of the curves in Figures 5-8 show an increase in both I/O and rendering performance at 512 cores. These may be functions of the underlying storage, computation, and communication architecture – things that we cannot change. However, a better understanding of the hardware may enable improved performance of visualization applications.

The ratio of speedup to the scaled increase in core count defines efficiency. Figure 8 shows efficiency of t_{io}, t_{render}, and t_{frame}. Compositing efficiency is not shown separately because compositing is a communication-bound operation. In an ideal setting, efficiency would remain near 100%: using n times as many processes should result in approximately n times the frame rate. Figure 8 tells quite a different story, and exposes the realities of both architecture and algorithm.

The upper curve, rendering efficiency, drops quickly but then remains at 30 – 60% throughout the experiment. Load imbalance between processes causes the drop from 100% to 60% between 2 and 16 cores. This is due to the static load distribution scheme that divides the dataset into uniform subvolumes irrespective of the actual rendering work to be done in each subvolume. For example, in this scheme it is possible for some subvolumes to have no data. Beyond that, the other poor performing locations are at 256, 1024, and 2048 cores. We hypothesize that cache coherence is worse at these configurations because of combinations of data size and cache size and we will be testing this further. The lower curve, I/O efficiency, decreases rapidly early on, and then slowly degrades further. Overall, I/O does not scale well yet on BG/P; at 4096 cores it is 6% compared to 2 cores. We expect this to improve in the near future. The middle curve is the efficiency of the total time, t_{frame}, and is principally an average of upper and lower curves.

The complete performance data for the 864^3 dataset and 1600^2 image appears in Table 1. These data include all three phases of the pipeline: I/O, rendering, and compositing. Sometimes, the I/O cost can be amortized over many frames, effectively hiding it. This is the case, for example, when multiple views of a single file or time step are visualized. We hope to similarly hide the I/O cost through prefetching multiple time steps of time-varying data in the future. Table 2 shows theoretical frame time assuming I/O cost can be completely hidden in this way.

5. Conclusions

We implemented a parallel ray-casting direct volume rendering algorithm on the IBM Blue Gene/P and tested performance over a large number of cores. In order to assess the viability of this architecture for large scale visualization, we intentionally chose to measure end-to-end frame time that includes not only classical visualization components such as rendering and compositing, but I/O time as well.

Our tests show that the Blue Gene architecture can be an appropriate platform for high-quality software visualization of large data. Its salient features with respect to this application are large numbers of tightly connected cores, a flexible programming model (MPI), a high-bandwidth connection to the parallel I/O system (MPI-IO and PVFS), and the ability to connect via sockets to remote displays. Software rendering cannot produce better performance than graphics clusters for small to medium-sized problems; but if current trends in data size [Mou04, JR07] continue, software volume rendering on massively parallel supercomputers may become a viable method in the future.

We believe that this approach will prove useful for data sizes of several gigavoxels in conjunction with image sizes of several megapixels. The method is also promising for in

Table 1: *Performance data for 864^3 dataset, 1600^2 image*

# Procs	t_{frame} (s)	t_{io} % of t_{frame}	t_{render} % of t_{frame}	$t_{composite}$ % of t_{frame}	t_{frame} % effcncy.
2	453.83	11.3	88.6	0.1	100.0
4	243.22	13.7	86.2	0.1	93.3
8	125.94	14.7	85.1	0.2	90.1
16	103.20	18.9	80.9	0.3	55.0
32	56.13	30.1	69.5	0.4	50.5
64	28.21	30.1	69.2	0.8	50.3
128	21.03	43.5	55.5	1.0	33.7
256	12.96	41.4	57.0	1.6	27.4
512	4.30	37.4	57.7	4.7	41.2
1024	5.01	54.3	41.3	4.4	17.7
2048	4.80	68.3	26.0	5.6	9.2
4096	3.41	77.4	9.4	13.2	6.5

Table 2: *Theoretical visualization performance assuming I/O costs are entirely hidden*

# Procs	t_{render} (s)	$t_{composite}$ (s)	vis. time = t_{render} + $t_{composite}$ (s)	vis. efficiency
2	401.94	0.4	402.34	100.00
4	209.56	0.32	209.88	95.85
8	107.15	0.3	107.45	93.61
16	83.47	0.27	83.74	60.06
32	39.01	0.24	39.25	64.07
64	19.51	0.22	19.73	63.73
128	11.67	0.21	11.88	52.92
256	7.39	0.21	7.60	41.36
512	2.48	0.2	2.68	58.64
1024	2.07	0.22	2.29	34.32
2048	1.25	0.27	1.52	25.85
4096	0.32	0.45	0.77	25.51

situ visualization [TYR*06], or in general when a very large dataset resides on the system already. As data sizes increase, transporting data between machines becomes nontrivial.

The relative cost of the three phases of the algorithm changes with the number of processes, although ultimately the application is I/O bound. Trade-offs exists between applying the correct number of cores to optimize I/O, rendering, and compositing, because these components of the total time trend in opposite directions and have various "sweet spots." It is unlikely that this method alone can effectively produce highly interactive performance, for example, 30 frames per second. More likely, its niche will be for very large data sets that cannot be accommodated by graphics clusters and can produce frame times on the order of a few seconds for such data.

Nonetheless, there is room for improvement. The parallel I/O system on BG/P will improve considerably over time – we know that it is not performing near capacity and work is ongoing in that regard. More sophisticated load-balancing techniques can improve the rendering efficiency, together with closer attention to cache and dual floating point pipeline usage. Compositing needs to be written with the communication backbone of the BG/P in mind.

When fully completed, BG/P will offer over one hundred thousand cores. This capacity can be leveraged by visualizing several frames through a multi-pipeline layout. Additional cores can also improve the quality of the rendering, for example to enable lighting and shading calculations. In the performance results, lighting was disabled; but Figure 1 shows that very high quality images can result through the addition of lighting.

6. Future Work

Our next tests will focus on scaling data size to gigavoxels and image size to megapixels and on improving image quality through lighting and shading. With 4 cores per node, BG/P offers the opportunity to experiment with multi-threading within an MPI process. This hybrid programming model may enable more efficient scaling, especially since the four cores share 2 GB of memory. This new level of parallelism can be exploited by modifying the rendering algorithm. We also are experimenting with tree-based compositing as a replacement for direct send. This may include binary swap [MPHK94] as a way to balance the number of messages with the size of a message and to keep more processes busy during the late stages of compositing.

We also plan to study how this research can be extended to encompass adaptive mesh refined (AMR) time-varying datasets [Ma99, WHH*01]. Varying levels of spatial resolution encoded in AMR data provide a compromise between the rigidity of completely structured data and the randomness of entirely unstructured data.

Another goal is to collate the performance data into a coherent model for predicting future performance. An open question is: what input criteria, such as processor speed, data size, number of processes, network bandwidth, memory bandwidth, and aggregate I/O throughput should be included in such a model. The result should be a relatively simple-to-use module that can analyze a parallel volume rendering problem and suggest an optimal configuration and predict its performance.

One of our long-term goals is to study how a supercomputer architecture can be used to support interactive rendering. The research so far has not included any elements of interactivity and performance data reveals that reaching interactive rates is difficult because of the tradeoffs between t_{io}, t_{render}, and $t_{composite}$. The next steps toward interactive rates may include LOD rendering as well as local view interpolation at the display machine(s). The ideal configuration may be the supercomputer and the graphics machine(s) sharing responsibilities in a client-server architecture.

Acknowledgments

We thank John Blondin and Anthony Mezzacappa for making their dataset available for this research. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. Work is also supported in part by NSF through grants CNS-0551727 and CCF-0325934, and DOE with agreement No. DE-FC02-06ER25777.

References

[IBM07] IBM Redbooks. http://www.redbooks.ibm.com/redpieces/abstracts/sg247287.html?Open 2007.

[SCI07] SciDAC Institute for Ultra-Scale Visualization. http://ultravis.ucdavis.edu/ 2007.

[BGM*07] BIDDISCOMBE, J., GEVECI, B., MARTIN, K., MORELAND, K. THOMPSON, D.: Time Dependent Processing in a Parallel Pipeline Architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13, 6, (October 2007), 1376-1383.

[BMD03] BLONDIN, J. M., MEZZACAPPA, A. DEMARINO, C.: Stability of Standing Accretion Shocks, with an Eye Toward Core Collapse Supernovae. *The Astrophysics Journal*, 584, 2, (2003), 971.

[CLRT00] CARNS, P., LIGON, W. B. I., ROSS, R. THAKUR, R.: PVFS: A Parallel File System for Linux Clusters. *Proceedings of 4th Annual Linux Showcase & Conference*, Atlanta, GA, (2000), 28.

[CMF05] CAVIN, X., MION, C. FIBOIS, A.: COTS Cluster-based Sort-last Rendering: Performance Evaluation and Pipelined Implementation. *Proceedings of IEEE Visualization 2005*, (October 2005), 111-118.

[CDM06] CHILDS, H., DUCHAINEAU, M. MA, K.-L.: A Scalable, Hybrid Scheme for Volume Rendering Massive Data Sets. *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2006*, Braga, Portugal, (May 2006), 153-162.

[CBB*05] CHILDS, H. R., BRUGGER, E. S., BONNELL, K. S., MEREDITH, J. S., MILLER, M. C., WHITLOCK, B. J. MAX, N. L.: A Contract Based System for Large Data Visualization. *Proceedings of IEEE Visualization 2005*, Minneapolis, MN, (October 2005), 190-198.

[DCH88] DREBIN, R. A., CARPENTER, L. HANRAHAN, P.: Volume Rendering. *ACM SIGGRAPH Computer Graphics*, 22, 4, (August 1988), 65-74.

[GGH*96] GEIST, A., GROPP, W., HUSS-LEDERMAN, S., LUMSDAINE, A., LUSK, E., SAPHIR, W. SKJELLUM, T.: MPI-2: Extending the Message-Passing Interface. *Proceedings of Euro-Par'96*, Lyon, France, (October 1996).

[Hsu93] HSU, W. M.: Segmented Ray Casting for Data Parallel Volume Rendering. *Proceedings of 1993 Parallel Rendering Symposium*, San Jose, CA, (1993), 7-14.

[JR07] JOHNSON, C. ROSS, R.: Visualization and Knowledge Discovery: Report from the DOE/ASCR Workshop on Visual Analysis and Data Exploration at Extreme Scale, 2007.

[LRN96] LEE, T.-Y., RAGHAVENDRA, C. S. NICHOLAS, J. B.: Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers. *IEEE Transactions on Visualization and Computer Graphics*, 2, 3, (September 1996), 202-217.

[Lev88] LEVOY, M.: Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8, 3, (May 1988), 29-37.

[Lev90] LEVOY, M.: Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9, 3, (July 1990), 245-261.

[Ma99] MA, K.-L.: Parallel Rendering of 3D AMR Data on the SGI/Cray T3E. *Proceedings of 7th Annual Symposium on the Frontiers of Massively Parallel Computation 1999*, Annapolis MD, (February 1999), 138-145.

[MC00] MA, K.-L. CAMP, D. M.: High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network. *Proceedings of Supercomputing 2000*, Dallas, TX, (November, 2000), 29.

[MI97] MA, K.-L. INTERRANTE, V.: Extracting Feature Lines from 3D Unstructured Grids. *Proceedings of IEEE Visualization 1997*, Phoenix, AZ, (October 1997), 285-292.

[MPHK93] MA, K.-L., PAINTER, J. S., HANSEN, C. D. KROGH, M. F.: A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering. *Proceedings of 1993 Parallel Rendering Symposium*, San Jose, CA, (October 1993), 15-22.

[MPHK94] MA, K.-L., PAINTER, J. S., HANSEN, C. D. KROGH, M. F.: Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications*, 14, 4, (July 1994), 59-68.

[MWY*07] MA, K.-L., WANG, C., YU, H. TIKHONOVA, A.: In-Situ Processing and Visualization for Ultrscale Simulations. *Journal of Physics*, 78, (June 2007).

[MMD06] MARCHESIN, S., MONGENET, C. DISCHLER, J.-M.: Dynamic Load Balancing for Parallel Volume Rendering. *Proceedings of Eurographics Symposium of Parallel Graphics and Visualization 2006*, Braga, Portugal, (May 2006)

[Max95] MAX, N. L.: Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1, 2, (June 1995), 99-108.

[MAF07] MORELAND, K., AVILA, L. FISK, L. A.: Parallel Unstructured Volume Rendering in ParaView. *Proceedings of IS&T SPIE Visualization and Data Analysis 2007*, San Jose, (January 2007).

[Mou04] MOUNT, R.: The Office of Science Data-Management Challenge. Report from the DOE Office of Science Data-Management Workshops, 2004.

[Neu94] NEUMANN, U.: Communication Costs for Parallel Volume-Rendering Algorithms. *IEEE Computer Graphics and Applications*, 14, 4, (July 1994), 49-58.

[Neu93] NEUMANN, U.: Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers. *Proceedings of 1993 Parallel Rendering Symposium*, San Jose, CA, (October 1993), 97-104.

[PPL*99] PARKER, S., PARKER, M., LIVNAT, Y., SLOAN, P.-P., HANSEN, C. D. SHIRLEY, P.: Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5, 3, (July 1999), 238-250.

[PD84] PORTER, T. DUFF, T.: Compositing Digital Images. *Proceedings of 11th Annual Conference on Computer Graphics and Interactive Techniques*, (1984), 253-259.

[SML*03] STOMPEL, A., MA, K.-L., LUM, E. B., AHRENS, J. PATCHETT, J.: SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. *Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, Seattle, WA, (October 2003), 33-40.

[TYR*06] TU, T., YU, H., RAMIREZ-GUZMAN, L., BIELAK, J., GHATTAS, O., MA, K.-L. O'HALLARON, D. R.: From Mesh Generation to Scientific Visualization: An End-to-end Approach to Parallel Supercomputing. *Proceedings of Supercomputing 2006*, Tampa, FL, (November 2006).

[WHH*01] WEBER, G. H., HAGEN, H., HAMANN, B., JOY, K. I., LIGOCKI, T. J., MA, K.-L. SHALF, J. M.: Visualization of Adaptive Mesh Refinement Data. *Proceedings of IS&T/SPIE Visual Data Exploration and Analysis VIII*, San Jose, CA, (2001), 121-132.

[YM05] YU, H. MA, K.-L.: A Study of I/O Methods for Parallel Visualization of Large-Scale Data. *Parallel Computing*, 31, 2, (February 2005), 167-183.

[YMW04] YU, H., MA, K.-L. WELLING, J.: A Parallel Visualization Pipeline for Terascale Earthquake Simulations. *Proceedings of Supercomputing 2004*, (November 2004), 49.