Eurographics Symposium on Parallel Graphics and Visualization (2014)
M. Amor and M. Hadwiger (Editors)

Performance Modeling of vl3 Volume Rendering on GPU-Based Clusters

S. Rizzi[1] M. Hereld[1] J. Insley[1] M.E. Papka[1,2] T. Uram[1] and V. Vishwanath[1]

[1]Argonne National Laboratory
[2]Northern Illinois University

Abstract

This paper presents an analytical model for parallel volume rendering of large datasets using GPU-based clusters. The model is focused on the parallel volume rendering and compositing stages and predicts their performance requiring only a few input parameters. We also present vl3, a novel parallel volume rendering framework for visualization of large datasets. Its performance is evaluated on a GPU-based cluster, weak and strong scaling are studied, and model predictions are validated with experimental results on up to 128 GPUs.

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

1. Introduction

Increasingly over the last few years, dedicated GPU-based computer clusters have been built to assist in visualization and analysis tasks, oftentimes sharing filesystems and network interconnections with large supercomputers. The visualization and analysis community has gathered significant knowledge and experience from building GPU clusters, yet a more comprehensive approach would be desirable to drive the design of these systems.

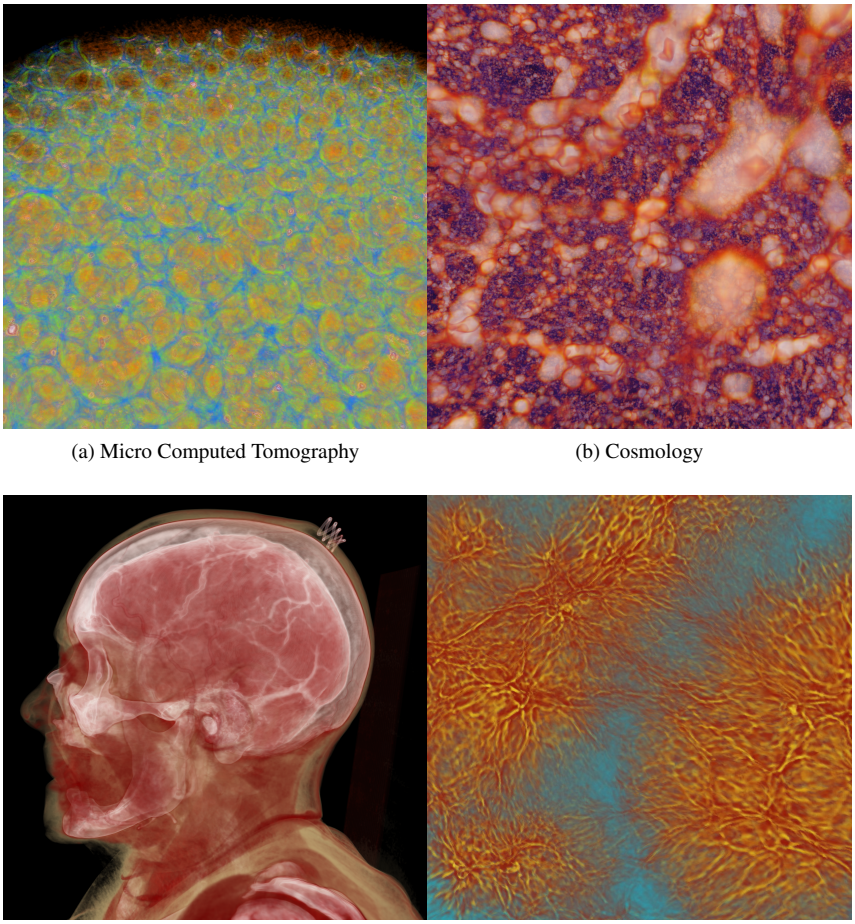
A prevalent application of GPU-based clusters is volume rendering of large datasets. Solutions to common problems in large-scale volume rendering, such as parallel rendering and compositing, are abundant in the literature. These advances are quickly assimilated by the visualization community in their open parallel visualization applications. Nonetheless, modeling specific stages of an application in concert with the hardware it will run on, along with their validation on existing machines, could become a powerful tool allowing us to make better design decisions in future systems.

The main contributions of this paper are the description of a model for parallel volume rendering on GPU-clusters and its validation with data collected from vl3, our scalable parallel GPU-based volume rendering application, running on a dedicated GPU cluster at the Argonne Leadership Computing Facility.

2. Related Work

Parallelizing volume rendering is a strategy used to both speedup the rendering process and render larger data. This introduces a compositing step to volume rendering, which can be time consuming on large systems, so different algorithms have focused on optimizing it. Compositing is usually parallelized by dividing up the data space and/or dividing up the image space. Sort-first techniques divide the output image among processes, while sort-last approaches divide up the data space. Sort-first is more common in shared memory systems [PTT98] than distributed memory systems [BIPS00], because with shared memory systems, memory and data are available to all processes. On distributed memory systems, sort-last techniques are often employed, using object decomposition to more efficiently use available memory by giving each node a subset of the full dataset. Sort-last is described in the original DirectSend [Hsu93] algorithm, where local sub-blocks are rendered and the over operator [PD84] is used to combine the resulting 2D renderings into a final image. The BinarySwap [MPHK94] algorithm introduced a tree-based compositing algorithm and used image space decomposition. Algorithms related to BinarySwap have been developed to improve the BinarySwap algorithm and apply it to different systems [ISTH03], [PYRM08]. A parallel implementation of DirectSend has been described in [EP07], where its performance was compared to the serial implementation and the BinarySwap algorithm. Other ad-

DOI: 10.2312/pgv.20141086



(a) Micro Computed Tomography (b) Cosmology

(c) Computed Tomography (d) Astrophysics

Figure 1: *Volume rendering applied in different domains. Images generated using vl3 parallel volume rendering on GPU-based clusters at the Argonne Leadership Computing Facility.*

vances in large-scale compositing algorithms are described in [MKPH11] and implemented in the IceT library from Sandia National Laboratories.

Parallel volume rendering provides the opportunity to load larger datasets whenever additional hardware is available. [CDM06] explored rendering larger datasets of 3000^3 voxels using hundreds of cores. [PYRM08], [PYR*09] investigated loading large datasets up to 4480^3 voxels, with tens of thousands of cores. [HBC10] analyzed the benefits of combined process and thread parallelism at 200,000 cores.

Volume rendering was first presented on commodity GPU clusters by [MHE01] and [BPT02]. Since then, additional algorithmic optimizations have been researched for GPUs [SMW*04], [WGS04]. Taking advantage of programmable graphics pipelines, ray casting volume rendering on the GPU was first described in [KW03]. Expanding on the subject, an entire chapter was devoted to GPU-based ray casting in [HKRs*06], including an example of fragment shader code for single-pass ray casting and a number of possible optimizations. In 2005, the FlowVR framework [AR05] used sort-first cluster volume to render a 512^3 voxel dataset to a tiled display. More recently, MapReduce was applied to distributed GPU volume rendering [SCMO10] scaling up to 32 GPUs and 1024^3 voxels. In [FCS*10], extensions to Visit for multi-gpu rendering were presented and performance was reported on the Lens and Longhorn GPU clusters, scaling up to 256 GPUs. Not directly involving GPUs, but relevant to the problems of scalable parallel volume rendering and compositing, [MRG*08] reported on ParaView scalability for parallel volume rendering with up to 512 processes running on a Cray XT3 supercomputer.

3. vl3 Parallel GPU Volume Rendering Architecture

vl3, a parallel GPU-based volume rendering application developed at Argonne National Laboratory and the Computation Institute of the University of Chicago, was used to collect experimental data in this work. Its architecture is driven by the parallel volume rendering process, with a modular design that enables its various components to be easily swapped out. This facilitates the development and testing of new methods and algorithms, as well as easily configuring vl3 based on the hardware it is run on. vl3 has been used to generate production level visualizations of data from different disciplines, as illustrated in Figure 1. The functions in vl3 are broken down into the following categories:

Data Input: This module is responsible for retrieving data from various sources, such as parallel file systems and networks, and presenting the data to the next stage of the pipeline via a common interface. vl3 implements loaders for several 3D data formats, providing a plugin architecture to allow new loaders to be written. The system supports common data formats, such as raw/binary, Dicom, and VTK, among others.

Rendering: vl3 parallelizes the volume rendering process by spatially decomposing the 3D volume into smaller equal-sized chunks (including ghost cells), and distributing them among multiple nodes of a cluster computer. Rendering of regular grids and Adaptive Mesh Refinement (AMR) data are supported [LVI*13]. Ray casting volume rendering is implemented using OpenGL and the OpenGL Shading Language (GLSL). The core rendering code is executed as GLSL fragment shaders, typically casting one ray per output pixel, and sampling the 3D texture along the ray to obtain the pixel final color. Additionally, GLSL allows the system to run code on GPU hardware or in software using the Mesa 3D library.

Compositing: When subvolumes are rendered by separate compute nodes, the result is a number of 2D images that need to be composited together into the final image. To do this, the images must be blended together in front-to-back order, with the latter determined by the camera location and a spatial representation of the subvolumes. Since the GPUs are typically on separate machines, both the network communication and the actual blending time need to be considered to determine the system performance. The simplest algorithm for compositing is the serial DirectSend algorithm, where all the rendering nodes send their final 2D images to a single machine, which blends them into a single final image. There is also parallel DirectSend, where the image space is partitioned among N processes, each responsible for blending over a subset of the full image. The modular architecture of vl3 makes it simple to switch among these or other compositing algorithms.

Output: Results must be displayed once the volume rendering and compositing steps are complete. vl3 supports multiple modalities, including batch mode, interactive visualization, and streaming. In batch mode, one or more image snapshots are written to files, and can be used later to generate videos. Another option is to run vl3 interactively on a local desktop or laptop computer with a discrete GPU, where images are viewed at interactive frame rates. More commonly when using larger datasets, vl3 runs as a Message Passing Interface (MPI) application on a GPU-cluster. In this case, vl3 can stream large pixel count images to high resolution displays such as 4K monitors or tiled displays [HIO*11].

4. Performance Model

In the scope of this paper we are mainly interested in estimating the impact of rendering and compositing times on the system performance, without considering storage access times. This simplification assumes that datasets fit into the combined CPU memory of all processors. Therefore, there is no need to consider intermediate accesses to storage during rendering and compositing. For similar considerations, we will ignore the time it takes to load the dataset from storage into CPU memory, and the time to either display or store the final renderings. Consequently, the total time to render a frame will be modeled solely by the individual contributions of GPU volume rendering and compositing times as:

$$T_{Total} = T_{Render} + T_{Compositing} \quad (1)$$

4.1. Volume Rendering

In GPU-based volume rendering using ray casting, there are usually three steps clearly differentiated: (i) the raw data is first loaded from the CPU memory onto the GPU memory; (ii) data is then rendered on the GPU by casting rays into the volume; and (iii) the rendered pixels are read back from the GPU to the CPU. The time to render a volume is thus given by:

$$T_{Render} = T_{LoadGPU} + T_{GPURender} + T_{ReadBackGPU} \quad (2)$$

Using a simple block division wherein the total data volume V_{Total} is distributed equally among N ranks, the volume in voxels on each rank is:

$$V_{rank} = \frac{V_{Total}}{N}$$

where there is one GPU per rank. The overhead added by ghost cells is negligible in large datasets, thus they have not been considered in this analysis.

A volume V_{rank} is present on the CPU memory and must be first transferred to the GPU over the system bus with bandwidth $BW_{SystemBus}$. Thus, the time to load the entire data volume from the CPU memory to the GPU memory on N

ranks is given by:

$$T_{LoadGPU} = \frac{V_{rank} \times d_{voxel}}{BW_{SystemBus}} \quad (3)$$

where d_{voxel} is the size in bytes for each voxel.

A GPU can render $Frag_{GPU}$ pixels at a time, where $Frag_{GPU}$ is the number of fragment shaders. Thus, a GPU needs to perform a number of iterations to render its sub-volume, given by:

$$Render_{Itr} = \frac{I_{rank}}{Frag_{GPU}}$$

where I_{rank} is the total number of bytes in the partial image generated by each GPU. As an example, if the output image is 4096×4096 pixels and there are 2 rendering GPUs, I_{rank} is $4096 \times 4096/2 = 8388608$ bytes.

Therefore, the time to ray cast on the GPU can be approximated by:

$$T_{GPURender} = Render_{Itr} \times Samples_{Ray} \times T_{ProcessSample} \quad (4)$$

where $T_{ProcessSample}$ is the time it takes to process an individual sample on the GPU and $Samples_{Ray}$ is the number of samples to be evaluated along the trajectory of the ray.

After rendering, pixels need to be read back to CPU memory. Therefore:

$$T_{ReadBackGPU} = \frac{I_{rank} \times d_{pixel}}{BW_{SystemBus}} \quad (5)$$

where d_{pixel} is the size in bytes for each pixel (e.g. four bytes for RGBA values).

Taking into account the three components in Equation 2, the total volume rendering time becomes:

$$\begin{aligned} T_{Render} = & \frac{V_{rank} \times d_{voxel}}{BW_{SystemBus}} \\ & + Render_{Itr} \times Samples_{Ray} \times T_{ProcessSample} \\ & + \frac{I_{rank} \times d_{pixel}}{BW_{SystemBus}} \end{aligned} \quad (6)$$

Note that all parameters in this equation are either known from operating conditions or system specifications, or can be obtained experimentally.

4.2. Compositing

The compositing stage takes partial images from individual rendering ranks and blends them adequately to create the final image. The total compositing time is given by:

$$T_{Compositing} = T_{NetworkTransfer} + T_{GPUCompositing} \quad (7)$$

In the following subsections we consider two compositing algorithms, serial Direct-send and its parallel counterpart.

4.2.1. Serial Direct-send Compositing

In the serial algorithm, a single node is used to composite the various rendered images. If BW_{Node} is the maximum achievable network throughput of the compositing node and s is the one-way latency, the total transmission time for $N-1$ concurrent transfers from rendering GPUs is given by:

$$T_{NetworkTransfer} = s + [(N-1) \times \frac{I_{rank} \times d_{pixel}}{BW_{Node}}] \quad (8)$$

Once the compositing node has received all sub-images from rendering nodes, they have to be loaded into the GPU, blended, and read back. The image size loaded to the GPU on the compositor is I_{rank}, therefore

$$T_{LoadToGPU} = \frac{I_{rank} \times d_{pixel}}{BW_{SystemBus}} \quad (9)$$

Compositing involves iterating over the sub-images sorted in front-to-back order and blending them two at a time. Since a GPU can blend $Frag_{GPU}$ pixels at a time, a number of iterations are needed, given by:

$$SerialC_{Itr} = \frac{I_{rank}}{Frag_{GPU}}$$

while the time to read back from GPU is given by:

$$T_{ReadBackfromGPU} = \frac{I_{Output} \times d_{pixel}}{BW_{SystemBus}} \quad (10)$$

where I_{Output} represents the desired size of the output image (1920×1080 pixels, for example).

With that, the total time for serial compositing is:

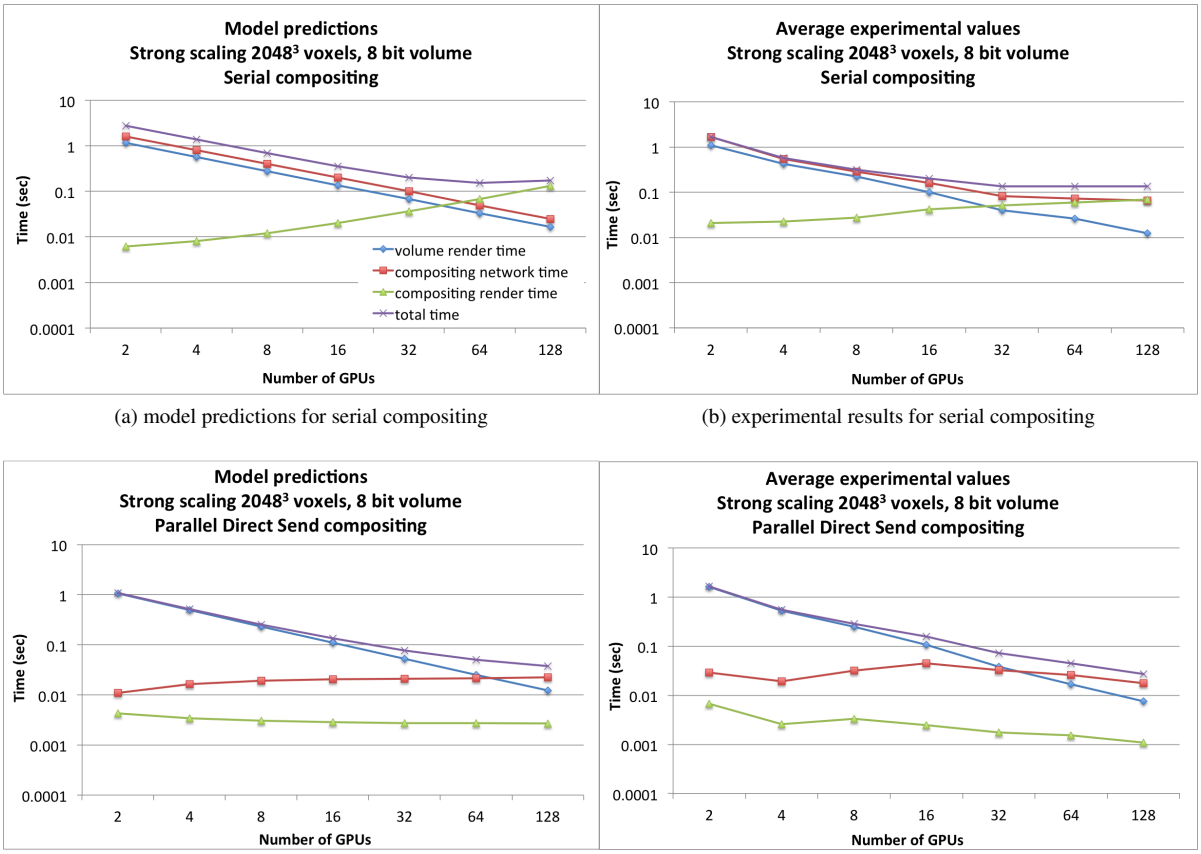
$$\begin{aligned} T_{SerialGPUCompositing} = N \times \Big[& T_{LoadToGPU} + T_{ReadBackfromGPU} \\ & + SerialC_{Itr} \times T_{BlendImages} \Big] \end{aligned} \quad (11)$$

where $T_{BlendImages}$ is the time it takes to blend two pixels on the GPU. Therefore, equation 7 will give the total serial compositing time as

$$\begin{aligned} T_{SerialCompositing} = & s + [(N-1) \times \frac{I_{rank} \times d_{pixel}}{BW_{Node}}] \\ & + N \times \Big[\frac{(I_{rank} + I_{Output}) \times d_{pixel}}{BW_{SystemBus}} \\ & + SerialC_{Itr} \times T_{BlendImages} \Big] \end{aligned} \quad (12)$$

4.2.2. Parallel Direct-send compositing

In case of parallel compositing, the amount of data composited on each rank is I_{rank} consisting of N image buffers in the worst case. The output image, I_{output} is composited across N ranks, with each rank producing a portion I_{Output}/N of the final image. Hence, we effectively parallelize the compositing computation at the added overhead of additional communication.



(a) model predictions for serial compositing

(b) experimental results for serial compositing

(c) model predictions for parallel compositing

(d) experimental results for parallel compositing

Figure 2: *Strong scaling using the serial and parallel Direct Send compositing algorithms*

As there are now N compositing ranks, the network transfer time becomes

$$T_{Network_{Parallel}} = [(N-1) \times s] + [(N-1) \times \frac{I_{rank} \times d_{pixel}}{N} \times \frac{1}{BW_{Node}}] \quad (13)$$

Since every compositing rank now has smaller size images to composite, the times to load and read back an image to and from the GPU become

$$T_{LoadToGPUParallel} = \frac{I_{rank} \times d_{pixel}}{N \times BW_{SystemBus}} \quad (14)$$

$$T_{ReadBackfromGPUParallel} = \frac{I_{Output} \times d_{pixel}}{N \times BW_{SystemBus}} \quad (15)$$

The number of blending iterations is also reduced by N

$$ParallelC_{Itr} = \frac{I_{rank}/N}{Frag_{GPU}}$$

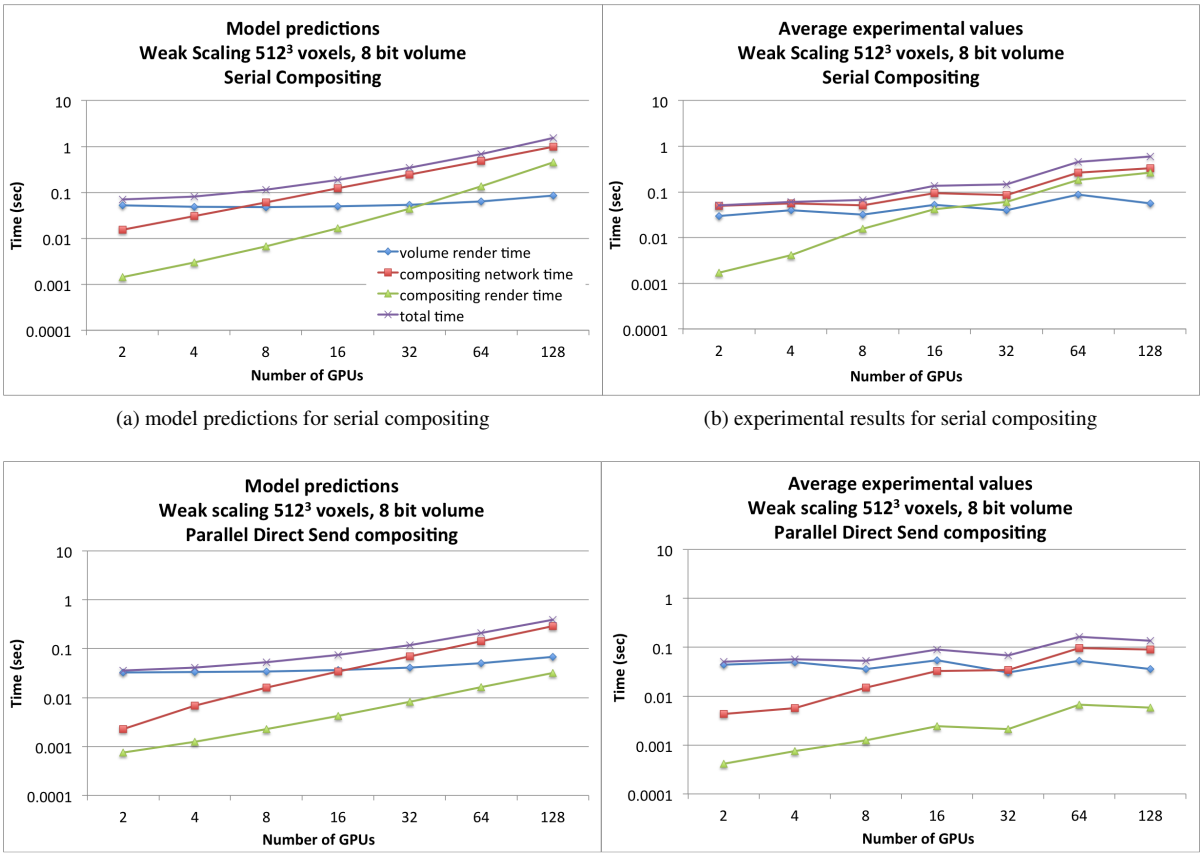
From that, the total time for parallel compositing is

$$T_{ParallelCompositing} = [(N-1) \times s] + [(N-1) \times \frac{I_{rank} \times d_{pixel}}{N} \times \frac{1}{BW_{Node}}] + N \times \left[\frac{(I_{rank} + I_{Output}) \times d_{pixel}}{N \times BW_{SystemBus}} + ParallelC_{Itr} \times T_{BlendImages}\right] \quad (16)$$

5. Evaluation

The performance of parallel volume rendering on GPU-based clusters is evaluated as the number of GPUs and the total volume data rendered are increased. The model is used to predict performance, as well as to determine the overall influence of the volume rendering and compositing components.

Our experimental testbed consists of Tukey, a computer cluster at Argonne National Laboratory built with multi-core



(a) model predictions for serial compositing

(b) experimental results for serial compositing

(c) model predictions for parallel compositing

(d) experimental results for parallel compositing

Figure 3: *Weak scaling using the serial and parallel Direct Send compositing algorithms*

X86 computers. There are 96 compute nodes based on the AMD Dual Opteron 6128 processor, with 16 cores per node. Every node also has 64 GB RAM and two Nvidia Tesla M2070 GPUs, adding up to 6 terabytes of CPU RAM and 1.1 terabytes of GPU RAM for the entire system. The peak GPU performance for the system is estimated at over 98 teraflop. The nodes are connected via a QDR Infiniband interconnect and the MPI implementation is Mvapich2 [MVA] from Ohio State University (OSU).

5.1. Strong scaling performance

In strong scaling we are interested in evaluating how the system performance changes as the problem size is preserved while more GPUs are used to obtain the solution. In this case, we use a synthetic dataset consisting of 2048^3 voxels with 8 bit per voxel and scaling to up to 128 GPUs. The resolution of the output image is 2048×2048 pixels and every rendering node samples each voxel at least once during the ray casting process.

Parameters for our models are empirically obtained from profiling tools such as the bandwidth test included in the Nvidia GPU Computing SDK [GPU] and the OSU microbenchmarks [MVA] for network latency and bandwidth. Using the model for volume rendering with serial and parallel DirectSend composition we obtain the curves shown in Figures 2a and 2c. In our experiments, there is one MPI rank per GPU, for which individual profiling information is collected for a thousand iterations. Execution times are shown averaged in figures 2b and 2d for serial and parallel DirectSend compositing, respectively.

For serial compositing we observe good agreement for volume rendering and compositing rendering times (blue and green curves, respectively). Using the RMS error in table 1 as a metric to quantify the deviations between prediction and experiment, the worst match is for compositing network time, which agrees with the deviations observed between both red lines in figures 2a and 2b. In addition, the serial compositing model predicts a slight increase of total time for a large number of GPUs. Similarly, a flattening of

the total time is observed in experiments, but this deviation is mainly attributed to the inherent reduction in network bandwidth when packet sizes decrease as a consequence of the increase of the number of GPUs. In case of parallel DirectSend compositing, there is good agreement for compositing network and rendering times in terms of RMS error (red and green curves in figures 2c and 2d). The worst match is volume rendering time (blue curves), though the trend observed in the experiments is still nicely captured by the model. As expected, volume rendering time (blue) is comparable in experimental results for serial and parallel DirectSend cases. More importantly, experiments confirm that the overall performance and scaling for parallel DirectSend is significantly better, by almost an order of magnitude, than the serial algorithm as the number of GPUs increases.

5.2. Weak Scaling

Figures 3a and 3c depict the model prediction for weak scaling using serial and parallel DirectSend compositing, wherein we keep the load per GPU constant as the total number of GPUs is increased. In this case we use a volume size of 512^3 voxels per GPU. Every time we double the number of GPUs, we also double the total number of voxels and the total number of pixels in the resulting image. For example, for 2 GPUs, the total volume size is 645^3 voxels, which divided by 2 results in 512^3 voxels per GPU. Similarly, the image size for 2 GPUs is 1024×512 pixels, preserving the ratio of 512^2 pixels per GPU. As we scale from 2 GPUs to 128 GPUs we reach a total volume size of 2560^3 voxels and an image size of 8192×4096 pixels. As expected, the predicted volume rendering time remains relatively flat, since every GPU keeps the same rendering load. On the other hand, for both compositing algorithms the compositing network time starts to dominate for larger values of the number of GPUs used. Experimental results for the same problem configurations are shown in figures 3b and 3d, where once again we see the network compositing time dominating the overall frame time. We also observe good agreement with the behavior predicted by the model, where the total frame time increases for an increasing number of GPUs due to the influence of the network transfer time in the compositing stage.

In this case, the worst RMS match is observed for compositing render time (green) for serial and parallel DirectSend compositing. Also, volume rendering time (blue) is similar in figures 3a and 3c. In general, the model captures fairly well the trends for the three main components. As in the strong scaling case, parallel DirectSend outperforms and shows better weak scalability than serial compositing in both model and experiment.

6. Conclusion and Future Work

In this paper we have proposed a simple model for the ray casting and compositing stages of volume rendering on GPU-based clusters. Predictions are supported by experimental data, proving that the model is reasonably close to reality.

	Strong Scaling		Weak Scaling	
	Serial	Paral.	Serial	Paral.
Vol.Rend. (blue)	0.065	**0.210**	0.019	0.016
Comp.Net. (red)	**0.108**	0.014	**0.266**	**0.078**
Comp.Rend(grn)	0.028	0.001	0.075	0.011
TotalTime (purp)	0.528	0.221	0.368	0.099
Refer to figures	2a 2b	2c 2d	3a 3b	3c 3d

Table 1: *RMS error between predicted and average experimental values for serial and parallel DirectSend for strong and weak scaling. Numbers in bold show worst cases. Total Time is shown but not considered for worst case.*

Results show that compositing network time is the dominant component of the total time when the number of GPUs increases, with mainly two possible factors for explaining the observations: (i) network contention in case of serial compositing; (ii) bandwidth decrease for lower packet sizes as a consequence of an increased number of GPUs. This could be a crucial factor in designing future network interconnects, where a sustained bandwidth is desired for all packet sizes.

We have shown scalability of vl3 for up to 128 GPUs and 2560^3 voxels. vl3 scalability, in terms of volume size, is determined by the number of GPUs used and their internal memory size. In production jobs we have scaled vl3 up to 8192^3 voxels with 128 GPUs and output resolutions of 6144×3072 pixels. Its exceptional adaptability, where GLSL volume renderers and compositors could be replaced with different approaches (i.e. CUDA on GPU-based systems with no graphics drivers, OpenMP and OpenACC on CPU-based systems), make vl3 an excellent candidate for volume rendering in future architectures.

The model presented here could be adapted by other research groups to evaluate the efficiency of different volume rendering components, providing reasonable starting points to optimize volume rendering performance. It could also allow them to analyze and predict performance of volume rendering on different GPU clusters. This could be invaluably helpful in the design of upcoming GPU clusters to avoid bottlenecks and produce the most powerful systems at a given cost.

In future work we will extend the model to account for optimizations such as compression, asynchronous transfers to/from GPU, and asynchronous network communication.

Acknowledgement

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Depart-

ment of Energy, under Contract DE-AC02-06CH11357 including the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization. This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

The authors would like to thank the HACC cosmology team at Argonne National Laboratory, the Advanced Photon Source at Argonne National Laboratory, the University of Chicago, and the San Diego Supercomputer Center for providing the data sets used to generate the images in Figure 1.

References

[AR05] ALLARD J., RAFFIN B.: A shader-based parallel rendering framework. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 127–134. 2

[BIPS00] BAJAJ C., IHM I., PARK S., SONG D.: Compression-based ray casting of very large volume data in distributed environments. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on* (2000), vol. 2, IEEE, pp. 720–725. 1

[BPT02] BAJAJ C., PARK S., THANE A.: Parallel multi-pc volume rendering system. *CS & ICES Technical Report, University of Texas at Austin 2* (2002). 2

[CDM06] CHILDS H., DUCHAINEAU M. A., MA K.-L.: A scalable, hybrid scheme for volume rendering massive data sets. In *Eurographics Symposium on Parallel Graphics and Visualization* (2006), pp. 153–162. 2

[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* (2007), EG PGV'07, pp. 29–36. 1

[FCS*10] FOGAL T., CHILDS H., SHANKAR S., KRÜGER J., BERGERON R. D., HATCHER P.: Large data visualization on distributed memory multi-gpu clusters. In *Proceedings of the Conference on High Performance Graphics* (2010), Eurographics Association, pp. 57–66. 2

[GPU] Nvidia gpu computing sdk. http://developer.nvidia.com/gpu-computing-sdk. Accessed: 2014-03-03. 6

[HBC10] HOWISON M., BETHEL E. W., CHILDS H.: MPI-hybrid Parallelism for Volume Rendering on Large, Multi-core Systems. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (Norrköping, Sweden, May 2010). LBNL-3297E. 2

[HIO*11] HERELD M., INSLEY J. A., OLSON E. C., PAPKA M. E., VISHWANATH V., NORMAN M. L., WAGNER R.: Exploring large data over wide area networks. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on* (2011), IEEE, pp. 133–134. 3

[HKRs*06] HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D., ENGEL K.: *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. 2

[Hsu93] HSU W. M.: Segmented ray casting for data parallel volume rendering. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering* (1993), pp. 7–14. 1

[ISTH03] INO F., SASAKI T., TAKEUCHI A., HAGIHARA K.: A divided-screenwise hierarchical compositing for sort-last parallel volume rendering. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* (2003), IEEE, pp. 10– pp. 1

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (2003), IEEE Computer Society, p. 38. 2

[LVI*13] LEAF N., VISHWANATH V., INSLEY J., HERELD M., PAPKA M. E., MA K.-L.: Efficient parallel volume rendering of large-scale adaptive mesh refinement data. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on* (2013), IEEE, pp. 35–42. 3

[MHE01] MAGALLÓN M., HOPF M., ERTL T.: Parallel volume rendering using pc graphics hardware. In *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on* (2001), IEEE, pp. 384–389. 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* (2011), SC '11, pp. 25:1–25:10. 2

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl. 14*, 4 (1994), 59–68. 1

[MRG*08] MORELAND K., ROGERS D., GREENFIELD J., GEVECI B., MARION P., NEUNDORF A., ESCHENBERG K.: Large scale visualization on the cray xt3 using paraview. *Cray User Group* (2008). 2

[MVA] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE, Network-Based Computing Laboratory, Department of Computer Science and Engineering, Ohio State University. http://mvapich.cse.ohio-state.edu/. Accessed: 2014-03-03. 6

[PD84] PORTER T., DUFF T.: Compositing digital images. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 253–259. 1

[PTT98] PALMER M., TOTTY B., TAYLOR S.: Ray casting on shared-memory architectures: memory-hierarchy considerations in volume rendering. *Concurrency, IEEE 6*, 1 (jan. 1998), 20 –35. 1

[PYR*09] PETERKA T., YU H., ROSS R., MA K.-L., LATHAM R.: End-to-end study of parallel volume rendering on the ibm blue gene/p. In *ICPP '09: Proceedings of the 2009 International Conference on Parallel Processing* (2009), pp. 566–573. 2

[PYRM08] PETERKA T., YU H., ROSS R., MA K.-L.: Parallel volume rendering on the ibm blue gene/p. In *Proceedings of Eurographics Parallel Graphics and Visualization Symposium (EGPGV 2008)* (April 2008), pp. 73–80. 1, 2

[SCMO10] STUART J. A., CHEN C.-K., MA K.-L., OWENS J. D.: Multi-gpu volume rendering using mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (2010), ACM, pp. 841–848. 2

[SMW*04] STRENGERT M., MAGALLÓN M., WEISKOPF D., GUTHE S., ERTL T.: Hierarchical visualization and compression of large volume datasets using gpu clusters. In *Proceedings of the 5th Eurographics conference on Parallel Graphics and Visualization* (2004), Eurographics Association, pp. 41–48. 2

[WGS04] WANG C., GAO J., SHEN H.-W.: Parallel multiresolution volume rendering of large data sets with error-guided load balancing. In *Proceedings of the 5th Eurographics conference on Parallel Graphics and Visualization* (2004), Eurographics Association, pp. 23–30. 2