A Static Load Balancing Scheme for Parallel Volume Rendering on Multi-GPU Clusters

Shusen Liu*[1], Venkatram Vishwanath †[2], Joseph Insley ‡[2], Mark Hereld §[2], Michael E. Papka ¶[2,3], and Valerio Pascucci ‖[1]

[1]Scientific Computing and Imaging Institute, University of Utah
[2]Argonne National Laboratory
[3]Northern Illinois University

ABSTRACT

GPU-based clusters are an attractive option for parallel volume rendering. One of the key issues in parallel volume rendering is load balancing, keeping a balanced workload per node is essential for improving performance. A good number of dynamic load balancing schemes have been proposed throughout the years. However, most of these approaches require runtime dynamic data movement or data duplication. For the large datasets routinely generated by scientific applications, frequent data transfer can be prohibitively expensive. In this work, we propose a static load balancing scheme. By optimizing data placement, a balanced workload can be achieved with minimal or no data movement, therefore improving the rendering speed and user experience.

Keywords: Parallel Volume Visualization, Load Balancing

1 INTRODUCTION

With the ever increasing amount of available computing resources scientific simulations are able to represent highly complex phenomena at unprecedented scale and resolution. This also introduces tremendous challenges for understanding and analyzing extremely large datasets. Parallelization of visualization algorithms is one of the most effective approaches to meet the challenges. Volume rendering as one of the standard visualization algorithms is our focus. Most parallel volume rendering algorithms can be divide into two major categories: Sort First (screen space task partition) and Sort Last(object space task partition). For large scale datasets, object space partition is usually employed. Screen space task partition innately requires a globally shared memory access pattern which is undesirable for extremely large datasets. Sort last algorithms divide the problem domain in 3D space and assign part of the dataset to different nodes (machines). The final image is generated by sorting the fragments produced by each node and compositing them using alpha blending. The overall rendering speed will be determined by the slowest node, so an equally distributed workload is essential for good performance.

In volume rendering, the rendering speed is a function of camera parameters (position and orientation), so during an interactive visualization session the workload on each node will be varying. The well known load balancing scheme is based on tree-like data partitioning and dynamic data reassignment. During runtime, it dynamically reassigns the workload based on the performance measure for each leaf node. This scheme works best under shared memory architectures since data reassignment requires no copy operation. However, for GPU clusters and supercomputers data movement is a critical component. As the dataset size grows rapidly, it is increasingly expensive, in terms of energy, to shuffle data between nodes.

In this work, we describe a novel static load balancing scheme based on assigning multiple blocks (a block is a 3D rectangular domain containing a small subset of the entire volume) per node and optimizing the global data layout. Our static data layout scheme greatly reduces the correlation between per node workload with camera position and angle, therefore dramatically improving the load balancing without data reassignment.

In particular, we make the following contributions:

- Introduce the interleaved multi-block layout scheme and a compatible GPU-based compositing mechanism.
- Provide a procedure to generate block layouts that are load balanced.
- Achieve almost near optimal load balance without any data movement or data duplication.

2 RELATED WORKS

Parallel volume rendering has been an active research area for decades. However, new challenges continue to motivate researchers to re-evaluate established techniques. Most methods for handling large scale volumetric data follow the sort-last architecture, where data is distributed among the nodes [7]. The images generated by each node are then composited by variations of the following algorithms: direct send approach by Neumann [6] and binary-swap algorithm [3]. Several studies have investigated load balancing, including Muller et al. [5] and Marchesin et al. [4], employ a kd-tree for dynamically reassigning the data in a cluster. Marchesin et al. [4] found that changing the view (zooming in on parts of the data, for example) leads to a significant load imbalance. This is very challenging and critical for interactive parallel rendering. Our proposed algorithm is intended to address this while eliminating the data transfer overhead exists in other dynamic load balancing schemes.

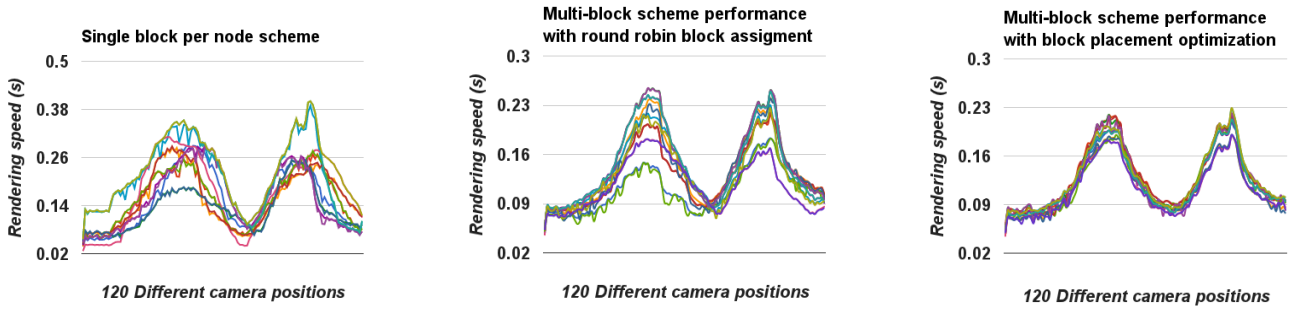
3 DESIGN

In our work, the data is divided into much finer blocks in comparison to prior efforts. Each node renders a selected group of these blocks, which span the entire domain. By re-arranging the blocks intelligently, we can potentially make every node have a similar workload profile even when camera parameters vary.

Block based data partitioning is commonly used in volume rendering for out of core rendering, kd-tree based load balancing, empty space skipping, etc. In our approach, the blocks rendered by one particular node are interleaved with blocks from other nodes. This means we cannot generate a simple image per node for the compositing phase. Instead every small image buffer rendered from each individual block will need to be stored. This would potentially increase the compositing time. However, for the GPU-cluster based volume renderer the compositing time is very small compared to the rendering time (less than 10% [2]). Also, by using GPUs for compositing, the compositing phase takes only around 6-7% of the total rendering time for our test dataset.

The most straight forward implementation of multi-block rendering is to render one block, read back the image, and then repeat

*e-mail: shusenl@sci.utah.edu
†e-mail:venkat@anl.gov
‡e-mail:insley@anl.gov
§e-mail:hereld@mcs.anl.gov
¶e-mail: papka@anl.gov
‖e-mail: pascucci@sci.utah.edu



(a) (b) (c)

Figure 1: How balanced is each approach? (a) single block per node, (b) multi-block per node, (c) multi-block per node with block layout optimization. In both (b) and (c) the block size is 256. The x-axis corresponds to the different camera positions, while the y-axis depicts the rendering time. Colored line represent different node in the cluster. Unaligned lines means unbalanced load while overlapped lines show perfectly balanced load.

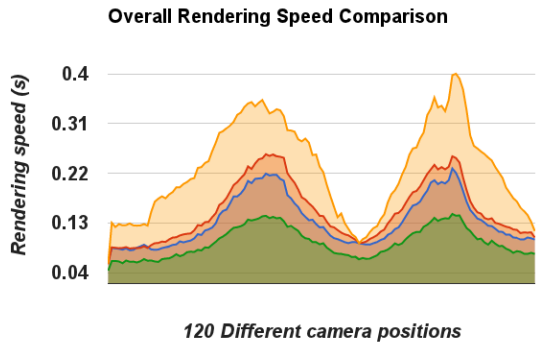


Figure 2: Here we provide the performance different between several test setups. The yellow line is the single block scheme, the red line is the multi-block scheme (256 block size) with simple round robin, the blue and green line is the multi-block scheme with blocks layout optimization(with 256 and 512 block size).

this for all the blocks. However, launching large numbers of rendering passes with very low computing load is a very inefficient way to use current generation GPUs. We prefer to render all blocks in one single pass. In our implementation, single pass interleaved multi-block rendering is achieved by utilizing scatter write. CUDA is utilized for its efficiency and ease of use in scatter write.

We evaluate with two mechanisms to generate the interleaved block layout. In the first, we map the block location to a "linear" index $(z \times dimensionX \times dimensionY + y \times dimensionX + x)$, and assign one of every N blocks, where N is the number of nodes, to each node (simple round-robin). This fails to produce a very balanced load, since it doesn't intentionally take spatial locality into consideration. Since our goal is to assign the nearby blocks to different nodes, a z-order index will ensure blocks with similar indices are spatially close. By replacing the "linear" index with the z-order [1] index, the load balance result is improved. A detailed comparison between single block method and these approaches is depicted in the results section.

4 Preliminary Results

We evaluate our approach on the GPU cluster "Tukey" at Argonne National Laboratory. Each Tukey node is equipped with 2 Tesla M2070 GPUs, and the test volume is a $2048 \times 2048 \times 2048$ regular grid (8GB - unsigned char). In order to measure the load balancing performance of different approaches, we measured the rendering time for a number of unique views on every node - a 280 degree camera rotation, with two zooming operations along the camera path, which correspond to the two peaks seen in the figures. In Figure 1, we compare the single block scheme with our proposed multi-block schemes. It clearly demonstrates the efficacy of our method in achieving static load balancing under drastically changing views. From (a) to (c), we gain better load balance by introducing multi-blocks per node and by reorganizing the multi-block block layout pattern. In Figure 2, we notice that improving the load balancing dramatically improves the rendering performance. One drawback of the multi-block per node scheme could also be observed in figure 2. Performance improves when using a block of size 512 instead of 256. This is mainly due to rendering multiple blocks. However by using a larger block we lose some load balance benefit. Eventually a tradeoff needs to be made between balanced loads and rendering overhead.

5 Future Development

We plan to extend our work to handle adaptive mesh refinement (AMR) datasets. We also plan to improve the block layout algorithm by optimizing various parameters.

Acknowledgements

The authors wish to thank Aaron Knoll, Eric Olson, for their valuable input and help during the course of this work.

References

[1] *Computer Graphics: Principles and Practice, second edition*. Addison-Wesley Professional, 1990.

[2] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. J. Hatcher. Large data visualization on distributed memory multi-gpu clusters. In *High Performance Graphics*, pages 57–66, 2010.

[3] K.-L. Ma, J. Painter, C. Hansen, and M. Krogh. A data distributed, parallel algorithm for ray-traced volume rendering. In *Parallel Rendering Symposium, 1993*, pages 15 –22, 105, oct 1993.

[4] S. Marchesin, C. Mongenet, and J.-M. Dischler. Dynamic load balancing for parallel volume rendering. In *EGPGV*, pages 43–50, 2006.

[5] C. Müller, M. Strengert, and T. Ertl. Optimized volume raycasting for graphics-hardware-based cluster systems. In *EGPGV*, pages 59–66, 2006.

[6] U. Neumann. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *Parallel Rendering Symposium, 1993*, pages 97 –104, oct 1993.

[7] B. N. Wylie, C. J. Pavlakos, V. Lewis, and K. Moreland. Scalable rendering on pc clusters. *IEEE Computer Graphics and Applications*, 21(4):62–70, 2001.