LA-UR-10-02237

Approved for public release;
distribution is unlimited.

Title:	Petascale visualization: Approaches and initial results
Author(s):	James P. Ahrens 113788 CCS-1 Li-Ta Lo 194699 CCS-1 Boonthanome Nouanesengsy Ohio State University John Patchett 148176 CCS-1 Allen McPherson 107908 CCS-1
Intended for:	Workshop on Ultrascale Visualization, pp.24-28, 16 Nov. 2008

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Form 836 (7/06)

Petascale Visualization: Approaches and Initial Results

James Ahrens, Li-Ta Lo, Boonthanome Nouanesengsy, John Patchett and Allen McPherson
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: ahrens@lanl.gov

***Abstract*—With the advent of the first petascale supercomputer, Los Alamos's Roadrunner, there is a pressing need to address how to visualize petascale data. The crux of the petascale visualization performance problem is interactive rendering, since it is the most computationally intensive portion of the visualization process. For terascale platforms, commodity clusters with graphics processors (GPUs) have been used for interactive rendering. For petascale platforms, visualization and rendering may be able to run efficiently on the supercomputer platform itself. In this work, we evaluated the rendering performance of multi-core CPU and GPU-based processors. To achieve high-performance on multi-core processors, we tested with multi-core optimized raytracing engines for rendering. For real-world performance testing, and to prepare for petascale visualization tasks, we interfaced these rendering engines with VTK and ParaView. Initial results show that rendering software optimized for multi-core CPU processors provides competitive performance to GPUs for the parallel rendering of massive data. The current architectural multi-core trend suggests multi-core based supercomputers are able to provide interactive visualization and rendering support now and in the future.**

I. Introduction

The Roadrunner supercomputer at Los Alamos National Laboratory recently achieved a long-sought supercomputing goal: performing more than a thousand trillion operations per second, or a petaflop. Roadrunner is the first supercomputer to use a hybrid processor architecture, which is based on both AMD Opteron processors and the IBM Cell Broadband Engine processing elements[3], [11]. The Cell processor is a collection of multi-core processors that achieves high-performance through the placement of multiple processing-cores on the same chip. This architectural trend drives all areas of computing, for example, there are currently tens of processors in a multi-core general-purpose CPU and hundreds of processors in a multi-core GPU. At the petascale range, supercomputers will generate an unprecedented amount of scientific data that will need to be visualized and analyzed in order to understand the science in the simulation results. This paper explores if we can efficiently run our visualization software on the supercomputing platform.

The data understanding process is composed of a number of activities including analysis and statistics, visualization and rendering. Visualization algorithms map simulation data to a visual representation. For example, an isosurfacing algorithm takes simulation data and produces contour geometry. Visualization, analysis and statistics already run fairly efficiently on most supercomputing platforms. Running these algorithms on the supercomputer, as part of parallel server, is an effiective way to visualize data without the need to move simulation data off the platform. Rendering maps geometry to imagery on the screen. On many previous supercomputing platforms it was difficult to interactively render directly on the platform. We define rendering for interactive exploration as 5-10 frames per second on a 1K by 1K pixel image at a minimum. We believe 60 frames per second is required for stereo rendering. Typically this performance has been provided by commodity clusters with attached graphics cards.

For petascale platforms, visualization and rendering may be able to run efficiently on the supercomputer platform instead of a visualization cluster. The contribution of this paper is a method to run interactive visualization on a multi-core based supercomputer using multi-core optimized ray-tracing. We tested rendering on multi-core CPU and GPU-based processors. Full testing of rendering using raytracing on Roadruuner using the Cell processor is currently in progress but not reported in this paper. For real-world performance testing, and to prepare for petascale visualization tasks, we started interfacing these rendering engines with VTK and ParaView (PV). Specifically we have an intial prototype using the Manta raytracing engine[2], that is optimized for multi-core rendering, integrated in VTK/PV. These additions will be available soon in the VTK/PV repositories for community use.

II. Related Work

As supercomputing technology has evolved from gigaflops to teraflops[5] and then to petaflops, visualization approaches have evolved as well. In the past decade, going from gigaflops to teraflops, visualization hardware for supercomputers went from SGI Infinite Reality graphics hardware[9] to commodity clusters with attached graphics cards[6]. The purpose of specialized graphics hardware in these systems was to support fast parallel rendering.

There are a number of different approaches to parallel rendering. Molnar et. al.[8] describe a sorting classification that characterizes when data is "sorted" to screen space.

IEEE Catalog Number: CFP08UVS
ISBN: 978-1-4244-2861-8
Library of Congress: 2008907542

Authorized licensed use limited to: Los Alamos National Laboratory. Downloaded on April 08,2010 at 17:46:34 UTC from IEEE Xplore. Restrictions apply.

Geometry sorting is identified as sort-first, scan-line sorting as sort-middle and image sorting as sort-last. A sort-last rendering approach is used in most parallel commerical and open-source visualization tools. Sort-last approaches do not require changes to the rendering software as is likely in the sort-middle case. Sort-last approaches also communicate load-balanced fixed-size images between processors. Load imbalance is a significant disadvantage of the sort-first case. Binary-swap[7] and direct-send[10] are two representative sort-last approaches. There are a number of different optimization approaches for sort-last rendering including network optimizations[12] and compression schemes[1].

III. Approach

Our approach to running visualization and rendering tasks on supercomputers was to evaluate the core components of the visualization process that are sufficiently different on supercomputers than on visualization clusters. We identified parallel rendering and specifically the rendering portion of the parallel rendering process, as the key bottleneck to interactive visualization on a supercomputer. This is where we focused our effort. We studied and evaluated two different rendering techniques: scan conversion approaches and raytracing approaches. Traditional rendering algorithms scan convert polygons to an image. To document the current state of the art we tested both software (Mesa) and hardware-accelerated (NVidia graphics card) scan-conversion approaches. In addition, we explored using raytracing techniques for polygonal rendering. Raytracing techniques offer more accurate results through the use of superior lighting models that incorporate reflection, refraction and shadows. Raytracing is inherently parallel and there are a number of high-performance multi-core implementations. We chose the open-source Manta raytracer[2] developed at the University of Utah for our work.

We use VTK as our visualization software as it already supports scan-conversion based rendering. The difficulty was replacing the OpenGL scan line renderer in VTK with the Manta raytracer.

IV. Implementation

We produced a version of VTK with the Manta raytracer replacing the OpenGL renderer. In this section we describe this implementation.

A useful feature of VTK is its rendering abstraction. Instead of a direct OpenGL interface, VTK offers a set of rendering classes that abstract the rendering process. Thus to have a raytracing rendering engine we needed to interface the raytracer to the VTK rendering interface.

There were three major issues:

A. Issue 1: Manta is a multi-threaded program

Manta is a multi-threaded program targeted to run efficiently on multi-core architectures. In order to maintain the correctness of the raytracer and thread safety, any modification to the state of the raytracer can only happen at certain barrier points in its main rendering loop. These necessary state modifications include adding and deleting geometric primitives and changing the position of the lights and camera. Manta provides a set of callback mechanisms to access these barrier points. We made use of these callbacks to synchronize the VTK rendering pipeline and Manta. We also made use of a similar mechanism to pause the raytracer after each frame is generated.

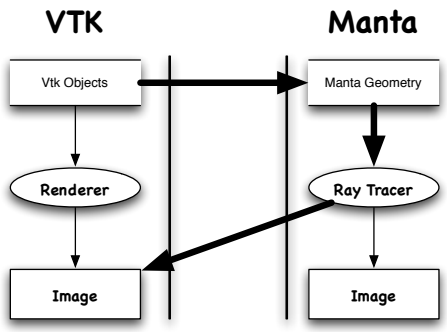

Fig. 1. A visual outline of the VTK Manta implementation challenges

B. Issue 2: Mapping VTK objects to Manta

Most of the mapping process from VTK to Manta polygonal data structures was straight-forward. We focused on triangle-based polygonal structures. One issue that needed to be addressed was mapping via a lookup table to color a polygonal surface by a scalar field. For example, an isosurface in a fluid simulation may be generated with a certain scalar value, say, the temperature of the fluid. Each cell or point of the isosurface may then be colored according to another scalar value, such as the magnitude of the velocity of the fluid. Generally, we can choose to give one color per cell or per point. When we color by cell, the same color is assigned to each point in the cell, thus, the whole cell has one uniform color. When we color by point, each point in the cell is assigned a different color and the cell is then smoothly shaded.

OpenGL renderers implement this feature in one of two ways. It can assign colors returned from the color map directly to the triangles or it can implement it through texture mapping. With a texture map, the renderer treats the color map as a 1D texture map and the texture coordinate is the scalar value used to index the color map. The renderer only has to assign each vertex the color map key as texture coordinate and the rendering hardware will do the rest of the job.

In Manta, we could have implemented cell coloring by assigning each triangle in the mesh its own material with a constant color. The problem with this approach is it needs to create a large number of Manta materials and associated textures and this can degrade performance. It also fails to solve the problem of coloring by point since there is no concept of ”per vertex” material for a triangle mesh in Manta; we can only assign materials on a triangle by triangle basis.

For scan-conversion, smooth shading is actually a weighted average of vertex color by using the barycentric coordinates of the interior point as a weight. In Manta, texture coordinates

Authorized licensed use limited to: Los Alamos National Laboratory. Downloaded on April 08,2010 at 17:46:34 UTC from IEEE Xplore. Restrictions apply.

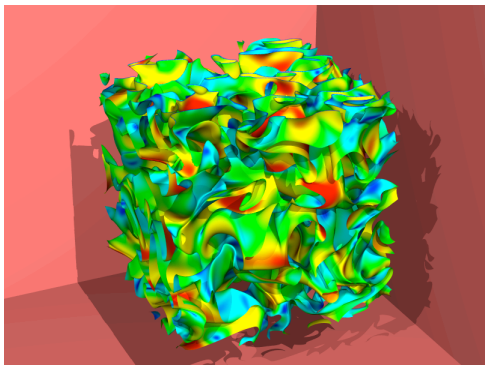

Fig. 2. A contour in a DNS-based turbulance simulation colored by velocity magnitude using a VTK-based Manta renderer

for an interior point of a triangle are calculated in exactly the same way. Thus we can use texture coordinate to calculate triangle interior point colors. There is no overhead with this method since texture coordinates are calculated at each ray-trangle intersection point at the shading phase anyway, even when we use a "constant color" for a triangle.

C. Issue 3: Depth buffer

A depth buffer is used in sort-last rendering algorithms to compute an ordering of different processors pixel values. To complete the VTK interface, Manta needed to produce a depth buffer. For a raytracer, the first ray/polygon intersection location can be used as a depth value. In Manta this value is calculated as part of the calculation of a pixel value. Additional coding was required to return it as a depth buffer value.

V. Results

In this section, we assess whether rendering on a supercomputer is a viable option. The standard approach for parallel rendering is a sort-last approach. Sort-last parallel rendering algorithms have two stages. The first stage is a rendering stage in which the processor renders its assigned geometry into a distance/depth buffer and image buffer. The second stage is a networking/compositing stage. Image buffers are composited together to create a complete result based on their associated depth buffers. In our performance analysis we will assume that the stages are pipelined and therefore performance is limited by the slower stage[4]. For rendering, we tested the performance of GPU and a 16-way multi-core CPU with both Mesa and Manta. We also tested compositing performance on Infiniband 1 and 2 network.

A. Rendering performance

We ran the GPU rendering test on an Nvidia Quadro FX 5600 with 1.5 GB of memory. We placed the GPU in a dual socket quad-core Opteron machine running at 2.0 GHz with 16GB of main memory. The rendering performance for a GPU in frames per second when rendering a million polygons to a 1K by 1K image are shown in Table I.

The timings of Mesa and Manta-based rendering were run on a quad socket, quad core Opteron (for a total of 16 cores) machine running at 1.1Ghz. The machine has a total 32 GB of main memory. For the multi-core tests we ran on 1, 2, 4, 8 and 16 cores. To achieve parallel rendering with Mesa, we used VTK to have one core partition the geometry and distribute it to other cores. All cores rendered their geometry, and then their images were composited together. The maximum time for any processor to render and composite is reported as the Mesa rendering performance.

Rendering Type	**Software**	**Architecture**	**Frames per second**
Scan conversion	OpenGL	Nvidia Quadro FX 5600	18.6

TABLE I
1 Million polygons rendering to a 1Kx1K image

			Frames per second				
Type	**Sw**	**Arch.**	**1**	**2**	**4**	**8**	**16**
Scan conv.	Mesa	4 quad core	0.7	1.2	2.0	3.2	4.6
Raytracing	Manta	4 quad core	1.6	2.8	5.6	10.9	19.4

TABLE II
1 Million polygons rendering to a 1Kx1K image

The rendering performance for both Mesa and Manta runs in parallel on 1, 2, 4, 8 and 16 cores in terms of frames per second when rendering a million polygons are shown in Table II. Table I shows a GPU performance of approximately 19 frames per second. Mesa performance on 16 cores was only 4.6 frames per second. Thus these results support the idea that given the choice between a commodity graphics cluster and a Mesa-based rendering supercomputer, a graphics cluster is required for high-performance interactive visualization and rendering. Manta performance on 16 cores, on the other hand, performed similarly to the real-world performance on a GPU: 19 frames per second. This result is significant because it suggests that multi-core software based rendering approaches may be viable alternatives to GPUs. Recall that Manta is a multi-threaded application designed to run effciently on multi-core processors. If Mesa were re-written to target efficient performance on multi-core processors, similiar performance results might be achieved.

B. Compositing performance

The compositing algorithm is an implementation of a binary-swap algorithm that is available in the parallel directory of VTK. We ran on a 128 node cluster with 1 process per node. Figure 3 and 4 show the compositing performance in frames per second for 1K by 1K image on the Infiniband 1 and 2, respectively. The blue lines show the network performance of the compositing step (i.e. without including the CPU time needed to composite images) while the red lines show the performance of the complete algorithm. Each pixel of the 1K by 1K image contained 5 floats, 1 each for r, g, b, α and z. This totals to 20 MB/image. It is important to note that compositing performance of 128 nodes is 10-15 frames per second. Recall

Authorized licensed use limited to: Los Alamos National Laboratory. Downloaded on April 08,2010 at 17:46:34 UTC from IEEE Xplore. Restrictions apply.

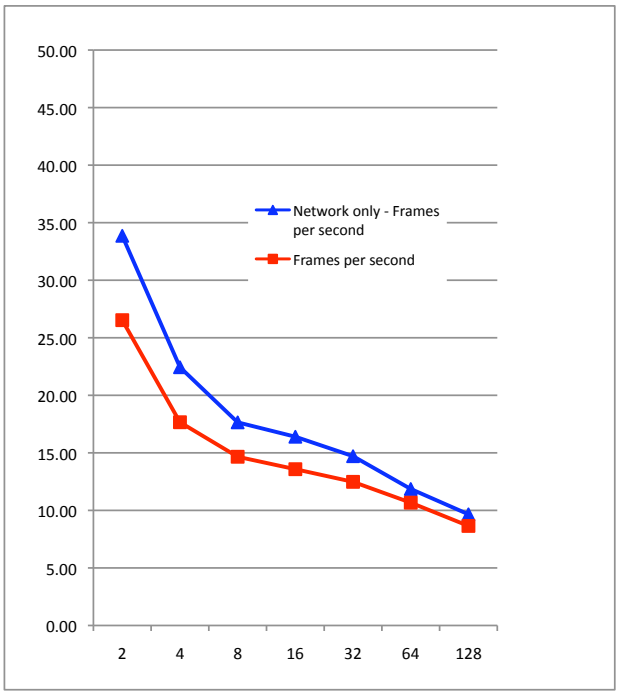

Fig. 3. Compositing Results for IB-1

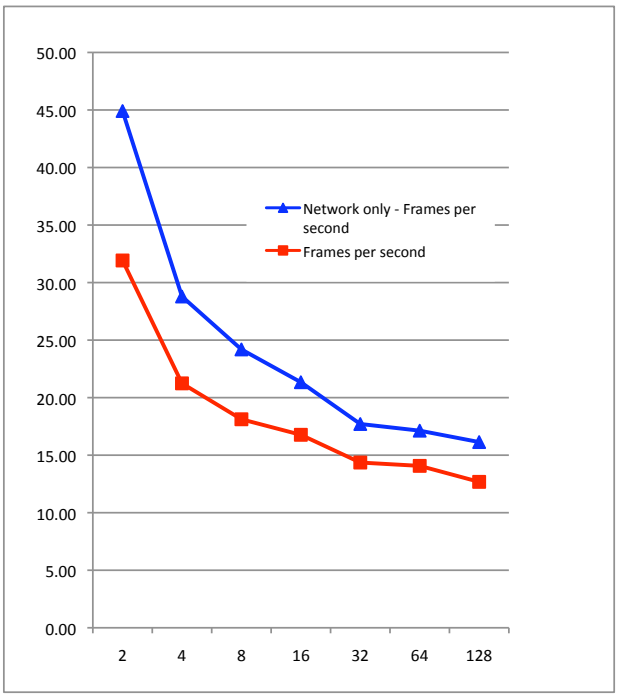

Fig. 4. Compositing Results for IB-2

that we achieve multi-core CPU software-based rendering performance of 19 frames per second. Therefore, in this test, the standard binary-swap algorithm for compositing/networking performance is the limiting pipelined performance factor and not the software-based rendering!

VI. Discussion

Our initial results suggest that software rendering on multi-core nodes of a supercomputer is a viable option for visualization. Therefore, we must consider the implications of using our supercomputers instead of using separate visualization clusters for this task. There are advantages and disadvantages to both approaches.

One of the main disadvantages of using the supercomputer for visualization is the cost to port rendering to the platform. Production quality interactive visualization and rendering on supercomputers will require a concerted software engineering effort. Portions of supercomputers will need to be set aside for interactive usage and this has historically been a challenge.

There are many advantages of using the supercomputer for visualization. We can scale up to use the entire supercomputer platform to visualize petascale results. We can visualize simulation results as they are generated in memory. This is an important approach as we start computing at the petascale. Petascale supercomputers will generate many more results in memory than can be saved to disk. For example, on Roadrunner, the data transfer rate to disk is 1 Gbyte/sec but data can be generated at 100 Gbytes/sec from a triblade (a collection of 32 cell processors) to memory.

Using visualization clusters has its own set of disadvantages. The cost of purchasing and maintaining visualization clusters and their associated networking and I/O infrastructure is large. The time spent moving data from supercomputers to visualization clusters is also large and increases the latency of the visualization and analysis cycle.

The advantages of a visualization cluster can not be overlooked. They are independent resources that are completely devoted to visualization tasks. They are very fast, especially on smaller data sets. They can also have their hardware-accelerated rendering component upgraded regularly at relatively low cost by replacing the graphics hardware.

We believe that these advantages and disadvantages will increasingly have to be evaluated. We think that interactive visualization on supercomputers is becoming a distinct possibility. With a community effort to make this approach a reality, we believe we can reduce costs and improve the visualization and analysis process.

VII. Future Work and Conclusions

Work is currently ongoing to integrate the IBM Cell-based raytracer into VTK/PV for visualization on Roadrunner platform. Once this is complete we will run the same performance testing as for the Manta raytracer. We want to evaluate what role raytracing will play in petascale visualization. Given raytracers can provide extremely good performance on supercomputing architectures, we will explore how visualization algorithms can be integrated into raytracers.

Initial results show that rendering software optimized for multi-core CPU processors provides competitive performance to GPUs for the parallel rendering of massive data. Therefore the current architectural trend of multi-core processors suggests multi-core based supercomputers will be able to provide

Authorized licensed use limited to: Los Alamos National Laboratory. Downloaded on April 08,2010 at 17:46:34 UTC from IEEE Xplore. Restrictions apply.

interactive visualization and rendering support now and in the future.

ACKNOWLEDGMENTS

This work was supported by the DOE NNSA ASC program and the DOE Office of Science.

REFERENCES

[1] James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. In *Proceedings of the 2nd Eurographics Workshop on Parallel Graphics and Visualization*, pages 145–151, 1998.

[2] J. Bigler, A. Stephens, and S.G. Parker. Design for parallel interactive ray tracing systems. *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 187–196, Sept. 2006.

[3] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 pflop/s trillion-particle kinetic modeling of laser plasma interaction on roadrunner. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

[4] X. Cavin, C. Mion, and A. Filbois. Cots cluster-based sort-last rendering: performance evaluation and pipelined implementation. *Visualization, 2005. VIS 05. IEEE*, pages 111–118, Oct. 2005.

[5] Philip D. Heermann. Production visualization for the asci one teraflops machine. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 459–462, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[6] Alan Heirich and Laurent Moll. Scalable distributed visualization using off-the-shelf components. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics*, pages 55–59, Washington, DC, USA, 1999. IEEE Computer Society.

[7] Kwan liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14:59–68, 1994.

[8] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, 1994.

[9] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinitereality: a real-time graphics system. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 293–302, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[10] Ulrich Neumann. Communication costs for parallel volume-rendering algorithms. *IEEE Comput. Graph. Appl.*, 14(4):49–58, 1994.

[11] Sriram Swaminarayan, Kai Kadau, Timothy C. Germann, and Gordon C. Fossum. 369 tflop/s molecular dynamics simulations on the roadrunner general-purpose heterogeneous supercomputer. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–10, Piscataway, NJ, USA, 2008. IEEE Press.

[12] Hongfeng Yu, Chaoli Wang, and Kwan-Liu Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.

Authorized licensed use limited to: Los Alamos National Laboratory. Downloaded on April 08,2010 at 17:46:34 UTC from IEEE Xplore. Restrictions apply.