Visualization at Extreme Scale Concurrency

Hank Childs

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

David Pugmire

Oak Ridge National Laboratory,
Oak Ridge, TN, USA, 37831.

Sean Ahern

Oak Ridge National Laboratory,
Oak Ridge, TN, USA, 37831.

Brad Whitlock

Lawrence Livermore National Laboratory,
Livermore, CA, USA, 94551.

Mark Howison

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Prabhat

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

Gunther Weber

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

E. Wes Bethel

Computational Research Division
Lawrence Berkeley National Laboratory,
Berkeley, California, USA, 94720.

October 2012

Acknowledgment

This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Some of the research in this work used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Legal Disclaimer

Abstract

There are some important and motivating questions that drive the research for processing massive data sets, like will it be possible to use the simpler pure parallelism technique to process tomorrow's data? Can pure parallelism scale sufficiently to process massive data sets?

To answer these questions, the researchers performed a series of experiments, originally published in *IEEE Computer Graphics and Applications* [2] and forming the basis of this report, that studied the scalability of pure parallelism in visualization software on massive data sets. These experiments utilized multiple visualization algorithms and were run on multiple architectures. There were two types of experiments performed. The first experiment examined performance at a massive scale: 16,000 or more cores and one trillion or more cells. The second experiment studied whether the approach can maintain a fixed amount of time to complete an operation when the data size is doubled and the amount of resources is doubled, also known as weak scalability. At the time of their original publication, these experiments represented the largest data set sizes ever published in visualization literature. Further, their findings still continue to contribute to the understanding of today's dominant processing paradigm (pure parallelism) on tomorrow's data, in the form of scaling characteristics and bottlenecks at high levels of concurrency and with very large data sets.

Preface

The material in this technical report is a chapter from the book entitled *High Performance Visualization—Enabling Extreme Scale Scientific Insight* [1], published by Taylor & Francis, and part of the CRC Computational Science series.

Contents

1 Overview—Pure Parallelism

Pure parallelism is the brute force approach to processing data: data parallelism with no optimizations to reduce the amount of data read. In this paradigm, the simulation writes data to a disk and the visualization software reads this data at full-resolution, storing it in primary memory. To deal with large data, parallel processing is used. The visualization software partitions data over its tasks, with each task working on a piece of the larger problem. Through parallelization, the visualization software has access to more I/O bandwidth (to load data faster), more memory (to store more data), and more compute power (to execute its algorithms more quickly).

The majority of visualization software for large data, including much of the production visualization software that serves large user communities, utilizes the pure parallelism paradigm. Some examples of tools that rely heavily on this processing technique include, VisIt, ParaView and EnSight.

The study described in this report sought to better understand how pure parallelism will perform on more and more cores, with larger and larger data sets. How does this technique scale? What bottlenecks are encountered? What pitfalls are encountered with running production software at a massive scale? In short, will pure parallelism be effective for the next generation of data sets? These questions are especially important because pure parallelism is not the only data processing paradigm. Where pure parallelism is heavily dependent on I/O bandwidth and large memory footprints, alternatives de-emphasize these traits.

The principal finding of this study was that pure parallelism at extreme scale works, that algorithms such as contouring and rendering performed well, and that I/O times for massive data dominated execution time. These I/O bottlenecks persisted over many supercomputers and also over I/O pattern (collective and noncollective I/O). These findings are discussed in Section 2. Another important finding was a validation of the weak scaling of pure parallelism when processing data sizes within the supercomputer's I/O bandwidth capabilities, and is described in Section 3. Finally, the study itself encountered common pitfalls at high concurrency that are the subject of Section 4.

2 Massive Data Experiments

The basic experiment for massive data at high levels of concurrency had a parallel program read in a data set with trillions of cells, apply a contouring algorithm ("Marching Cubes" [3]), and render the resulting surface as a 1024×1024 pixel image (see Fig. 1).

The study originally set out to perform volume rendering as well, but encountered difficulties (see Section 4 on Pitfalls). An unfortunate reality in experiments of this nature is that running large jobs on the largest supercomputers in the world is a difficult and opportunistic undertaking. Where the initial set of experiments demonstrated the problem, it was not possible for the authors to re-run data on these machines, after improvements were made to the volume rendering code. Further, these runs were undoubtedly affected by real-world issues, like I/O and network contention. That said, the study still had great value since isocontouring is representative of the typical visualization operations: loading data, applying an algorithm, and rendering.

The variations of this experiment fell into three categories:

- *Diverse supercomputing environments*, to test the viability of these techniques with different operating systems, I/O behavior, compute power (e.g., FLOPs), and network characteristics. These tests were performed on two Cray XT machines (Oak Ridge National Laboratory's JaguarPF and Lawrence Berkeley National Laboratory's Franklin), a Sun Linux machine (the Texas Advanced Computing Center's Ranger), a CHAOS Linux machine (Lawrence Livermore National Laboratory's Juno), an AIX machine (LLNL's Purple), and a BlueGene/P machine (LLNL's Dawn). For five of the six machines, the experiment consisted of 16,000 cores and visualizing one trillion cells. Runs on the Purple machine were limited to 8,000 cores and one half trillion cells, because the full machine has only

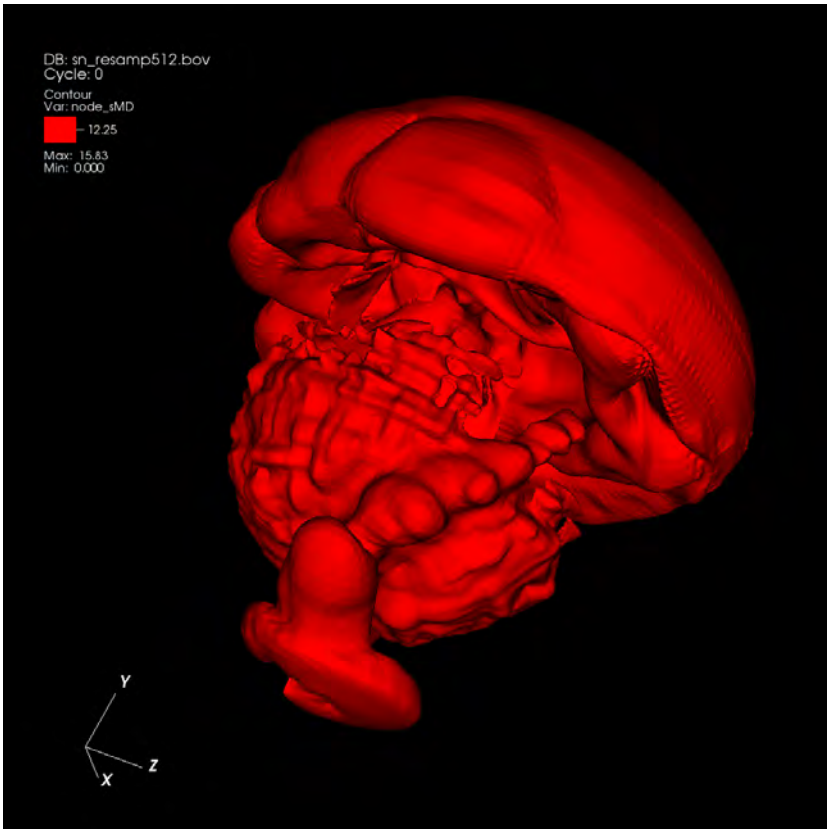

Figure 1: Contouring of two trillion cells, visualized with VisIt on Franklin using 32,000 cores. Image source: Childs et al., 2010 [2].

12,208 cores, and only 8,000 were easily obtainable for large jobs. For the machines with more than 16,000 cores available, like JaguarPF and Franklin, additional tests were added to perform a weak scaling study, maintaining a ratio of one trillion cells for every 16,000 cores. More information about the machines can be found in Table 1.

- *I/O pattern*, to understand the impact of collective and noncollective communication patterns at scale. Collective communication refers to an activity where there is coordination between the tasks; noncollective communication requires no coordination. For the noncollective tests, the data was stored as compressed binary data (gzipped). The study used ten files for every task and every file contained 6.25 million data points, for a total of 62.5 million data points per task. The study aimed to approximate real-world conditions, as simulation codes often write out one file per task and visualization codes receive, at most, one-tenth of the tasks of the simulation code. Of course, reading many small files is not optimal for I/O access, so the study also considered a separate test where all tasks use collective access on a single, large file via MPI-IO.
- *Data generation.* No simulations produced meshes with trillions of cells at the time of the study, so the experimenters created synthetic data. The primary mechanism for generating this data was to upsample data by interpolating a scalar field from a smaller mesh onto a high resolution rectilinear mesh. However, to offset concerns that upsampled data may be unrepresentatively smooth, the study included a second experiment where the large data set was a many times over replication of a small data set. The data set came from a core-collapse supernova simulation, using the CHIMERA code on a curvilinear mesh of

Machine Name	Machine Type/OS	System Type	Top 500 Rank (as of 11/2009)
JaguarPF	Cray	XT5	#1
Ranger	Sun Linux	Opteron Quad	#9
Dawn	BG/P	PowerPC	#11
Franklin	Cray	XT4	#15
Juno	Commodity (Linux)	Opteron Quad	#27
Purple	AIX	POWER5	#66

Machine Name	Total # of Cores	Memory Per Core	Clock Speed	Peak FLOPS
JaguarPF	224,162	2GB	2.6GHz	2.33PFLOPs
Ranger	62,976	2GB	2.0GHz	503.8 TFLOPs
Dawn	147,456	1GB	850MHz	415.7 TFLOPs
Franklin	38,128	1GB	2.6GHz	352 TFLOPs
Juno	18402	2GB	2.2GHz	131.6 TFLOPs
Purple	12,208	3.5GB	1.9GHz	92.8 TFLOPs

Table 1: Characteristics of supercomputers used in a trillion cell performance study.

more than three and one half million cells.[1] The use of synthetic data, while not ideal, was not a large concern for the experiment, since it was sufficient for meeting the study's primary objective: to better understand the performance and functional limits of parallel visualization software.

2.1 Varying over Supercomputing Environment

The first variant of the experiment was designed to understand differences from supercomputing environment. The experiment consisted of running an identical problem on multiple platforms, keeping the I/O pattern and data generation fixed, and using noncollective I/O and upsampled data generation. Results can be found in Figure 2 and Table 2.

Machine	Cores	Data set size	I/O	Contour	TPE	Render
Purple	8000	0.5 TCells	53.4s	10.0s	63.7s	2.9s
Dawn	16384	1 TCells	240.9s	32.4s	277.6s	10.6s
Juno	16000	1 TCells	102.9s	7.2s	110.4s	10.4s
Ranger	16000	1 TCells	251.2s	8.3s	259.7s	4.4s
Franklin	16000	1 TCells	129.3s	7.9s	137.3s	1.6s
JaguarPF	16000	1 TCells	236.1s	10.4s	246.7s	1.5s
Franklin	32000	2 TCells	292.4s	8.0s	300.6s	9.7s
JaguarPF	32000	2 TCells	707.2s	7.7s	715.2s	1.5s

Table 2: Performance across diverse architectures. "TPE" is short for total pipeline execution (the amount of time to generate the surface). Dawn's number of cores is different from the rest since that machine requires all jobs to have core counts that are a power of two.

[1]Sample data came courtesy of Tony Mezzacappa and Bronson Messer (ORNL), Steve Bruenn (Florida Atlantic University) and Reuben Budjiara (University of Tennessee).

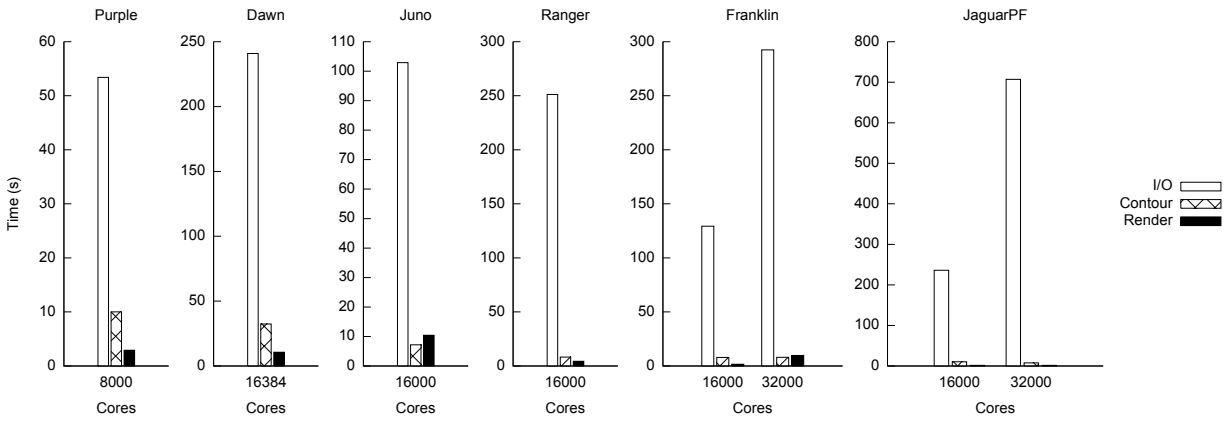

Figure 2: Plots of execution time for the I/O, contouring, and rendering phases of the trillion cell visualizations over six supercomputing environments. I/O was by far the slowest portion. Image source: Childs et al., 2010 [2].

There were several noteworthy observations:

- I/O striping refers to transparently distributing data over multiple disks to make them appear as a single fast, large disk; careful consideration of the striping parameters was necessary for optimal I/O performance on Lustre filesystems (Franklin, JaguarPF, Ranger, Juno, & Dawn). Even though JaguarPF had more I/O resources than Franklin, its I/O performance did not perform as well in these experiments, because its default stripe count was four. In contrast, Franklin's default stripe count of two was better suited for the I/O pattern which read ten separate compressed files per task. Smaller stripe counts often benefit file-per-task I/O because the files were usually small enough (tens of MB) that they would not contain many stripes, and spreading them thinly over many I/O servers increases contention.
- Because the data was stored on disk in a compressed format, there was an unequal I/O load across the tasks. The reported I/O times measure the elapsed time between a file open and a barrier, after all the tasks were finished reading. Because of this load imbalance, I/O time did not scale linearly from 16,000 to 32,000 cores on Franklin and JaguarPF.
- The Dawn machine has the slowest clock speed (850MHz), which was reflected in its contouring and rendering times.
- Some variation in the observations could not be explained by slow clock speeds, interconnects, or I/O servers:
  - For Franklin's increase in rendering time from 16,000 to 32,000 cores, seven to ten network links failed that day and had to be statically re-routed, resulting in suboptimal network performance. Rendering algorithms are "all reduce" type operations that are very sensitive to bisection bandwidth, which was affected by this issue.
  - The experimenters concluded Juno's slow rendering time was similarly due to a network problem.

2.2 Varying over I/O Pattern

This variant was designed to understand the effects of different I/O patterns. It compared collective and noncollective I/O patterns on Franklin for a one trillion cell upsampled data set. In the noncollective test, each task performed ten pairs of `fopen` and `fread` calls on independent gzipped files without any coordination among tasks. In the collective test, all tasks synchronously called `MPI_File_open` once, then called `MPI_File_read_at_all` ten times on a

shared file (each read call corresponded to a different piece of the data set). An underlying collective buffering, or "two phase" algorithm, in Cray's MPI-IO implementation aggregated read requests onto a subset of 48 nodes (matching the 48 stripe count of the file) that coordinated the low-level I/O workload, dividing it into 4MB stripe-aligned `fread` calls. As the 48 aggregator nodes filled their read buffers, they shipped the data using message passing to their final destination among the 16,016 tasks. A different number of tasks was used for each scheme (16,000 versus 16,016), because the collective communication scheme could not use an arbitrary number of tasks; the closest value to 16,000 possible was picked. Performance results are listed in Table 3.

I/O pattern	Cores	Total I/O time	Data read	Read bandwidth
Collective	16016	478.3s	3725.3GB	7.8GB/s
Noncollective	16000	129.3s	954.2GB	7.4GB/s

Table 3: Performance with different I/O patterns. The bandwidth for the two approaches are very similar. The data set size for collective I/O corresponds to four bytes for each of the one trillion cells. The data read is less than 4000GB because, 1GB is 1,073,741,824 bytes. The data set size for noncollective I/O is much smaller because it was compressed.

Both patterns led to similar read bandwidths, 7.4 and 7.8GB/s, which are about 60% of the maximum available bandwidth of 12GB/s on Franklin. In the noncollective case, load imbalances, caused by different compression factors, may account for this discrepancy. For the collective I/O, coordination overhead between the MPI tasks may be limiting efficiency. Of course, the processing would still be I/O dominated, even if perfect efficiency was achieved.

2.3 Varying over Data Generation

This variant was designed to understand the effects of source data. It compared upsampled and replicated data sets, with each test processing one trillion cells on 16,016 cores of Franklin using collective I/O. Performance results are listed in Table 4.

Data generation	Total I/O time	Contour time	TPE	Rendering time
Upsampling	478.3s	7.6s	486.0s	2.8s
Replicated	493.0s	7.6s	500.7s	4.9s

Table 4: Performance across different data generation methods. "TPE" is short for total pipeline execution (the amount of time to generate the surface).

The contouring times were nearly identical, likely since this operation is dominated by the movement of data through the memory hierarchy (L2 cache to L1 cache to registers), rather than the relatively rare case where a cell contains a contribution to the isosurface. The rendering time, which is proportional to the number of triangles in the isosurface, nearly doubled, because the isocontouring algorithm run on the replicated data set produced twice as many triangles.

3 Scaling Experiments

Where the first part of the experiment [2] informed performance bottlenecks of pure parallelism at an extreme scale, the second part sought to assess its weak scaling properties for both isosurface generation and volume rendering. Once again, these algorithms exercise a large portion of the underlying pure parallelism infrastructure and indicates a strong likelihood of weak scaling for other algorithms in this setting. Further, demonstrating weak scaling properties on high

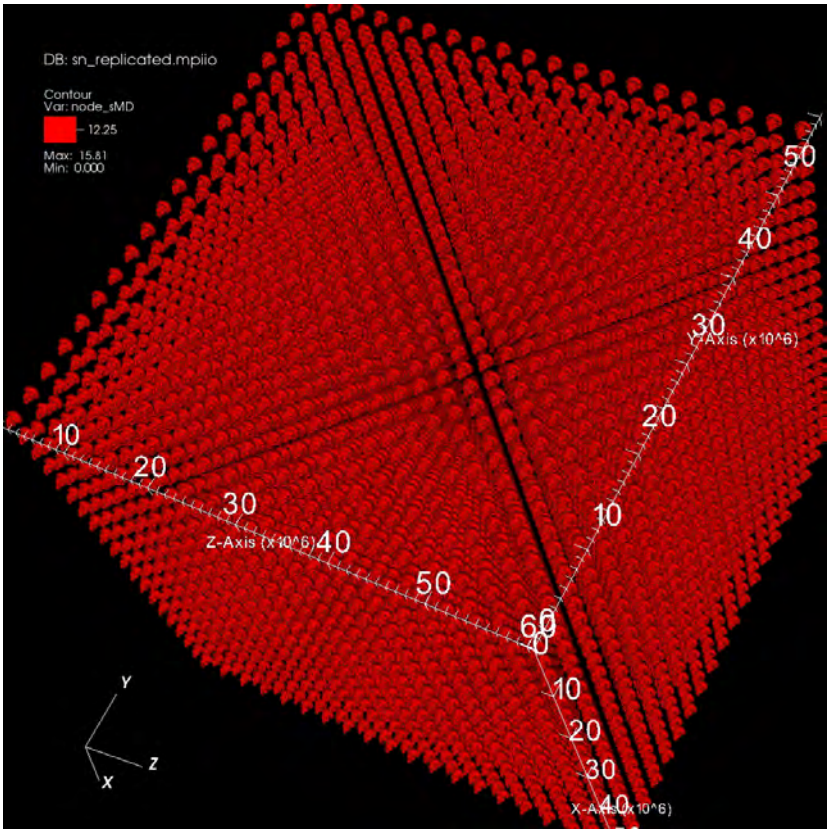

Figure 3: Contouring of replicated data (one trillion cells total), visualized with VisIt on Franklin using 16,016 cores. Image source: Childs et al., 2010 [2].

performance computing systems met the accepted standards of "Joule certification," which is a program within the U.S. Office of Management and Budget to confirm that supercomputers are being used efficiently.

3.1 Study Overview

The weak scaling studies were performed on an output from Denovo, which is a 3D radiation transport code from ORNL that models radiation dose levels in a nuclear reactor core and its surrounding areas. The Denovo simulation code does not directly output a scalar field representing effective dose. Instead, this dose is calculated at runtime through a linear combination of 27 scalar fluxes. For both the isosurface and volume rendering tests, VisIt read in 27 scalar fluxes and combined them to form a single scalar field representing radiation dose levels. The isosurface extraction test consisted of extracting six evenly spaced isocontour values of the radiation dose levels and rendering an 1024×1024 pixel image. The volume rendering test consisted of ray casting with 1000, 2000 and 4000 samples per ray of the radiation dose level on a 1024×1024 pixel image.

These visualization algorithms were run on a baseline Denovo simulation consisting of 103,716,288 cells on 4,096 spatial domains, with a total size on disk of 83.5GB. The second test was run on a Denovo simulation nearly three times the size of the baseline run, with 321,117,360 cells on 12,720 spatial domains and a total size on disk of 258.4GB. These core counts are large relative to the problem size and were chosen because they represent the number of cores used by Denovo. This matching core count was important for the Joule study and is also indicative of performance for an *in situ* approach.

3.2 Results

Tables 5 and 6 show the performance for contouring and volume rendering respectively, and Figures 4 and 5 show the images they produced. The time to perform each phase was nearly identical over the two concurrency levels, which suggests the code has favorable weak scaling characteristics. Note that I/O was not included in these tests.

Algorithm	Cores	Minimum Time	Maximum Time	Average Time
Calculate radiation	4,096	0.18s	0.25s	0.21s
Calculate radiation	12,270	0.19s	0.25s	0.22s
Isosurface	4,096	0.014s	0.027s	0.018s
Isosurface	12,270	0.014s	0.027s	0.017s
Render (on task)	4,096	0.020s	0.065s	0.0225s
Render (on task)	12,270	0.021s	0.069s	0.023s
Render (across tasks)	4,096	0.048s	0.087s	0.052s
Render (across tasks)	12,270	0.050s	0.091s	0.053s

Table 5: Weak scaling study of isosurfacing. **Isosurface** refers to the execution time of the isosurface algorithm, **Render (on task)** indicates the time to render that task's surface, while **Render (across tasks)** indicates the time to combine that image with the images of other tasks. **Calculate radiation** refers to the time to calculate the linear combination of the 27 scalar fluxes.

Cores	Samples Per Ray: 1000	2000	4000
4,096	7.21s	4.56s	7.54s
12,270	6.53s	6.60s	6.85s

Table 6: Weak scaling study of volume rendering. 1000, 2000, and 4000 represent the number of samples per ray. The algorithm demonstrates super-linear performance, because the number of samples per task (which directly affects work performed) is smaller at 12,270 task, while the number of cells per task is constant. The anomaly where performance increases at 2000 samples per ray requires further study. The times for each operation are similar at the two concurrency levels, showing favorable weak scaling characteristics.

4 Pitfalls at Scale

Algorithms that work well on the order of hundreds of tasks, can become extremely slow with tens of thousands of tasks. The common theme of this section is how implementations that were appropriate at modest scales became unusable at extreme scales. The problematic code existed at various levels of the software, from core algorithms (volume rendering), to code that supported the algorithms (status updates), to foundational code (plug-in loading).

4.1 Volume Rendering

VisIt's volume rendering code uses an all-to-all communication phase to redistribute samples along rays according to a partition with dynamic assignments. An "optimization" for this phase was to minimize the number of samples that needed to be communicated by favoring assignments

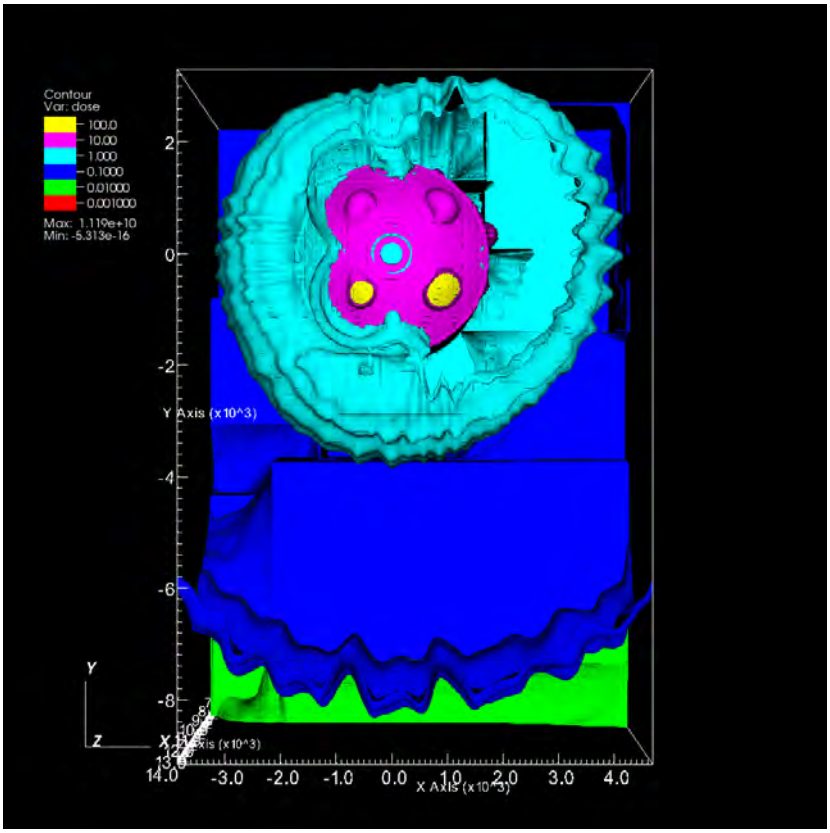

Figure 4: Rendering of an isosurface from a 321 million cell Denovo simulation, produced by VisIt using 12,270 cores of JaguarPF. Image source: Childs et al., 2010 [2].

that kept samples on their originating task. This "optimization" required an $O(n^2)$ buffer that contained mostly zeroes. Although this "optimization" was, indeed, effective for small task counts, the coordination overhead caused VisIt to run out of memory at this scale. Removing the optimization—by simply assigning pixels to tasks without concern of where individual samples lay—significantly improved performance. The authors concluded that, as the number of samples gets smaller with larger task counts, coordination costs outweigh the benefits that might come from keeping samples on the task where it was calculated.

The authors did not produce a comprehensive performance study for the one trillion cell data sets. However, they observed that after removing the coordination costs, ray casting performance was approximately five seconds per frame for a 1024×1024 image. The resulting image is shown in Figure 6.

After implementing this improvement, the authors were able to re-run the experiment on the Denovo data, however. They saw an approximate $5\times$ speedup running with 4,096 cores (see Table 7).

Cores	Date run	Samples Per Ray: 1000	2000	4000
4,096	Spring 2009	34.7s	29.0s	31.5s
4,096	Summer 2009	7.21s	4.56s	7.54s

Table 7: Volume rendering of Denovo data at 4,096 cores before and after speedup.

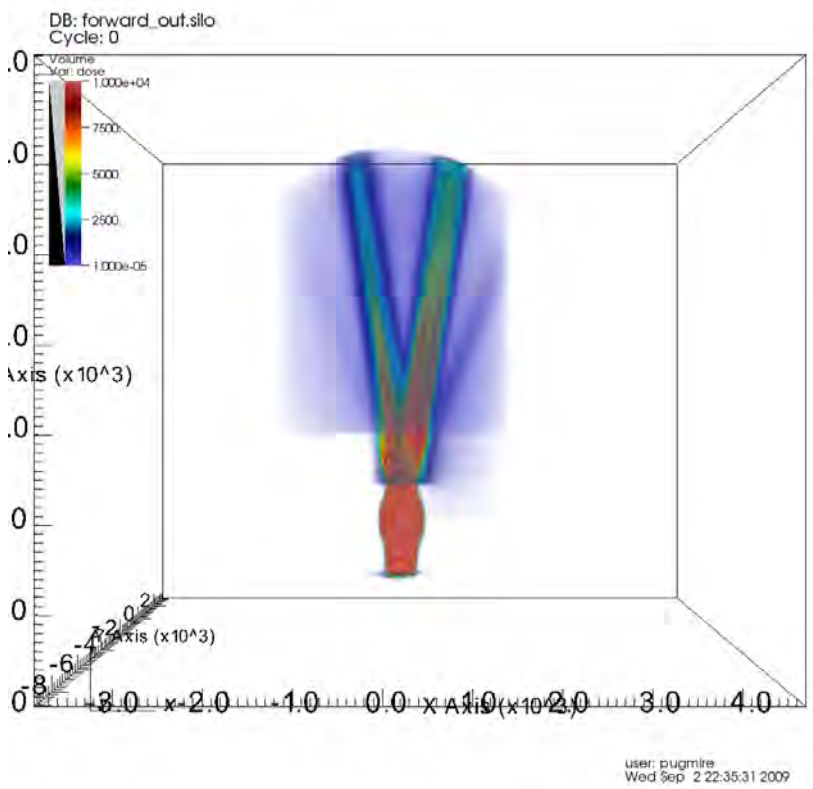

Figure 5: Volume rendering of data from a 321 million cell Denovo simulation, produced by VisIt using 12,270 cores on JaguarPF. Image source: Childs et al., 2010 [2].

4.2 All-to-One Communication

Upon completing a pipeline execution, each task reports its status (success or failure), as well as some metadata (extents, etc). These statuses and extents were being communicated from each task to task #0, through point-to-point communication. However, having every task send a message to task #0 led to a significant delay, as shown in Table 8. This problem was corrected subsequently with a tree communication scheme. Table 8 shows the extent of the delay using experiments run on LLNL's Dawn machine in June 2009 (using the old all-to-one status scheme) and August 2009 (using the new tree communication scheme).

Another "pitfall" was the difficulty in obtaining consistent results. Looking at the I/O times from the Dawn runs, there was a dramatic slowdown from June to August. This is because, in July, the I/O servers backing the file system became unbalanced in their disk usage. This caused the algorithm, that assigns files to servers, to switch from a "round robin" scheme to a statistical scheme, meaning files were no longer assigned uniformly across I/O servers. This scheme makes sense from an operating system perspective by leveling out storage imbalance, but it hampers access times for end users.

4.3 Shared Libraries and Start-up Time

The experimenters observed that VisIt's start-up time was longer than expected on Dawn, beginning at 4,096 tasks and worsening with higher task counts. They concluded it was because each task was reading plug-in information from the filesystem, creating contention for I/O resources. They partially addressed the problem during their experiments by modifying VisIt's plug-in infrastructure to load plug-in information on task #0 and to broadcast the information

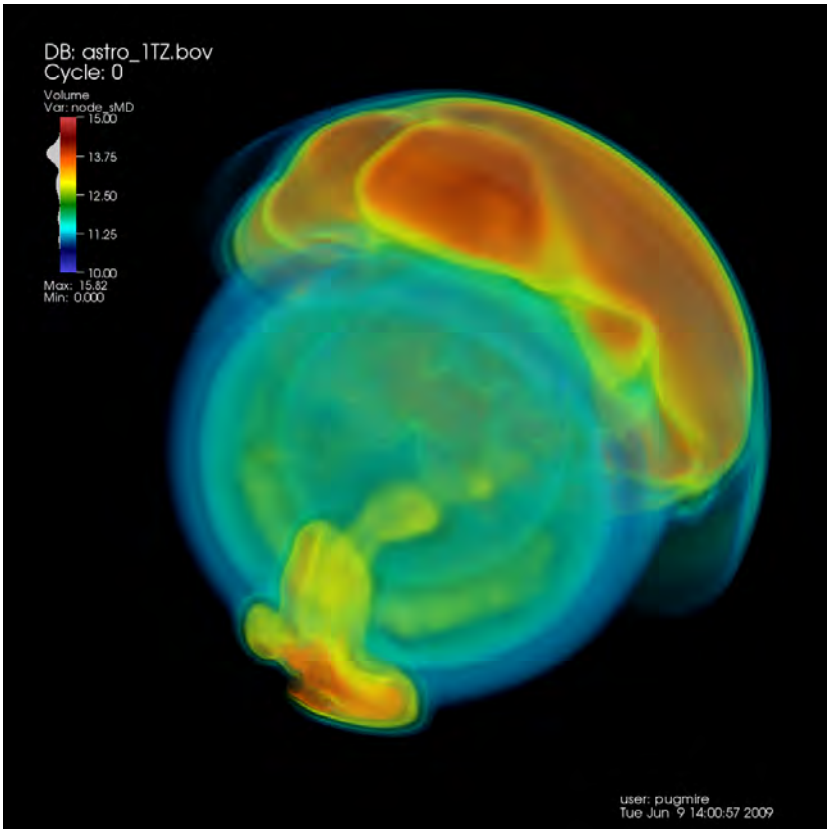

Figure 6: Volume rendering of one trillion cells, visualized by VisIt on JaguarPF using 16,000 cores. Image source: Childs et al., 2010 [2].

to other tasks. This change made plug-in loading nine times faster. However, start-up time was still quite slow, taking as long as five minutes.

VisIt's design made use of shared libraries because it encourages the development of new plug-ins that augment existing capabilities. Using shared libraries allows new plug-ins to access symbols that are not used by any current VisIt routines; linking statically removes these symbols. After this study, the developers decided that the performance penalties were too extreme at high task counts and so they added an option for static linking. Dynamic linking is still common, especially at low task counts—but static linking is now available for high task counts.

5 Conclusion

This report began by asking whether or not pure parallelism, the dominant paradigm for production visualization tools, could be successful for tomorrow's data sets. To answer this question, the report presented the results of a study designed to answer two questions about pure parallelism, which helps answer a much bigger question about the future of pure parallelism in exascale computing. (1) Can pure parallelism be successful on extreme data sets at extreme concurrency on a variety of architectures? And (2) does pure parallelism exhibit weak scaling properties? Although the results provided evidence that pure parallelism scales well, they also showed that the technique is only as good as its supporting I/O infrastructure and that I/O limitations are already prevalent on many supercomputers.

Insufficient I/O bandwidth will only increase in the future, as supercomputing budgets are inordinately devoted to FLOPs at the cost of I/O performance. Improvements can come from either software or hardware solutions, or some combination of the two. On the software side,

All-to-one status	Cores	Data set size	Total I/O time	Contour time	TPE	TPE minus contour & I/O
yes	16384	1 TCells	88.0s	32.2s	368.7s	248.5s
yes	65536	4 TCells	95.3s	38.6s	425.9s	294.0s
no	16384	1 TCells	240.9s	32.4s	277.6s	4.3s

Table 8: Performance with old status checking code vs. new status checking code. "TPE" stands for Total Pipeline Execution. The "TPE minus contour & I/O" time approximates the time spent waiting for status and extents updates. Note, the other runs reported in this experiment had a status checking code disabled and the last Dawn run was the only reported run with a new status code.

query-driven visualization, multiresolution processing, and *in situ* processing are approaches that all significantly reduce I/O. On the hardware side, emerging I/O technologies, such as solid-state drives (SSDs), offer significantly faster read times than the spinning disks. If simulations could stage their data on these SSDs for visualization programs, bypassing the spinning disk, it could increase access times by an order of magnitude and possibly make pure parallelism viable.

References

[1] E. Wes Bethel, Hank Childs, and Charles Hansen, editors. *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. Chapman & Hall, CRC Computational Science. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA, November 2012. `http://www.crcpress.com/product/isbn/9781439875728`.

[2] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther Weber, and E. Wes Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010.

[3] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Computer Graphics*, 21:163–169, August 1987.