SANDIA REPORT

SAND2010-6118
Unlimited Release
Printed September 2010

Visualization on Supercomputing Platform Level II ASC Milestone (3537-1B) Results from Sandia

Kenneth Moreland, Nathan Fabian, Pat Marion, Berk Geveci

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2010-6118
Unlimited Release
Printed September 2010

Visualization on Supercomputing Platform Level II ASC Milestone (3537-1B) Results from Sandia

Kenneth Moreland
Data Analysis and Visualization
Sandia National Laboratories
P.O. Box 5800 MS 1323
Albuquerque, NM 87185-1323
kmorel@sandia.gov

Nathan Fabian
Data Analysis and Visualization
Sandia National Laboratories
P.O. Box 5800 MS 1323
Albuquerque, NM 87185-1323
ndfabian@sandia.gov

Pat Marion
Kitware, Inc.
28 Corporate Dr.
Clifton Park, NY 12065
pat.marion@kitware.com

Berk Geveci
Kitware, Inc.
28 Corporate Dr.
Clifton Park, NY 12065
berk.geveci@kitware.com

Abstract

This report provides documentation for the completion of the Sandia portion of the ASC Level II Visualization on the platform milestone. This ASC Level II milestone is a joint milestone between Sandia National Laboratories and Los Alamos National Laboratories. This milestone contains functionality required for performing visualization directly on a supercomputing platform, which is necessary for peta-scale visualization. Sandia's contribution concerns *in-situ* visualization, running a visualization in tandem with a solver.

Contents

Appendix

Figures

Executive Summary

This report provides documentation for the completion of the Sandia portion of the ASC Level II Visualization on the platform milestone. This ASC Level II milestone is a joint milestone between Sandia National Laboratories and Los Alamos National Laboratories. The milestone text is shown below with the Sandia portions highlighted in boldfaced text.

> Visualization and analysis of petascale data is limited by several factors which must be addressed as ACES delivers the Cielo platform. Two primary difficulties are:
>
> 1. Performance of interactive rendering, which is 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.
> 2. **I/O bandwidth, which limits how much information can be written to disk. If we simply analyze the sparse information that is saved to disk we miss the opportunity to analyze the rich information produced every timestep by the simulation. For the first issue, we are pursuing *in-situ* analysis, in which simulations are coupled directly with analysis libraries at runtime.**
>
> This milestone will evaluate the visualization and rendering performance of current and next generation supercomputers in contrast to GPU-based visualization clusters, and **evaluate the performance of common analysis libraries coupled with the simulation that analyze and write data to disk during a running simulation. This milestone will explore, evaluate and advance the maturity level of these technologies and their applicability to problems of interest to the ASC program.**

Scientific simulation on parallel supercomputers is traditionally performed in four sequential steps: meshing, partitioning, solver, and visualization. Not all of these components are necessarily run on the supercomputer. In particular, the meshing and visualization typically happen on smaller but more interactive computing resources. However, the previous decade has seen a growth in both the need and ability to perform scalable parallel analysis, and this gives motivation for coupling the solver and visualization.

To objectively determine whether we meet the intention of the milestone, we selected this list of success criteria that must be completed.

1. Develop a "coprocessing" library: a simple interface to the ParaView post-processing tools that can be leveraged by solver codes.
   (a) Enable specification of processing via Python scripts that can be generated by ParaView and edited.
   (b) Enable "hard-coded" processing.
2. Demonstrate the use of the coprocessing library to perform processing *in situ* with two preexisting solvers.
   (a) Demonstrate the run-time selection of coprocessing calculations with the use of Python scripts.
3. Characterize the running time of the ParaView framework up to at least 1000 cores.
   (a) Instrument the overhead of the ParaView framework in successively larger groups.
   (b) Instrument the operation of some basic post-processing operations.
      i. Isosurface extraction
      ii. Slicing
      iii. Surface rendering
      iv. Data export (to file)

In addition to these minimum success criteria, we also set for ourselves a stretch goal to perform the characterization of criteria 3 for up to 10,000 cores.

Motivation

Although many projects integrate visualization with the solver to various degrees of success, for the most part visualization remains independent of the solver in both research and implementation. Historically, this has been because visualization was most effectively performed on specialized computing hardware and because the loose coupling of solver and visualization through reading and writing files was sufficient.

As we begin to run solvers on supercomputers with computation speeds in excess of one petaFLOP, we are discovering that our current methods of scalable visualization are no longer viable. Although the raw number crunching power of parallel visualization computers keeps pace with those of petascale supercomputers, the other aspects of the system, such as networking, file storage, and cooling, do not and are threatening to drive the cost past an acceptable limit [1]. Even if we do continue to build specialized visualization computers, the time spent in writing data to and reading data from disk storage is beginning to dominate the time spent in both the solver and the visualization [3].

Coprocessing, running the visualization and analysis in tandem with the solver, can be an effective tool for alleviating the overhead for disk storage [6], and studies show that visualization algorithms, including rendering, can often be run efficiently on today's supercomputers; the visualization requires only a fraction of the time required by the solver [7].

Whereas much of the other previous work in visualization coprocessing completely couples the solver and visualization components, thereby creating a final visual representation, our work provides a framework for the more general notion of salient data extraction. Rather than dump the raw data generated by the solver, our framework extracts the information that is relevant for analysis, possibly transforming the data in the process. The extracted information has a small data representation, which can be written at a much higher fidelity than the original data, which in turn provides more information for analysis. This difference is demonstrated in Figure 1.

A visual representation certainly could be one way to extract information (and one which we demonstrate), but there are numerous other ways to extract information. A simple means of extraction is to take subsets of the data such as slices or subvolumes. Other examples include creating isosurfaces, deriving statistical quantities, creating particle tracks, and identifying features.

The choice and implementation of the extraction varies greatly with the problem, so it is important that our framework is flexible and expandable. To this end, our coprocessing framework is based on the ParaView scalable visualization system. This provides us with a large set of analysis algorithms already available and also helps integrate our system with the ParaView application to simplify the specification of processing.

To demonstrate the necessary progress we made in providing coprocessing technology we

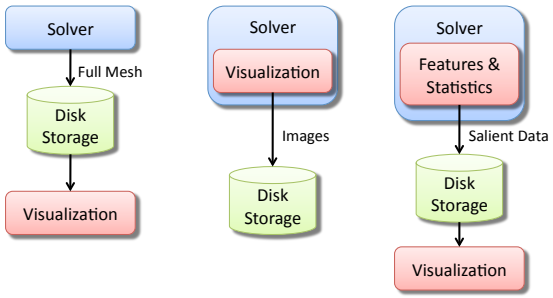


Figure 1: Different modes of visualization. In the traditional mode of visualization at left, the solver dumps all data to disk. Many *in-situ* visualization projects couple the entire visualization within the solver and dump viewable images to disk, as shown in the middle. Although the ParaView Coprocessing Library supports this mode, we encourage the more versatile mode at right where the coprocessing extracts salient features and computes statistics on the data within the solver.

first describe the development of the ParaView Coprocessing Library, then demonstrate its use, and finally characterize the running time of the parallel system.

Develop a coprocessing library

Our ParaView Coprocessing Library is a C++ library with interfaces to C, FORTRAN, and Python. It has two categories of input information: the simulation data (simulation time, time step, mesh, and fields) and the information that specifies what the ParaView Coprocessing Library extracts and when it gets outputted. The ParaView Coprocessing Library is built on the Visualization Toolkit (VTK) [4] and ParaView [5], and is currently maintained in and distributed from the ParaView code repository.

Since most use cases for the ParaView Coprocessing Library will be an extension of a current solver code, we cannot expect the API for the library to be able to easily and efficiently deal with all possible solver codes. The solution to this is to develop adaptors to translate data structures between the simulation code and what is specified for the API of the ParaView Coprocessing Library.

The ParaView Coprocessing Library is designed to be linked against the simulation application and requires that the solver code be modified to invoke it at given points during the solution. The solver code calls an adaptor module that is responsible for passing control information between the simulation code and the ParaView Coprocessing Library as well as converting the simulation code data structures to VTK data structures. The adaptor module has to be created for each simulation code with different data structures, but this results in the ParaView Coprocessing Library being independent of the solver codes.

Because of the separation between the solver and the ParaView Coprocessing Library, the solver may not be directly aware of when the coprocessing should be performed or what information is required in order to do the coprocessing. The adaptor is responsible for passing the simulation time and time step to the ParaView Coprocessing Library, which determines if any coprocessing needs to be performed for that invocation. If coprocessing is needed for that invocation, the adaptor returns what field information (i.e. temperature, velocity, etc.) is required. The adaptor then constructs the required information for the actual coprocessing computation.

One of the big advantages of basing the ParaView Coprocessing Library on the ParaView system is that almost all of the features within ParaView are also available within the ParaView Coprocessing Library. Any reader, source, filter, or writer available in ParaView is also available in the ParaView Coprocessing Library. The same parameters and options are available in both. There are some caveats, of course. Functions dependent on the GUI (such as Qt components) are not available because the Qt components are not compiled in the coprocessor. Filters that attempt to control time (such as the temporal interpolator) will probably not behave as expected as there is generally only data for the current iteration from the solver.

Demonstrate the ParaView Coprocessing Library

As a demonstration of the adaptability of the ParaView Coprocessing Library, we have integrated it with several solver codes: Sandia National Laboratories' CTH and ALEGRA, Los Alamos National Laboratories' NPIC, and Rensselaer Polytechnic Institute's PHASTA. These codes have been run on a variety of computing platforms, the largest of which are RedSky at Sandia National Laboratories, the CCNI BlueGene/L at Rensselaer Polytechnic Institute, and the Intrepid BlueGene/P at Argonne National Laboratory.

There are multiple ways to specify a visualization pipeline from within a solver. The first method is to "hard-code" the pipeline in C++. That is, from within the C++ interface code, establish a predefined pipeline. We have an adaptor for CTH that does just this. The CTH adaptor plugs into the Spymaster interface [2] and exposes a new command to its S-Lang interface.

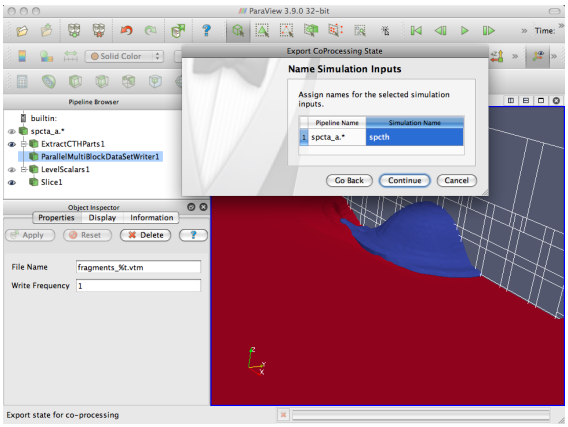
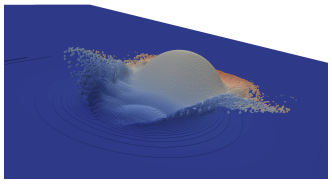


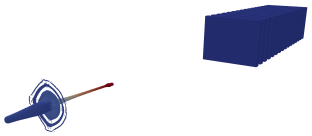
Figure 2: Using ParaView, the coprocessing plugin, and a proxy geometry to define an input script for coprocessing.

However, for much greater flexibility, the ParaView Coprocessing Library provides a means for the adaptor to supply a Python script that defines the visualization tasks to perform. This Python script may be loaded at runtime via, for example, from an input deck or read from a file. To assist in generating these scripts, ParaView provides a plugin that allows a user to establish a visualization pipeline in ParaView (generally with a proxy geometry) as shown in Figure 2 and then capture the state as a python script that can be loaded by the ParaView Coprocessing Library. Using this flexible input, we are capable of extracting and saving a variety of information including summary statistics in CSV files, mesh topology with field data, and rendered images such as those shown in Figure 3. Simply changing the Python script used, which can be done at runtime, can change the outputs generated from the same data, such as between Figures 3f and 3g.

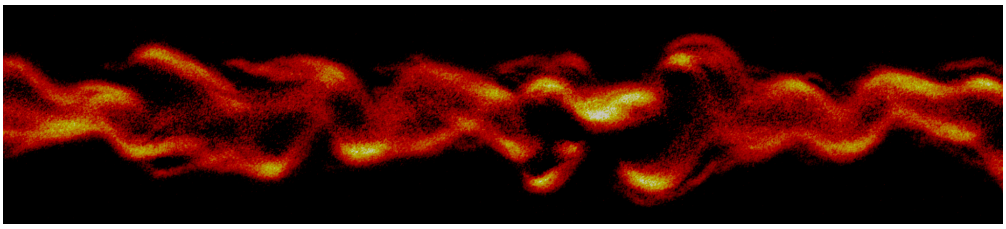


(a) CTH ball and brick test problem.

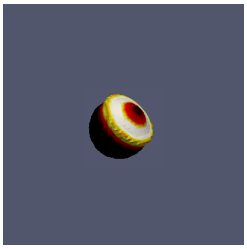
(b) CTH exploding pipe.



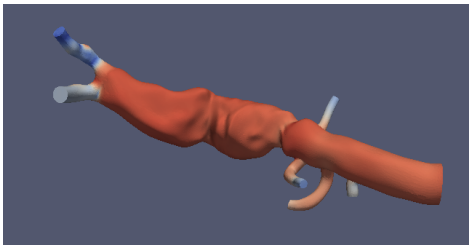
(c) CTH projectile.



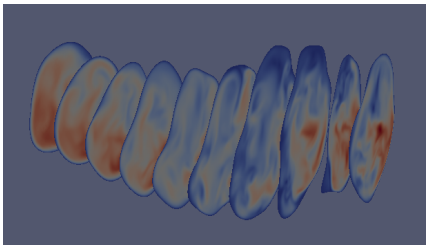
(d) NPIC instability of plasma within a current sheet.



(e) ALEGRA impacting spheres



(f) PHASTA abdominal aortic aneurysm surface pressure.



(g) PHASTA abdominal aortic aneurysm speed through slices.

Figure 3: Outputs from the ParaView Coprocessing Library running in tandem with a solver. The CTH, NPIC, and ALEGRA coprocessing generated images. The PHASTA coprocessing generated polygonal geometry.

Characterize the running time

To demonstrate that the ParaView Coprocessing Library is viable for large-scale visualization, we have performed several scaling studies both on the ParaView parallel services in isolation and when coupled with a solver via the ParaView Coprocessing Library. These tests monitor the performance of several common visualization operations as we move to larger numbers of processes and, nominally, larger data sizes.

Large Interactive Sessions

Our first set of experiments were performed on Sandia National Laboratories' RedSky cluster to determine the scalability of the interactive parallel ParaView server. Although these experiments did not exercise the ParaView Coprocessing Library itself, they do exercise the underlying parallel service that both the ParaView Coprocessing Library and the interactive server share.

For each of these experiments, we generated a data set using ParaView's wavelet source, which samples a three dimensional sinusoidal function on a uniform grid. This source provides a data set with nontrivial contours for many isosurface values and is easy to scale to arbitrary size. Because we can arbitrarily size this data, we used this source to perform a weak scaling study. That is, we made the size of the grid proportional to the number of processes so that across all experiments each process had roughly the same number of cells. Perfect scaling occurs when the execution time remains constant.

Our experiments involved creating the wavelet source, extracting several contours from the wavelet field, and then rendering the resulting surfaces. The experiment was repeated using 512, 1024, 2048, and 4096 cores, respectively.

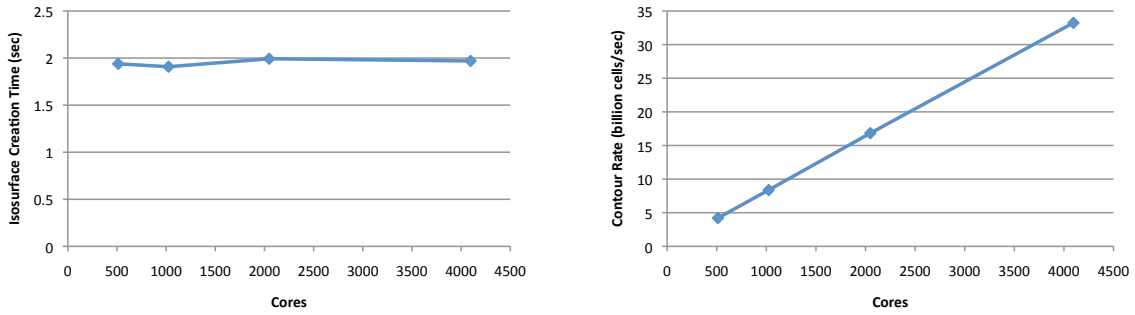


Figure 4: Running time and rate of extracting isosurfaces from a wavelet source.

Figure 4 shows the running time for various numbers of cores. As is clearly shown, the scaling for this isosurface algorithm is, by all practical measurements, perfect. The total run

time remains constant and if you consider the rate of computation measured in terms of cells per second, we see a linear speedup.

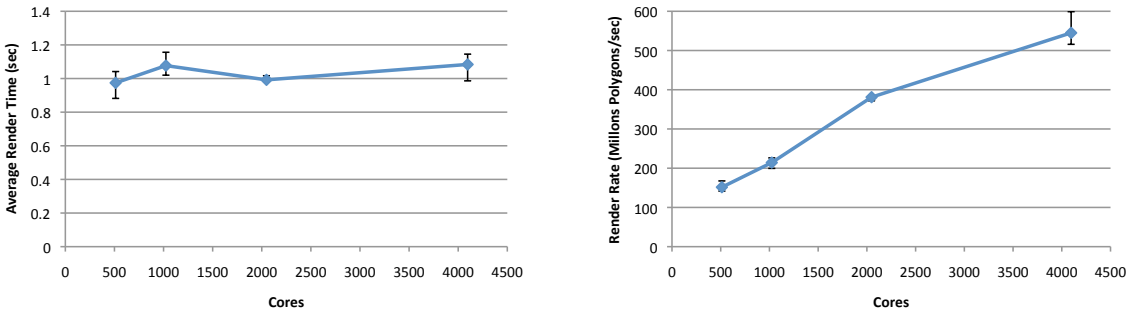


Figure 5: Running time and rate of rendering isosurfaces from a wavelet source.

Figure 5 shows the running time for various numbers of cores. The time shown is the average of multiple renders with an error bar showing the range of values. The scaling performance of our rendering is quite good.

Memory Usage of Coprocessing within CTH

This set of experiments was run on Sandia's RedSky computing platform using the CTH solver integrated with the ParaView Coprocessing Library. The source data comes from CTH and is identical for each of these experiments.

We ran two sets of experiments, first running CTH with no coupled analysis and then running CTH integrated with the coprocessor that extracted an isosurface and rendered it. Both sets of experiments were repeated on 256, 512, and 1024 cores of RedSky. Each experiment ran for 2–4 hours. While the experiments were running, we used the UNIX free command to periodically determine how much memory was being used on each node of the job (each node contains 8 cores). We can now use this data to compare the memory overhead of running with the coprocessor.

Figure 6 presents a detailed plot of the memory usage of each RedSky node. We present the usage this way because the important metric is to not exceed the available memory on any node. As can be seen, the overhead of using the ParaView Coprocessing Library is significantly smaller than the overall memory usage of the solver itself. For example, in the experiment run on 512 cores, it appears that partitioning decisions within CTH cause the maximum per-node memory to exceed that of running with the coprocessor (although the total amount of memory consumed without the coprocessor is, as expected, marginally less).

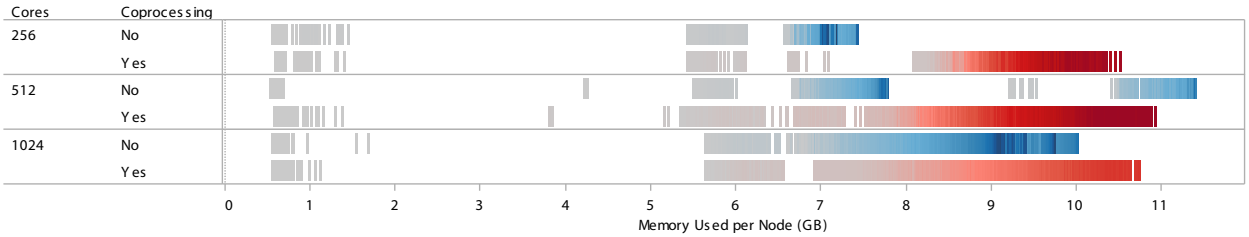


Figure 6: Plot of the memory usage of each RedSky node for each of the six CTH experiments. Darker hashes represent measurements taken later in the simulation.

Memory Usage of Coprocessing within NPIC

This set of experiments was run on Sandia's RedSky computing platform using the NPIC solver integrated with the ParaView Coprocessing Library. The source data comes from NPIC and its size is scaled linearly with respect to the number of processors used. In each case, every core processes 12,800 cells.

We ran two sets of experiments, first running NPIC with no coupled analysis and then running NPIC integrated with the coprocessor that extracted an isosurface and rendered it. Both sets of experiments were repeated on 256, 512, and 1024 cores of RedSky. Each experiment ran for 3.5–4 hours. While the experiments were running, we used the UNIX free command to periodically determine how much memory was being used on each node of the job (each node contains 8 cores). We can now use this data to compare the memory overhead of running with the coprocessor.

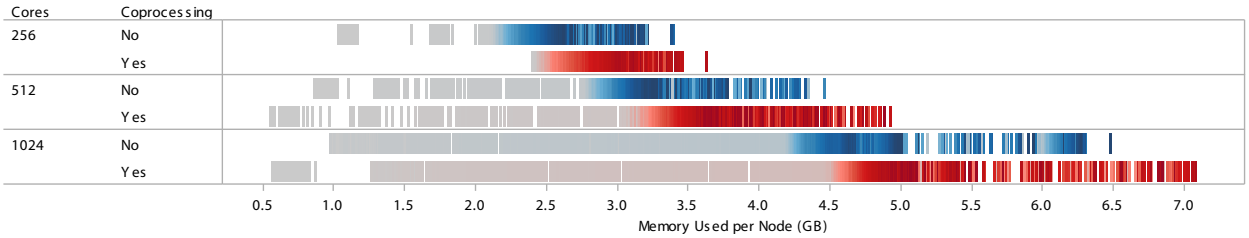


Figure 7: Plot of the memory usage of each RedSky node for each of the six NPIC experiments. Darker hashes represent measurements taken later in the simulation.

Figure 7 presents a detailed plot of the memory usage of each RedSky node. We present the usage this way because the important metric is to not exceed the available memory on any node. As can be seen, the overhead of using the ParaView Coprocessing Library is significantly smaller than the overall memory usage of the solver itself. Figure 8 presents a summary of the overhead by showing the maximum amount of memory needed per node that we encountered and the overhead added by the ParaView Coprocessing Library. According to these figures, the memory overhead incurred by the ParaView Coprocessing Library never exceeds 10% of the total memory.

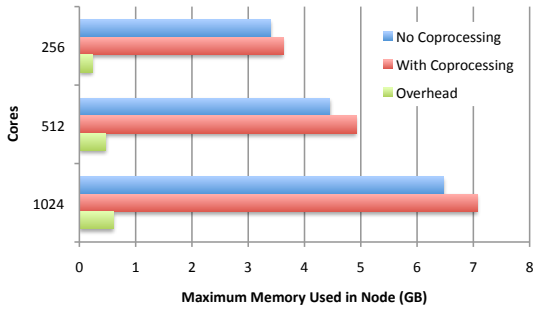


Figure 8: Summary of the maximum memory usage needed in each RedSky node with and without the ParaView Coprocessing Library. The computed overhead of the ParaView Coprocessing Library is also given.

PHASTA Coprocessing on CCNI

This set of experiments was run on the CCNI BlueGene/L using the PHASTA solver integrated with the ParaView Coprocessing Library. The source data comes from PHASTA itself and remains constant, as uniformly scaling it is not possible in general. Thus, we used this source to perform a strong scaling analysis. Perfect scaling occurs when the running time is inversely proportional to the number of processes.

Our experiments involved slicing the data by a number of cutting planes and then writing the sliced polygons to disk where they can be loaded later. The writing to disk included a collection operation that reduced the amount of contention on the BlueGene disks. The experiment was repeated using 512, 1024, 2048, 4096, and 8192 cores, respectively. The coprocessing was run for 10 separate time steps during the course of the simulation.

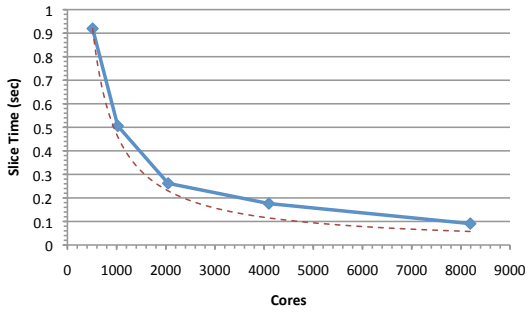


Figure 9: Running time of extracting slices from PHASTA data. The dashed line represents perfect scaling.

Figure 9 shows the running time of the slice filter. The times for each run are averaged over the ten time steps, although the variance between the running times amongst time steps is negligible. As can be seen, the running time drops significantly as cores are added and

closely follows perfect scaling.

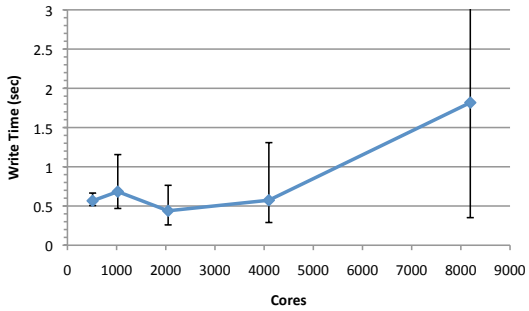


Figure 10: Time to write slices extracted from PHASTA data.

Figure 10 shows the running time for writing the data on various numbers of cores. The running times shown for each run are averaged over ten time steps with an error bar showing the range of values.

The running times are mostly level, and this is about as close to optimal as we can get with file I/O because although we are increasing the amount of computing resources, we are using about the same amount of file I/O resources each time. Also, the uptick in time for the larger jobs is misleading. Notice the large amount of variance in the write times. We believe that for some of the time steps there was contention with the file I/O that sabotaged performance and consequently increased the average time. Note that the minimum write time, when contention did not exist, is nearly constant.

PHASTA Coprocessing on Intrepid

This set of experiments was run on the Intrepid BlueGene/P using the PHASTA solver integrated with the ParaView Coprocessing Library. The source data comes from the PHASTA itself and remains constant, as uniformly scaling it is not possible in general. Thus, we used this source to perform a strong scaling analysis. Perfect scaling occurs when the running time is inversely proportional to the number of processes.

Our experiments involved slicing the data by a number of cutting planes and then writing the sliced polygons to disk where they can be loaded later. The experiments also ran a decimation algorithm on the full 3D mesh and wrote that resulting geometry to disk as well. The writing to disk included a collection operation that reduced the amount of contention on the BlueGene disks. The experiment was repeated using 4096, 8192, 16,384, and 32,768 cores, respectively. The coprocessing was run for 9 separate time steps during the course of the simulation.

Figure 11 shows the running time of the slice filter and the decimate filter. The times for

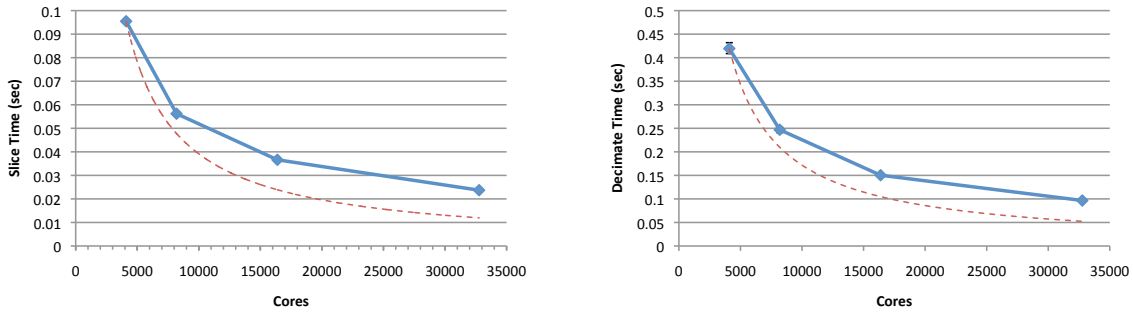


Figure 11: Running time of extracting slices (left) and decimating geometry (right) from PHASTA data. The dashed lines indicate perfect scaling.

each run are averaged over the nine time steps, although the variance between the running times amongst time steps is negligible in both cases. As can be seen, the running time drops significantly as cores are added. Although the running time diverges a bit from perfect scaling, the scaling is still quite good, particularly for a strong scaling study with running times under 0.1 seconds.

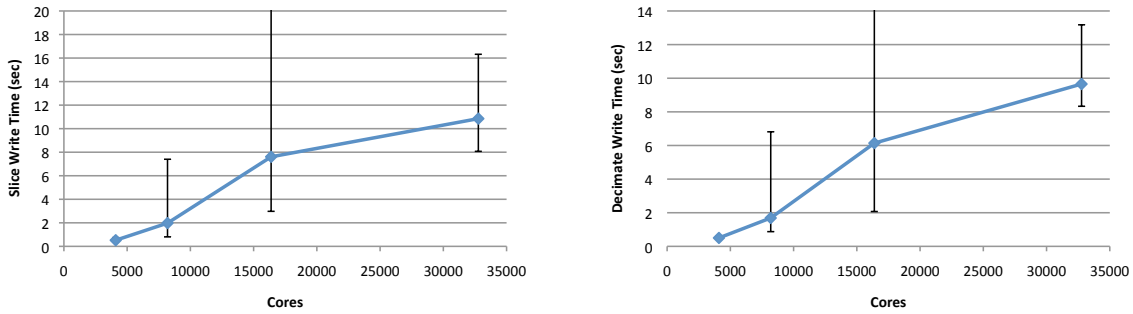


Figure 12: Time to write slices (left) and decimated geometry (right) extracted from PHASTA data.

Figure 12 shows the running time for writing the data on various numbers of cores. The running times shown for each run are averaged over nine time steps with an error bar showing the range of values.

Ideally the running times would be level; although we are adding more computing resources, we are using about the same amount of file I/O resources each time. In our case, the write time is actually increasing a bit even though we are writing roughly the same amount of data. This is probably caused by added contention in the file system as more processes attempt to access it. For future work, it appears that we could use a file I/O format and library that does a better job coordinating writes. We also notice that there appears to be file contention with other jobs accessing the same filesystem during some of the writes where the time increases drastically. There is little we can do about this contention.

Write Time Comparison

The main motivating factor for coprocessing within a solver is to circumvent problematic file I/O speeds. The assertion is that by performing our analysis before writing to disk, we can save time overall. To prove this assertion we compare the time it takes to write out a full mesh and the time it takes to write the data generated by our analyses.

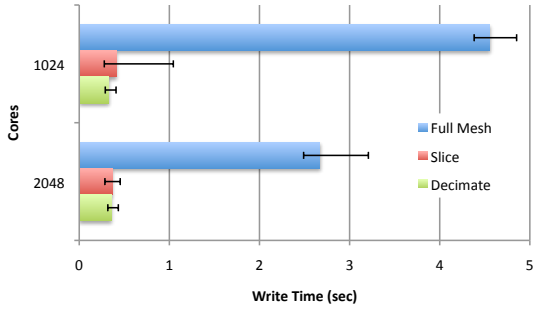


Figure 13: A comparison of the write times for full PHASTA mesh and the results of our slice and decimate filters.

Figure 13 compares the time spent writing the full image from a PHASTA simulation and the time spent writing the results of our slice and decimate filters described previously. These measurements were taken on the Intrepid BlueGene/P.

In each case the time spent writing the analysis results is significantly smaller than that to write out the entire data set. Even when considering the time spent in performing the analysis, the overall time is smaller. Were we to perform an analysis on a larger mesh, we would expect even greater savings.

Conclusion

As required by the milestone we have developed a common analysis library, demonstrated its use while coupled with multiple simulations, and evaluated its performance. We have also demonstrated the scalability of several common visualization operations.

Most of the operations we tested scale extremely well. Isosurfacing, slicing, decimating, and rendering all have near linear speedup. Of the operations we tested, writing is the most problematic. This is of no surprise as relatively poor file I/O performance is one of the main motivating factors for this work. Because our coprocessing system extracts relevant information and reduces the overall data *in situ*, we can tame the cost of file I/O. Nevertheless, our experiments suggest that optimizing the efficiency of our file I/O is one of the critical issues to address moving forward. For this we hope to leverage the wide breadth of research and development currently underway in file I/O systems.

References

[1] Hank Childs. Architectural challenges and solutions for petascale postprocessing. *Journal of Physics: Conference Series*, 78(012012), 2007. DOI=10.1088/1742-6596/78/1/012012.

[2] D. A. Crawford. Spymaster user's guide. Technical report, Sandia National Laboratories, February 2002.

[3] R B Ross, T Peterka, H-W Shen, Y Hong, K-L Ma, H Yu, and K Moreland. Visualization and parallel I/O at extreme scale. *Journal of Physics: Conference Series*, 125(012099), 2008. DOI=10.1088/1742-6596/125/1/012099.

[4] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Kitware Inc., fourth edition, 2004. ISBN 1-930934-19-X.

[5] Amy Henderson Squillacote. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2007. ISBN 1-930934-21-1.

[6] David Thompson, Nathan D. Fabian, Ken D. Moreland, and Lisa G. Ice. Design issues for performing *in situ* analysis of simulation data. Technical Report SAND2009-2014, Sandia National Laboratories, 2009.

[7] Tiankai Tu, Hongfeng Yu, Leonardo Ramirez-Guzman, Jacobo Bielak, Omar Ghattas, Kwan-Liu Ma, and David R. O'Hallaron. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

A Signoff Memos

Sandia National Laboratories

Operated for the U.S. Department of Energy by
Sandia Corporation

Albuquerque, New Mexico 87185

date: August 18, 2010

to: D. H. Rogers (01424), MS 1323; K. D. Moreland (01424) MS 1323

from: A. G. Salinger (01414), MS 1318

Andrew G. Salinger

subject: Achievement of Visualization on Supercomputing Platform Level II ASC Milestone (3537-1B)

I have examined the document "Visualization on Supercomputing Platform Level II ASC Milestone (3537-1B) Results from Sandia" dated August 18, 2010, submitted as evidence of successful completion of this FY2010 milestone. I also discussed the work at length with two of the main developers. I have not yet used the capability.

The Sandia team has delivered and *in-situ* visualization capability that attains scalability as well as flexibility and usability. The parallel scalability results are very good, and the extensions have a limited memory footprint. One result shows a factor of 10 decrease in output time due to pre-processing of the data for visualization. The high degree of usability comes from the ability to capture python code for the coprocessor from performing commands in the ParaView GUI, so the user does not have to learn a new interface. Since this python code is editable, expert users can then make modifications without going through ParaView GUI. The ability to code directly in C++ is also appropriate for other users.

In my judgment, as an independent observer, all of the stated acceptance requirements for this milestone have been met. In fact, the goal to demonstrate the use in two application codes was exceeded (they show results for four) as was the goal of characterizing the scaling up to 1000 processors (they went to 32768).

Exceptional Service in the National Interest

Sandia National Laboratories
Operated for the U.S. Department of Energy by
Sandia Corporation

Albuquerque, New Mexico 87185

date: September 9, 2010

to: Sudip S. Dosanjh, Ph.D.

subject: Achievement of Visualization on Supercomputing Platform Level II ASC Milestone 3537

Sudip:

On Thursday, September 9, 2010 we held a formal review of ASC Level II Milestone 3537: Visualization on Supercomputing Platform. The review was held in CSRI 148. The review team consisted of Bruce Hendrickson, Senior Manager of Computer Science and Mathematics Group, Rob Hoekstra, Manager for Applied Math and Applications Department, Erik Strack, Manager for Comp. Shock and Multiphysics Department, Ron Brightwell, Manager Scalable System Software Department, and Greg Weirs, code developer in Comp. Shock and Multiphysics Department. Also in attendance were Kenneth Moreland and Nathan Fabian, members of the Milestone team.

Kenneth Moreland and I presented the Milestone results from both LANL and Sandia, with emphasis on the work done by Sandia's team members. Kenneth presented evidence that demonstrated the delivery of an open source, VTK-based analysis library that can be coupled with simulations, allowing them to perform a range of analysis and visualization operations at runtime. In particular, Kenneth showed that this is a general purpose library by showing results from running the library coupled with NPIC (LANL), CTH (SNL) and preliminary results from Alegra (SNL). In addition, Kenneth presented evidence of interactive runs on 4K processors (utilizing the same underlying analysis code), as well as results from in-situ runs up to 32K cores by collaborators at Kitware on Argonne's Intrepid Supercomputer.

Based on the success of these results, Kenneth and I also presented the preliminary plan for follow up work in FY11 and FY12, which includes collaboration with the Sierra code team, and continued engagement with external partners leveraging and contributing to this open source library.

The review team all agreed that the Sandia Milestone work was fully completed.

Sincerely,

David H. Rogers,
Manager, Department 1424 Data Analysis and Visualization

Exceptional Service in the National Interest

DISTRIBUTION:

1 Pat Marion
Kitware, Inc.
28 Corporate Drive
Clifton Park, NY 12065

1 Berk Geveci
Kitware, Inc.
28 Corporate Drive
Clifton Park, NY 12065

2	MS 1323	Kenneth Moreland, 1424
1	MS 1323	Nathan Fabian, 1424
1	MS 1323	David Rogers, 1424
1	MS 1318	Andrew Salinger, 1414
1	MS 1322	Sudip Dosanjh, 1420
1	MS 1319	Ronald Brightwell, 1423
1	MS 1319	James Ang, 1422
1	MS 0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories