Scalability of Paraview's Coprocessing Capability

Nathan Fabian, *Sandia National Laboratories*

ABSTRACT: For exceedingly large high performance computing runs, writing all data to disk is unmanageably slow. It becomes necessary to have analysis and visualization communicate with the simulation in memory instead of through disk, retaining access to all available data for analysis. The open source visualization and analysis tool, Paraview, has recently added a coprocessing API allowing it to be linked into simulation codes. We will demonstrate scalability of Paraview coprocessing on up to 64000 cores on the new NNSA platform, Cielo.

KEYWORDS: coprocessing, in situ

1 Introduction

Planning is in place to reach exascale by the 2018-2021 timeframe [1]. High performance computing software must be ready to run at this scale. Historically visualization has most effectively performed on specialized visualization hardware. While the raw number crunching power of visualization clusters has kept pace with the larger systems supporting simulation, the cost of saving data out to disk and possibly moving it and then reading it from disk on the visualization cluster is becoming unacceptably high [11]. As simulations problems grow to exascale in size, so much data will be written to disk it may not be worth writing it at all.

By coupling the visualization with the simulation either directly via memory-to-memory copy or through the network to a process running on a separate visualization cluster, we can avoid writing to disk as late into the final analysis as possible. Whereas many projects have integrated visualization with the solver to various degrees of success, they have tended to completely couple the solver and visualization components, thereby creating a single path to a final visual representation. Instead we use ParaView's coprocessing library which provides a framework for the more general notion of salient data extraction. Rather than simply render the raw data generated by the solver, in coprocessing we extract 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 shown in Figure 1.

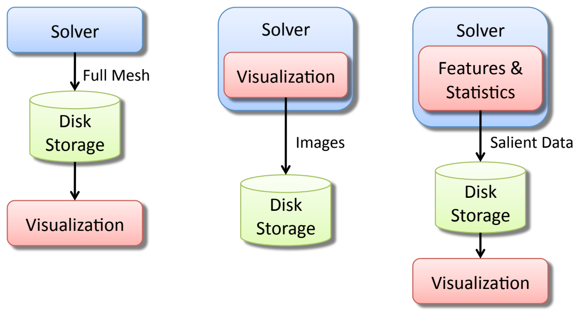

Figure 1: Different modes of visualization. In the traditional mode of visualization at left, the solver dumps all data to disk. Many *in-situ* visualization projects couple the entire visualization within the solver and dump viewable images to disk, as shown in the middle. We encourage the more versatile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

In this paper we will discuss the coprocessing library

[1]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.

and in particular focus on scaling a fragment detection algorithm. This algorithm is a compelling example because one of the results analysts are interested in is a histogram of various quantities on each fragment, thus regardless of original input data size the final extracted data of interest is a very small constant size, easily written to disk. While it might be useful to incorporate this particular algorithm directly into the simulation code, by implementing it through the coprocessing API of a general visualization solver, we retain access to a larger number of possible analysis results, such as the raw fragment geometry for visual inspection. We will focus on the effort to scale this algorithm from running effectively on a typical visualization cluster size of up to 512 cores to being capable of running on 65,536 cores of the new Alliance for Computing at Extreme Scale (ACES) petaFLOP platform, Cielo.

2 Previous Work

Coupling visualization with the simulation was first mentioned in the 1987 National Science Foundation Visualization in Scientific Computing workshop report, [9] which is often attributed to launching the field of scientific visualization. Over the years, there have been many visualization systems built to run in tandem with simulation, often on supercomputing resources. Recent examples include a visualization and delivery system for hurricane prediction simulations [4] and a completely integrated meshing-to-visualization system for earthquake simulation [14]. These systems are typically lightweight and specialized to run a specific type of visualization under the given simulation framework. A general coupling system exists [5] which uses a framework called EPSN to connect *M* simulation nodes to *N* visualization nodes through a network layer.

SCIRun [7] provides a general problem solving environment that contains general purpose visualization tools that are easily integrated with several solvers so long as they are also part of the SCIRun problem solving environment. Other more general purpose libraries exist that are designed to be integrated into a variety of solver frameworks such as pV3 [6] and RVSLIB [3]. However, these tools are focused on providing imagery results whereas in our experience it is often most useful to provide intermediate geometry or statistics during coprocessing rather than final imagery.

Ultimately, the integration of coprocessing libraries into solvers gets around the issues involved with file I/O. There are also some related efforts in making the I/O interfaces abstract to allow loose coupling through file I/O to be directly coupled instead. Examples include the Interoperable Technologies for Advanced Petascale Simulations (ITAPS) mesh interface [2] and the Adaptable I/O System (ADIOS) [8]. If these systems become widely adopted, then it could simplify the integration of coprocessing libraries with multiple solvers.

3 Coprocessing Library

ParaView coprocessing is a C++ library with an externally facing API to C, FORTRAN and Python. It is built atop the Visualization Toolkit (VTK) [12] and ParaView [13]. Nearly all the algorithms available in traditionally interactive interface are also available through the coprocessing pipeline. The exceptions are those algorithms that expect to see the entire time-series. When coprocessing only the current time step is available.

To interface a new simulation with the coprocessing library, we write a small piece of code to translate data structures between the simulation's code and the coprocessing library's VTK-based architecture. An adaptor is also responsible for determining when the coprocessing library will run. By calling out to the coprocessing pipeline at regular intervals (even as often as every simulation step) the coprocessing can run at as high a fidelity as necessary to achieve the required level of accuracy.

4 Simulation Coupling

CTH is an Eulerian shock physics code that uses an adaptive mesh refinement (AMR) data model. We examine a simulation of an exploding pipe bomb, Figure 2. Using an algorithm which finds water-tight fragment isosurfaces over each material volume fraction within the AMR cells, we can find fragments that separate from the original mesh and measure various quantities of interest in these fragments.

The challenge in finding an isosurface over values in an AMR mesh is in the difference of resolution between cells. More so this difference can also bridge processor boundaries, requiring ghost cell information at two different resolutions. We handle this by finding connected neighbors using an all-to-all communication at the beginning of computation and then exchanging ghost data between only connected neighbors and finally perform the AMR corrected isosurface algorithm. The result is a polyhedral mesh surface which contains no gaps and can be significantly smaller than the original AMR mesh. In some cases, where an analyst is only concerned with a

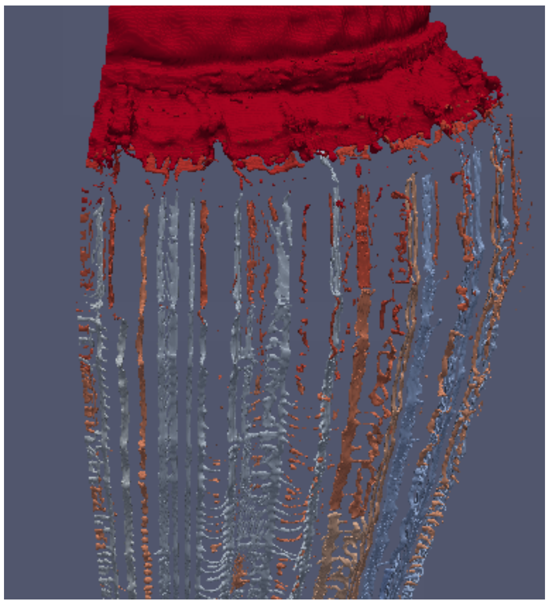

Figure 2: Fragments detected in a simulation of an exploding pipe.

histogram of fragment quantities, the data written can be on the order of bytes.

In general, CTH is running at the upper edge of available memory, which leaves very little room for the adaptor to copy the simulations memory into the pipeline. One solution may be to increase the number of nodes in the job request so that the memory is not so limited, but this is usually not an option. Instead, we have developed an interface above the standard VTK array to shallow copy the data from CTH and view it in it is native layout. Although VTK has the ability to work with external pointers to memory, the layout in CTH is different than the one VTK algorithms expect. Therefore the interface acts as a wrapper over the array's accessor functions to translate to CTH layout before reading from a memory location. This allows us to circumvent the memory copy using a much smaller overhead.

A future option to explore with the coprocessing framework would involve combining ParaView's streaming capability with the adaptor source. This would allow the pipeline to work in extremely limited memory cases by passing small pieces of the simulation memory through the pipeline at a time. While the above method for addressing CTH's memory restrictions works well, the fragment detection algorithm creates a surface representation of the data which is in addition to the memory representing the AMR volume. By employing the streaming interface the pipeline could operate on fragments individually, which would require much less resident memory with some additional computational overhead. Looking into the future at exascale where computation is much less expensive than everything else this option may become unavoidable.

5 Results

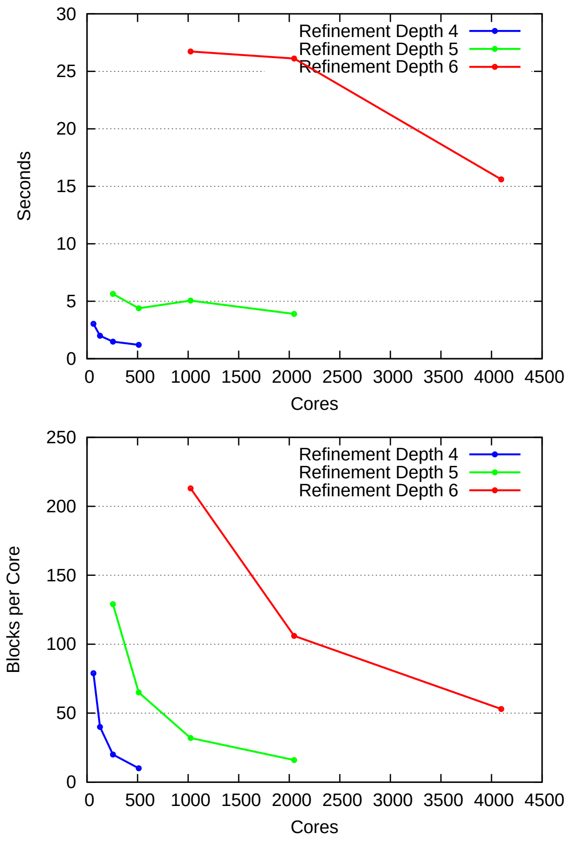

Figure 3: Results running on the capacity cluster, Redsky. Each line is an instance of strong scaling at one particular refinement depth of the AMR representation. The top image shows the running time, and the bottom image shows the corresponding number of blocks per core.

To simplify the scaling process, we increase only the depth limit of CTH's mesh refinement process. By incrementing this parameter the mesh size will increase by at most a factor of eight, but in practice will increase less than eight due to CTH choosing not to refine certain regions. We ran each depth as a strong scaling problem up until the number of blocks per core dropped below a minimum suggested by the simulation. We then increase the depth and rerun to overlap the timings. Each refinement depth is represented as a separate line on all charts.

In order to establish the scaling, we first tested the algorithm at lower scale using Sandia's capacity cluster, Redsky, Figure 3. Although there is some noise in the measurements, it scales acceptably through to 4096 cores.

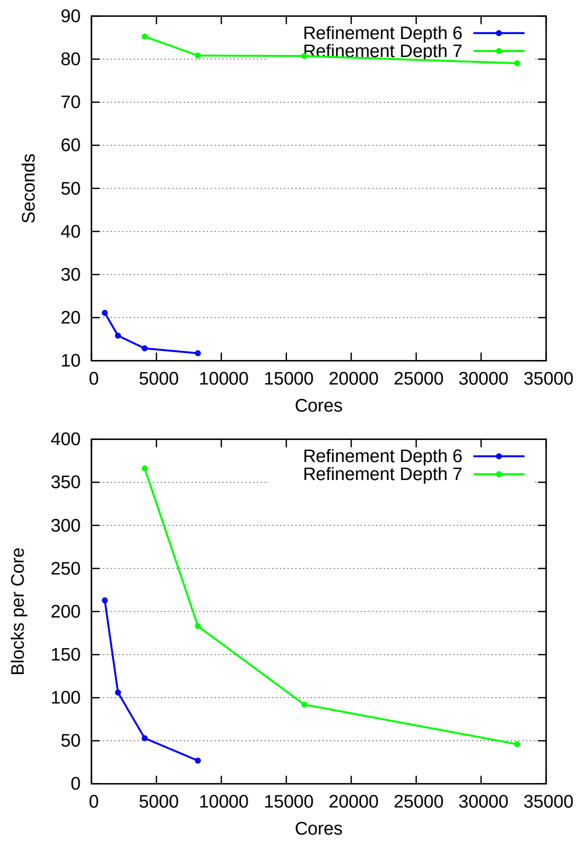

Figure 4: Results running on Cielo. As in Figure 3 each line is an instance of strong scaling at a refinement depth.

Once we established scaling on the capacity cluster, we ran this pipeline on the new ASC/NNSA machine Cielo using from 1 thousand to 32 thousand cores. The results of these runs are shown in Figure 4, again with each refinement depth represented as a separate line. The sizes in number of blocks resulting from these refinement levels are shown below.

Although this algorithm works well enough to achieve a high number of communicating cores, as is visible in Figure 4, it is no longer performing any speedup beyond 16 thousand processors. Although we have executed at 64 thousand cores, the pipeline didn't run to completion in a reasonable amount of time, so no timing results are available. Our next steps will be to increase the speedup for the lower processor counts leading up to the 64 thousand run. It is important to note that these scalability results are specific to this fragment algorithm and not to the framework in general. For further work on the framework's general scalability, please see Moreland et al. [10]

Currently the fragment finding requires an all-to-all communication to determine neighbor information. This information is also maintained within the simulation code, but not exported through normal file I/O to conserve space. The current adaptor implementation relies on the existing file I/O API in the simulation to transfer the data structures to VTK. However, because we are running in the same memory space as the simulation, we can potentially access this information via the adaptor and avoid the visualization having to repeat finding neighbor information.

Despite the current slowdown, there remains an overall speedup due to the scalability of the simulation. Thus it is still effective to run coprocessing on larger problems using larger numbers of cores. More importantly, it is much faster than writing the full dataset to disk.

6 Conclusion

In this paper we have examined fragment detection as a means of extracting the features of interest in a CTH dataset. For exceptionally large runs storing, moving, and managing data can be a prohibitive expense. The ability to shrink the data down to a small constant-sized histogram of interesting values becomes increasingly necessary. Although it takes more computation to produce results with a coprocessing simulation, these results can be substantially smaller than the unprocessed version time is saved not writing to disk. In addition, when the analysis results are produced with the simulation results no further analysis processing or data management is needed.

Acknowledgements

The author would like to thank Charles Law at Kitware Inc. for explaining the depths of the fragment detection algorithm and Andrew Bauer at Kitware Inc. for implementing the coprocessing API.

About the Authors

Nathan Fabian is a Member of Staff in the Scalable Analysis and Visualization department at Sandia National Laboratories. Nathan is the technical lead for Sandia's coprocessing efforts.

References

[1] J. Ang, D. Doerfler, S. Dosanjh, S. Hemmert, K. Koch, J. Morrison, and M. Vigil. The alliance for computing at the extreme scale. In *Proceedings of CUG2010*, Edinburgh, UK, 2010.

[2] K. Chand, B. Fix, T. Dahlgren, L. F. Diachin, X. Li, C. Ollivier-Gooch, E. S. Seol, M. S. Shephard, T. Tautges, and H. Trease. The ITAPS iMesh interface. Technical Report Version 0.7, U. S. Department of Energy: Science Discovery through Advanced Computing (SciDAC), 2007.

[3] S. Doi, T. Takei, and H. Matsumoto. Experiences in large-scale volume data visualization with RVSLIB. *Computer Graphics*, 35(2), May 2001.

[4] D. Ellsworth, B. Green, C. Henze, P. Moran, and T. Sandstrom. Concurrent visualization in a production supercomputing environment. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), September/October 2006.

[5] A. Esnard, N. Richart, and O. Coulaud. A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006)*, pages 7–14, Torremolinos, Malaga, Spain, October 2006. IEEE Press.

[6] R. Haimes and D. E. Edwards. Visualization in a parallel processing environment. In *Proceedings of the 35th AIAA Aerospace Sciences Meeting*, number AIAA Paper 97-0348, January 1997.

[7] C. Johnson, S. Parker, C. Hansen, G. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, December 1999.

[8] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, pages 15–24, 2008.

[9] B. H. McCormick, T. A. DeFanti, and M. D. Brown, editors. *Visualization in Scientific Computing (special issue of Computer Graphics)*, volume 21. ACM, 1987.

[10] K. Moreland, N. Fabian, P. Marion, and B. Geveci. Visualization on supercomputing platform level ii asc milestone (3537-1b) results from sandia. Technical Report SAND2010-6118, Sandia National Laboratories, 2010.

[11] 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.

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

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

[14] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. 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.