LLNL-TR-411304

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Distributed Data-Flow for In-Situ Visualization and Analysis at Petascale

D. E. Laney, H. R. Childs

March 2, 2009

Disclaimer

Auspices Statement

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work was funded by the Laboratory Directed Research and Development Program at LLNL under project tracking code 08-FS-006.

FY08 LDRD Final Report
Distributed Data-Flow or In-Situ Visualization and Analysis at Petascale
LDRD Project Tracking Code: 08-FS-006
Daniel Laney, Principal Investigator

Abstract

We conducted a feasibility study to research modifications to data-flow architectures to enable data-flow to be distributed across multiple machines automatically. Distributed data-flow is a crucial technology to ensure that tools like the VisIt visualization application can provide in-situ data analysis and post-processing for simulations on peta-scale machines. We modified a version of VisIt to study load-balancing trade-offs between light-weight kernel compute environments and dedicated post-processing cluster nodes. Our research focused on memory overheads for contouring operations, which involves variable amounts of generated geometry on each node and computation of normal vectors for all generated vertices. Each compute node independently decided whether to send data to dedicated post-processing nodes at each stage of pipeline execution, depending on available memory. We instrumented the code to allow user settable available memory amounts to test extremely low-overhead compute environments. We performed initial testing of this prototype distributed streaming framework, but did not have time to perform scaling studies at and beyond 1000 compute-nodes.

Introduction/Background

As the scientific simulation community starts using petascale computers, the amount of data produced by these simulations will be staggering. Interactively visualizing and analyzing this large data in a traditional manner is often difficult, primarily because I/O bandwidth is relatively small. One approach to this problem is to process the data in situ, meaning that the simulation code itself is coupled with visualization and analysis algorithms, which operate on the data without it ever being written to disk. However, it is important to consider the memory footprint of these algorithms. Scientific simulations typically saturate the supercomputer's primary memory and can only spare a relatively small buffer for visualization and algorithms.

To overcome the memory footprint problem, we propose a modification to the standard in situ paradigm. Our proposed system does distributed in situ visualization, meaning that some visualization work is done on the supercomputer and the rest is sent to a small, dedicated cluster. The driving principle behind this design is to maximize how much work is done by the many processors on the supercomputer, but, at the same time, enable the memory footprint on each these processors to remain low.

An important motivation for this design lies in the heterogeneous nature of work each processor has to do. Scientific simulations often parallelize by partitioning space into "domains," and assigning one domain to each processor. When running in situ, each processor operates only on that processor's domain. But the amount of work to be performed on a given domain will often vary greatly from processor to processor.

For example, if the goal is to calculate an isosurface, many of the domains may not intersect the isosurface. The isosurface on these domains can be calculated quickly and with minimal memory footprint overhead. But other domains may contain many intersections with the isosurface and produce many facets. These facets may exceed the memory budget from the simulation, meaning the work could not be done in situ. We address this problem by identifying the domains that will generate many facets and sending them to a dedicated post-processing resource.

In-situ processing and postprocessing with dedicated resources have typically been viewed as an "either-or" proposition. With this architecture, we introduce a hybrid approach that allows the two traditional approaches to be viewed as extremes on a spectrum (see figure 1). We believe portions of the spectrum covered by the hybrid space are significant because it responds to the trend in supercomputing platforms towards low memory footprint, which in turn hampers in situ processing.

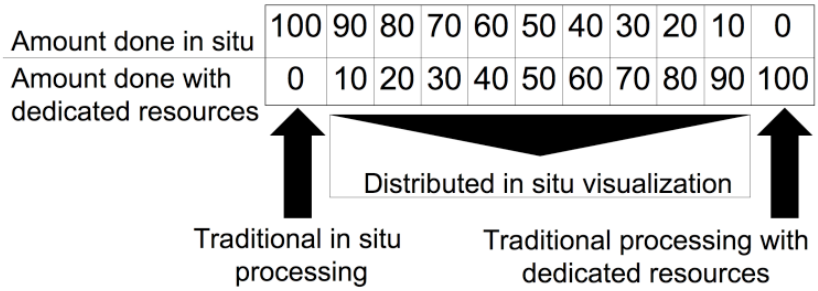

Figure 1

Related Work

Ma et. al. [2] conjecture that in-situ processing is the most plausible solution to the problem of visualization and analysis at peta and exa scale and produce a preliminary study of strategies to implement an in-situ based visualization system.

Ma et. al. [4] examines three areas related to visualization and analysis at peta scale and beyond. They present a study of the relative costs of I/O, rendering, and compositing for a volume rendering application. They investigate the implications of eliminating the processing from the workflow and examine a technique that uses data reorganization to improve access times for the volume rendering use case. In contrast, the present work regards data reorganization as a technique that is not generally applicable to the case of a running simulation. Our approach is also directed at a larger set of use-cases spanning standard visualization to more costly and intensive analysis in which derived fields and user expressions are used to extract meaning from data.

In a related work, Ma et. al. [3] describe large scale volume rendering on the Blue Gene/P architecture. They study the relative costs of various strategies for obtaining scalable volume rendering, as a stand-in for other visualization and analysis tasks. Their goal was not to obtain the fastest results, but to understand the trade-off between using compute hardware and dedicated assets with hardware accelerated capabilities. They present data from several experiments studying several parameters such as data set size, number of processors, offline storage of results and streaming of images for remote display. Their overall conclusion is that a BlueGene/P type architecture is a viable platform for some types of processing at

extreme scales. The approach of the present paper agrees with this assessment, but goes further in stating that rendering performance is only one criterion for determine the proper approach for peta and exa scale visualization and analysis. In this paper we study a more general visualization query that combines several operations in a data flow architecture that is closer to how users typically investigate a large simulation.

Research Activities

As most of the major visualization packages use data flow networks, we implemented our system in such a framework inside the VisIt visualization application. When a user constructs a desired visualization through the user interface, corresponding data flow networks are constructed on both the in-situ machine and on the dedicated machine. These networks are identical and the only thing that differentiates them are the portions of the data set they operate on. Furthermore, filters on the in-situ machines are modified to contain a "predictor" module and filters on the dedicated machine are modified to contain a "receiver" module. The predictor module determines if its accompanying filter is in danger of exceeding the memory budget. If so, the data set is from the in situ machine to the dedicated machine. Each processor from the in-situ machine initially operates on that processor's domain from the simulation code. Each processor from the dedicated machine initially starts with no data. Execution then proceeds as follows:

1. For each filter in the pipeline
   - If the filter is on the in-situ machine
     - (a) If the predictor module returns true
       1. Send the input to the filter to the dedicated machine
       2. Cease all further processing on this node
     - (b) If the **predictor** module returns false
       1. Execute the filter
       2. Proceed to next filter in the pipeline
   - If the filter is on the dedicated machine
     1. Execute receiver module. If data is sent from an in situ processor, then add that data to this filter's input (which
   - may have started as empty).
   - (b) Execute the filter
   - (c) Proceed to next filter in the pipeline
2. Transfer any remaining data from the in situ machine to the dedicated machine
3. Perform all rendering on the dedicated machine

To provide more clarity about the design, consider the following notional example in figure 2

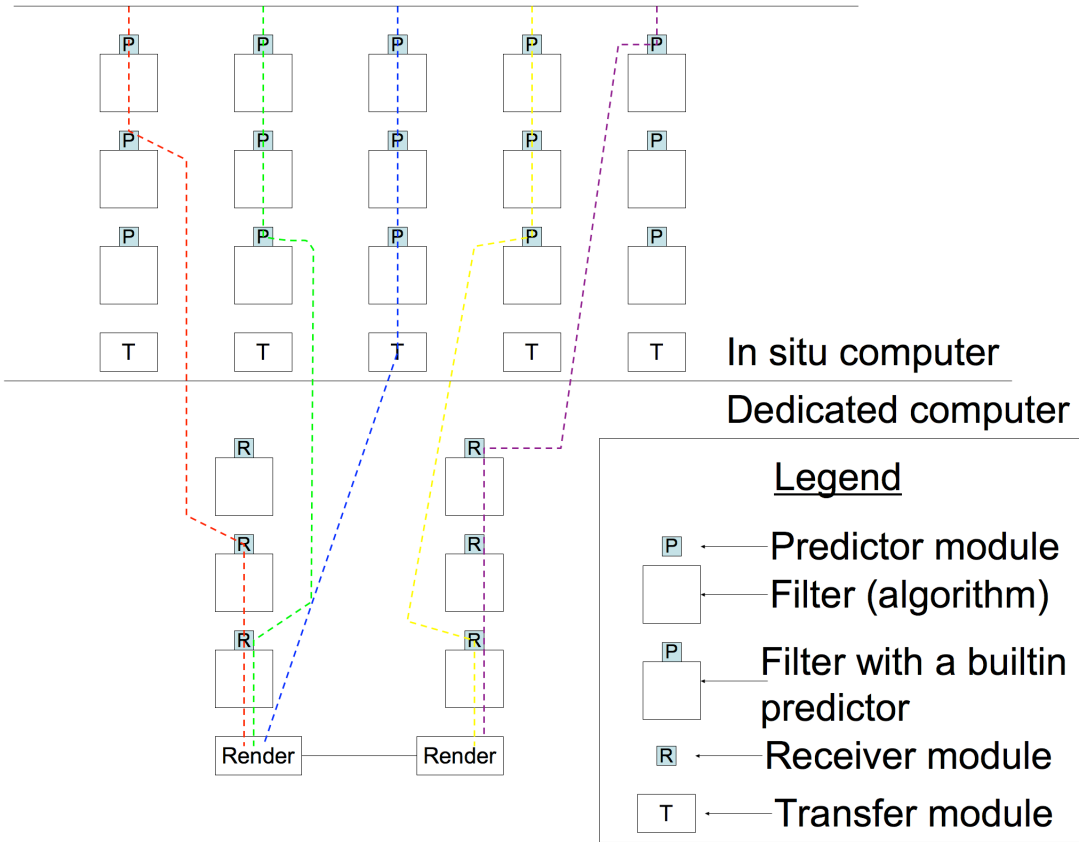

Figure 2

In this example there are five processors on the in situ computer and two on the dedicated computer. For the first processor (red), the first filter's predictor returns false, allowing that domain to remain on the in situ computer. It then executes the first filter. The second filter's predictor returns true, meaning that the data should be sent to the dedicated computer. The dedicated computer receives this data set as input to the corresponding filter in the pipeline and processes it for the rest of the time, including rendering. For the second processor (green), the data set remains on the in situ computer for an additional filter. When it is sent to the dedicated computer, the data is processed alongside the "red" domain's output from the second filter. The rest of the processorsare similar, with a few subtleties. The third processor (blue) is allowed to move through the entire pipeline by its predictors, and thus has to use the transfer module. The fifth processor (purple) is forced by the very first predictor to send its data to the dedicated computer, meaning no work is done in situ for that domain.

Once a processor from the in situ machine sends its data to the dedicated machine, we can no longer utilize any of its processing power. Thus we value keeping data on the in situ machine for as long as possible. So the predictor module is based solely on memory footprint. Our predictor modules calculate the increased memory footprint that would come about as a result of execution. They then compare that with the current memory budget. If the new footprint would exceed the budget, then they return true, else false.

Each filter needs a specialized predictor. The predictor for the filter that calculates the surface normals to provide good shading is easy to implement. This simply

returns the three times the number of points times the size of a floating point number in bytes. The predictor for a contouring filter is much harder. We solve this problem by performing a "false contour" where statistics are taken, but no data is generated. This is obviously a non-optimal approach, but we believe it is not quite as bad as it appears at first glance because there is so much compute power on the in situ machine. More efficient and accurate predictors for various filters would be a good area of future work.

Results/Technical Outcome

We developed a prototype of the above system by modifying VisIt, a popular and open source visualization application authored primarily at Lawrence Livermore National Laboratory. We introduced multiple socket connections between VisIt compute engines, which are the processing component that is attached to the simulation code or running on the dedicated post-processing resource. We modified the core execution methodology of VisIt processing pipelines, and introduced predictors for a small set of VisIt processing modules.

These modifications entailed deeper changes in the pipeline execution algorithm than originally planned, but since this project was tasked with experimenting with a distributed in-situ design that could be used in existing post-processing tools, we felt that this extra effort was worthwhile. The research prototype executes as expected. The performance of distributed operations was slower due to the time required to send data over sockets, and to queue up socket sends for multiple processors on the compute nodes to a single post-processing node.

We performed an initial performance study on the last time step of a Rayleigh-Taylor turbulence simulation. The pipeline consisted of contouring the density field, normal vector computation, and rendering. The data set consisted of 1152x1152x1152 samples on a regular grid. The data stored in a 9x9x9 array of 128x128x128 blocks. We studied ratios of 4-to-1 and 8-to-1 between compute nodes and post-processing nodes. In these studies, the compute nodes read the data from disk instead of accessing it in memory from a simulation code. An 'available memory' parameter was coded that caused the compute nodes for some blocks to send data earlier to the post-processing nodes. In all studies of up to 72 total processors the time difference between zero available memory and no limit on available memory was negligible. The reason for this was that the small number of processors caused most of the time to execute the pipeline to be spent in data read and actual computation. The data transfer over the socket was not of the same scale. We did not have time to pursue dedicated application time to perform studies of 1000 or more processors, but we expect the difference in performance to be greater in that case, in part due to greater contention for post-processing resources, and because compute and disk read will be spread over more processors.

The system is designed to trade performance degradation for the ability to perform post-processing at all. That is, we expect that some queries will cause the in-situ computation to run up against memory requirements and thus our technique may be the only way to enable in-situ computation in those cases. Our sockets based approach may need to be modified on future architectures, in particular, replacing sockets with MPI on machines that have an integrated visualization partition.

Exit Plan

We expect to use a small amount of existing ASC Pre and Post Processing Environment funding to perform scaling studies of the research prototype. In addition, some parts of the prototype, including the socket communication infrastructure, may be introduced into the main VisIt distribution.

Summary

The successful conclusion of this project enabled an initial investigation of distributed data-flow architectures for the VisIt application framework and provided important information for future efforts in in-situ analysis and visualization on peta-scale systems.

Acknowledgements (if applicable)

We gratefully acknowledge Brad Whitlock for his advice on modifying the VisIt socket communication infrastructure.

References

[1] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. A contract-based system for large data visualization. In Proceedings of IEEE Visualization 2005, pages 190–198, Minneapolis, Minnesota, October 2005.
[2] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova. In situ processing and visualization for ultrascale simulations. Journal of Physics, 78, June 2007. (Proceedings of SciDAC 2007 Conference).
[3] T. Peterka, H. Yu, R. Ross, and K.-L. Ma. Parallel volume rendering on the ibm bluegene/p. In Proceedings of IEEE Pacific Visualization Symposium, pages 175–182, March 2008.
[4] R. Ross, T. P. H.-W. Shen, K.-L. Ma, H. Yu, and K. Moreland. Visualization and parallel i/o at extreme scale. Journal of Physics (Conference Series), 125, July 2008.