# Delivering Supercomputing to the Ultrascale

Thesis approved by
the Department of Computer Science of the University of Kaiserslautern
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)
to

**John M. Patchett**

| | |
|---|---|
| Date of Defense: | December 15, 2017 |
| Dean: | Prof. Dr.-Ing. Stefan Deßloch |
| Reviewer: | Prof. Dr.-Rer.nat. Hans Hagen |
| Reviewer: | Prof. Bernd Hamann, PhD. |
| Reviewer: | James Ahrens, PhD. |

D 386

This thesis contains 123 pages in total.

# Acknowledgments

I thank my family. My wife Rita who supported me through the process and my children Michael and Matthew who inspired me to do better. My mother, Linda Coleman who set me on path of discovery in efforts to always be more informed. My step father Ulysses Coleman, Jr. who always pushed me to do the best I could and then get prepared to do better tomorrow. My father, Walter Patchett, who has always been willing to delve into minutia and reflect openly on past experience.

I also owe great thanks to my "Doctor Father" and committee. Professor Hans Hagen encouraged me to focus and get it done, "We are not here for a vacation! We are here to work!" James Ahrens has patiently mentored me for nearly twenty years and provided plenty of encouragement and support to gain more academic knowledge of computer science and to find passion in the work I do. Professor Bernd Hamann inspired my thinking by years of counseling his own students in my presence. I learned to be aware of the importance of my work in the broader world.

Many friends and colleagues have greatly influenced my work. Al McPherson and James Ahrens cornered me in a hallway about 2004 and suggested that I would benefit from going to school in a more directed way to study computer science. This started my adventure. Jon Woodring and Boonthanome Nouansengsy have suffered my poor writing skills, more than most, and have been instrumental in the preparation of many publications. Christina Gillmann and Anne Berres from the University of Kaiserslautern provided invaluable help in navigating the University bureaucracy. Max Zeyen, Anne Berres, and Jonas Lukasczyk patiently tutored me in topology.

I would like to thank the Los Alamos National Laboratory, they have provided me with a sea of inspiration and resources to explore.

Finally, I want to thank Giorgio I. Sapadaro. Georgio taught me many things, but most importantly: plan what I want to do and collect the tools to do it.

# Short Summary

**Summary**

Computational simulations run on large supercomputers balance their outputs with the need of the scientist and the capability of the machine. Persistent storage is typically expensive and slow, its performance grows at a slower rate than the processing power of the machine. This forces scientists to be practical about the size and frequency of the simulation outputs that can be later analyzed to understand the simulation states. Flexibility in the trade-offs of flexibility and accessibility of the outputs of the simulations are critical the success of scientists using the supercomputers to understand their science. In situ transformations of the simulation state to be persistently stored is the focus of this dissertation.

The extreme size and parallelism of simulations can cause challenges for visualization and data analysis. This is coupled with the need to accept pre partitioned data into the analysis algorithms, which is not always well oriented toward existing software infrastructures. The work in this dissertation is focused on improving current work flows and software to accept data as it is, and efficiently produce smaller, more information rich data, for persistent storage that is easily consumed by end-user scientists. I attack this problem from both a theoretical and practical basis, by managing completely raw data to quantities of information dense visualizations and study methods for managing both the creation and persistence of data products from large scale simulations.

# Summary of Thesis

**Abstract**

Computational simulations run on large supercomputers balance their outputs with the need of the scientist and the capability of the machine. Persistent storage is typically expensive and slow, its performance grows at a slower rate than the processing power of the machine. This forces scientists to be practical about the size and frequency of the simulation outputs that can be later analyzed to understand the simulation states. Flexibility in the trade-offs of flexibility and accessibility of the outputs of the simulations are critical the success of scientists using the supercomputers to understand their science. In situ transformations of the simulation state to be persistently stored is the focus of this dissertation.

The extreme size and parallelism of simulations can cause challenges for visualization and data analysis. This is coupled with the need to accept pre partitioned data into the analysis algorithms, which is not always well oriented toward existing software infrastructures. The work in this dissertation is focused on improving current work flows and software to accept data as it is, and efficiently produce smaller, more information rich data, for persistent storage that is easily consumed by end-user scientists. I attack this problem from both a theoretical and practical basis, by managing completely raw data to quantities of information dense visualizations and study methods for managing both the creation and persistence of data products from large scale simulations.

**Summary Of My Results**

Starting with Chapter 2 only new results will be presented and these results will now be summarized here. publications [1, 2]

**Chapter 2: In Situ Visualization with Ghost Cells**

Ghost cells are important for distributed memory parallel operations that require neighborhood information, and are required for correctness on the boundaries of local data partitions. Ghost cells are one or more layers of grid cells surrounding the local domain, which are owned by other data partitions. They are used by the local domain when neighbor information is required.

Finding ghost cells in structured data is trivial and can normally be done by calculating on cell indices. Obtaining ghost cells for unstructured grid data, however, is a nontrivial task. It requires an analysis of the connectivity of the grid in order to find neighbor cells. When the grid is distributed, the operation is further complicated by the need to determine which processes own the neighbor cells, and coordinating communication with them. This is a problem when operating on unstructured grid data sets that do not already have ghost cells. Parallel visualization algorithms will usually assume that a cell does not exist if it is not in the local data partition. Without ghost cells, this leads to operations that need neighborhood information, such as point data interpolated from cell data, being calculated incorrectly at partition boundaries. Production visualization tools generally support the existence of ghost cells, but not their generation, especially for unstructured grids. In the literature, there is no documented algorithm for generating in parallel one or more layers of ghost cells for unstructured grid data.

A new algorithm to compute ghost cells in parallel on distributed unstructured data

sets is presented. Given a partitioned data set without ghost cells, this algorithm is capable of producing any number of layers of ghost cells necessary to support parallel operations. Performance results and timing comparisons to ParaView's D3 filter are presented. A number of optimizations to the algorithm are also presented, with a space and performance analysis showing the benefits and trade-offs enabled with this ghost cell generator. The algorithm is currently available in the Visualization Toolkit (VTK) and ParaView as the vtkPUnstructuredGridGhostCellsGenerator filter. Publications [3, 4]

**Chapter 3: In Situ and Post Processing Work Flows Case Study: Asteroids** Simulation scientists need to make decisions about what and how much output to produce. They must balance their ability to efficiently ingest the analysis with their ability to get more analysis. This balance as a trade-off between flexibility of saved data products and accessibility of saved data products is studied in this chapter. One end of the spectrum is raw data that comes directly from the simulation, making it highly flexible, but inaccessible due to its size and format. The other end of the spectrum is highly processed and comparatively small data, often in the form of imagery or single scalar values. This data is typically highly accessible, needing no special equipment or software, but lacks flexibility for deeper analysis than what is presented. A user driven model and analysis that considers the scientists' output needs in regards to flexibility and accessibility is presented. This model allows for the analysis of a real-world example of a large simulation, lasting months of wall clock time, on thousands of processing cores. Though, the ensemble of simulation's original intent was to study asteroid generated tsunamis, the simulations are now being used beyond that scope to study the asteroid ablation as it moves through the atmosphere. With increasingly large supercomputers, designing work flows that support an intentional and understood balance of flexibility and accessibility is necessary. In this chapter, a new strategy is presented, developed from a user driven perspective to support the collaborative capability between simulation developers, designers, users and analysts to effectively support science by wisely using both computer and human time. Publications [5–8]

**Chapter 4: Optimizing File Access Patterns using Spatio-Temporal Parallelism**
For many years now, I/O read time has been recognized as the primary bottleneck for parallel visualization and analysis of large-scale data. In this chapter, a model that can estimate the read time for a file stored in a parallel file system when given the file access pattern is presented. The file access pattern will be dictated by the type of parallel decomposition used. Spatio-temporal parallelism, which combines both spatial and temporal parallelism, to provide greater flexibility to possible file access patterns is employed. This work enables the configuration of the spatio-temporal parallelism to design optimized read access patterns that result in a speedup factor of approximately 400 over traditional file access patterns. Publications [9]

**Chapter 5: Cinema** Extreme scale scientific simulations are leading a charge to exascale computation, and data analytics runs the risk of being a bottleneck to scientific discovery. Due to power and I/O constraints, it is expected that in situ visualization and analysis will be a critical component of these work flows. Options for extreme scale data analysis are often presented as a stark contrast: write large files to disk for interactive, exploratory analysis, or perform in situ analysis to save detailed data about phenom-

ena that a scientists knows about in advance. A novel framework for a third option is presented– a highly interactive, image-based approach that promotes exploration of simulation results, and is easily accessed through extensions to widely used open source tools. This in situ approach supports interactive exploration of a wide range of results, while still significantly reducing data movement and storage. Publications [10–12]

**Chapter 6: Analysis Driven Refinement**
Prioritization of data is necessary for managing large-scale scientific data, as the scale of the data implies that there are only enough resources available to process a limited subset of the data. For example, data prioritization is used during in situ triage to scale with bandwidth bottlenecks, and used during focus+context visualization to save time during analysis by guiding the user to important information. In this chapter, *ADR* visualization, a generalized analysis framework for ranking large-scale data using Analysis-Driven Refinement (ADR), which is inspired by Adaptive Mesh Refinement (AMR) is presented. A large-scale data set is partitioned in space, time, and variable, using user-defined importance measurements for prioritization. This process creates a *prioritization tree* over the data set. Using this tree, selection methods can generate sparse data products for analysis, such as focus+context visualizations or sparse data sets. Publication [13]

# Core Thesis References


- John Patchett, Boonthanome Nouanesengsy, Joachim Pouderoux, James Ahrens, and Hans Hagen. Parallel multi-level ghost cell generation for distributed unstructured grids. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on*. IEEE, 2017

- John M Patchett, Boonthanome Nouanesengsy, G Gisler, J Ahrens, and H Hagen. In situ and post processing workflows for asteroid ablation studies. In *Eurographics Conference on Visualization (EuroVis)*, 2017

- John M Patchett, Boonthanome Nouanesengsy, James Paul Ahrens, Michael Kenneth Lang, David Honegger Rogers, Jennifer Kathleen Green, Francesca Samsel, Giovanni Antonio Cone, and Hans-Jurgen Hagen. Delivery of in situ capability to end users. *Visualization in Practice*, 2017

- Boonthanome Nouanesengsy, John Patchett, James Ahrens, Andrew Bauer, Aashish Chaudhary, Ross Miller, Berk Geveci, Galen M Shipman, and Dean N Williams. A model for optimizing file access patterns using spatio-temporal parallelism. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 4. ACM, 2013

- Jonathan Woodring, Mark Petersen, Andre Schmei$\beta$er, John Patchett, James Ahrens, and Hans Hagen. In situ eddy analysis in a high-resolution ocean climate model. *IEEE transactions on visualization and computer graphics*, 22(1):857–866, 2016

- John Patchett, Boonthanome Nouanesengsy, Patricia Fasel, and James Ahrens. 2016 CSSE L3 milestone: Deliver in situ to XTD end users. Technical report, Los Alamos National Laboratory, 2016. LA-UR-16-26987

- John Patchett, Francesca Samsel, Karen Tsai, Galen Gisler, David Rogers, Greg Abram, and Terece Turton. Visualization and analysis of threats from asteroid ocean impacts. 2016. Winner, Best Scientific Visualization & Data Analytics Showcase; LA-UR-16-26258

- Francesca Samsel, John M Patchett, David Honegger Rogers, and Karen Tsai. Employing color theory to visualize volume-rendered multivariate ensembles of asteroid impact simulations. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1126–1134. ACM, 2017

- James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press

- Stephen Hamilton, Randal Burns, Charles Meneveau, Perry Johnson, Peter Lindstrom, John Patchett, and Alexander S Szalay. Extreme event analysis in next

generation simulation architectures. In *International Supercomputing Conference*, pages 277–293. Springer, 2017

- D. Williams, C. Doutriaux, J. Patchett, S. Williams, G. Shipman, R. Miller, C. Steed, H. Krishnan, C. Silva, A. Chaudhary, P. Bremer, D. Pugmire, W. Bethel, H. Childs, M. Prabhat, B. Geveci, A. Bauer, A. Pletzer, J. Poco, T. Ellqvist, E. Santos, G. Potter, B. Smith, T. Maxwell, D. Kindig, and D. Koop. The ultra-scale visualization climate data analysis tools (UV-CDAT): Data analysis and visualization for geoscience data. *Computer*, PP(99):1–1, 2013

- J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012

- Boonthanome Nouanesengsy, Jonathan Woodring, John Patchett, Kary Myers, and James Ahrens. ADR visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 43–50. IEEE, 2014

- John Patchett, James Ahrens, Boonthanome Nouanesengsy, Patricia Fasel, Patrick O'leary, Chris Sewell, Jon Woodring, Christopher Mitchell, Li-Ta Lo, Kary Myers, Joanne Wendelberger, Curt Canada, Marcus Daniels, Hilary Abhold, and Gabe Rockefeller. Case study of in situ data analysis in ASC integrated codes, 2013. LA-UR-13-26599

# Contents

# Chapter 1

# Introduction

In situ is the Latin word for in place, that is in its original or natural position. In the realm of supercomputing and simulation science, in situ refers to the production of analysis products at or around the time the simulation code is running on the supercomputer, while the data is resident in the memory where the simulation has operated on it. In place processing to create analysis products is in contrast to a traditional post hoc method as can be seen in Figure 1.2. Post hoc processing is a traditional method, where relatively unprocessed data representing the simulation state is stored on a persistent file system and later back into a computer's memory to be analyzed. Institutions that are driving supercomputing hardware and their associated simulation teams are increasingly becoming concerned with their ability to perform in in situ visualization and analysis so that simulations can take advantage of burgeoning hardware, this is evident in a recent level three milestone (L3) for the Advanced Simulation and Computing Program (ASC) at the Los Alamos National Laboratory (LANL) [7]. Much of the work presented here is driven by the needs of such programs.

Some nuances of in situ exist. Emerging memory systems called burst buffers are disk arrays composed of solid state drives (SSD) that allow for very fast random access reads. If a simulation were to drop raw data to one of these file systems there would be an opportunity to use another parallel system [12] not sharing the same memory with the simulation compute nodes, to quickly read data into its memory and produce analysis products. Figure 5.10 shows the results of our experimental setup for that work. On the left, Figure 1.1a shows the traditional post hoc processing where the simulation runs (red) stops and drops persistent data files representing the simulations state (green) then


(a) Direct to Lustre

(b) Using Burst Buffers

Figure 1.1: Plots showing the improvement by pipelineing in situ activities with an extended memory hierarchy in the form of a burst buffer.

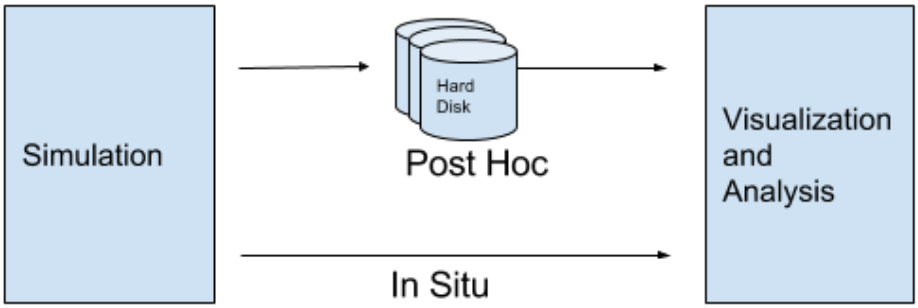



Figure 1.2: The in situ work flow does not use an intermediate file system, like post hoc, to allow for later analysis. In situ implies that visualization and data analysis products are produced at runtime of the simulation on or near the compter that is running the simulation.

moves on to post process those file and produce visualizations products(blue). Adding a much faster file system in the form of a burst buffer composed of SSD drives, figure 1.1b shows a much faster write(magenta), another copy from the burst buffer to the Lustre file system (green) and finally visualization products generated on another set of computers getting the data directly from the burst buffer (blue). This type of system allows for pipelining of the compute with the analysis production and, in this case, the total job is completed 33% faster. File systems create an additional layer of complexity in that they are very fast if they are written to and read from in particular ways. Frequently domain decompositions proper for visualization read files in different ways than they were intended from the file system perspective and incredibly slow reads can result [9]

An analysis product is the persistent storage of representations of the simulation state. The persistent representations might include checkpoint-restart dumps which can be used to restart the simulation from that state or to be inspected and analyzed to gain understanding of the simulation at that state. Visualization imagery could also be generated based on the simulation state that exists in the memory of the supercomputer at the time it exists in that memory. This type of analysis generates a set of potential problems for visualization software that is typically responsible for decomposing data sets in a post hoc way. Plots, histograms, and other types of two dimensional analysis products can also be made during the runtime of the simulation which can help domain scientists to evaluate the simulation in progress. Domain scientists at the Los Alamos National Laboratory have been slow to adopt in situ analysis and visualization as a standard mode of operation for multiple reasons. These reasons include a lack of formal support in the production environments, a perception that it adds complications to their already complicated work flows and difficulty within the code development teams who are already burdened with excessive amounts of work and code maintenance. Delivering in situ to LANL end users then necessitates solving these problems in demonstrable ways.

It is the role of the Data Science at Scale (DSS) team at the Los Alamos National Laboratory(LANL) to ensure the supercomputing systems are properly positioned and can be leveraged to not only run simulations, but they can be used to extract results from the simulations to help domain scientists answer the questions that are causing them to run the simulations in the first place. To that end the DSS team looks at possible futures and strategically performs research and develops capability that can be quickly

turned into production. Production systems like CEI, Inc's Ensight is a commonly used production visualization tool that LANL has purchased and maintains to meet the needs of production users. Increasingly rapid development can more easily occur in open source software packages like ParaView [16] which can be deployed with very current research algorithms and capabilities, yet also be maintained as a core functional piece of usable software. The DSS team also works to ensure that future possible software packages like UVCDAT [1] get developed so they can be quickly made available to users who need them. There are many other software packages and technologies that get followed, but what is presented in this document are efforts that the DSS team, ASC, and particularly John Patchett apply human resource efforts to ensure their success.

## 1.1 Data Analysis and Visualization

Producing analysis products while the simulation is running can be thought of as sensors in real world experiments. Physical experiments employ sensors: cameras, thermometers, potentiometers, they measure things like magentic fields, gravity, moisture, vibrations, etc. These produce streams of data that scientists are accustomed to receiving and analyzing. Similarly, computational simulations can produce similar outputs rather than simply saving large spatially dense samples, at low time frequencies. Using in situ allows a simulation scientist to recognize the sampling nature of simulation outputs and begin to control those sampling rates to meet their needs. Given that domain scientists run a simulation for a reason, they can customize the outputs in regards to type and resolution to ensure that the questions causing them to run the simulations can be properly answered with the outputs of the simulations. Indeed, well designed simulation experiments could go even further to produce the actual answers being sought. Visualization itself is rendering intensive in order to turn large quantities of triangles into pixels on displays or in images. Rendering has traditionally been done using specialized graphics hardware in the form of CPUs, though as processing speeds have been increasing enabling infrastructure work was being conceived and developed to allow CPU based rendering [17], [4].

## 1.2 Data Reduction

In situ gives the promise of data reduction. At one extreme a simulation state could be saved each and every time that simulation state changes. More reasonably, a simulation state could be preserved at each completed interval or time step, but this also is problematic for high resolution simulations with very small time steps that frequently run at tens to thousands or even more time step cycles.

A recent simulation run at LANL of an asteroid impacting deep ocean water ran nearly 50,000 time step cycles for 90 seconds worth of simulated time. This simulation averaged almost 200 million variable unstructured grid cells per time step with eleven, four byte floating point values per cell. The Grid representation itself required over 6 gigabytes just for indices to the 200 Mega Points to define each cell. The 200 Mega Points, defined by three floating point numbers each, is 3.6GB. Finally 200 Mega Cells * 11 four byte floating point numbers is 44 bytes per cell times 200 Mega Cells is 2.4 gigabytes so nearly 17 gigabytes per time step. 17 GB/time step * 50,000 time steps would be be around 850 terabytes of data. Of course an ensemble of 12 simulations was run and

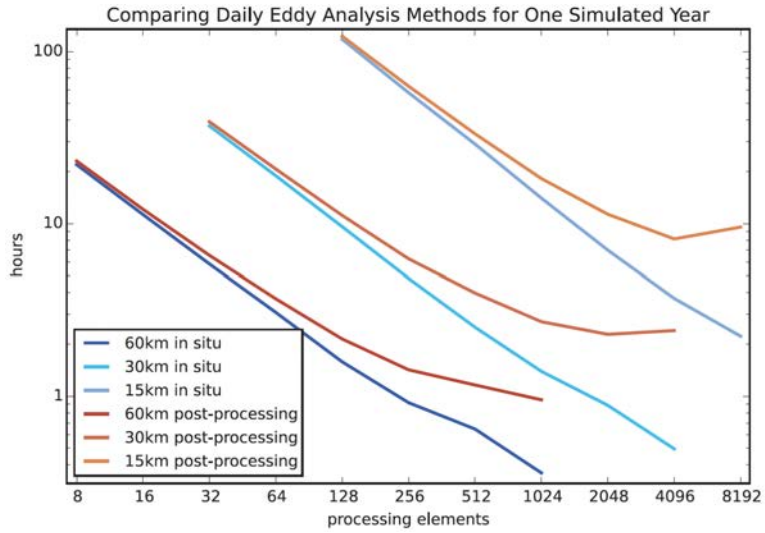



Figure 1.3: Comparison of 3 global ocean model sizes on log-log plot for performing daily eddy analysis across appropriate processor size groupings. Notice the post hoc method does not scale. This is probably true for more than just global ocean models.

if this is representative that would end up at over 10 PetaBytes of data. Alternatively, each point in the simulation requires roughly 88 bytes to store all of its information. The yA31 data set total points is a little over 11 trillion cells. This rough calculation would bring us very near 1 Peta Byte of data to be stored on disk per simulation.

Not only is the storage of such vast quantities of data unreasonable, the time to store and even worse, retrieve such quantities is not tractable given current storage systems. This means trade-offs must be made. For years simulation scientists have been sampling time steps. That is, they save every nth time step or more commonly every regular interval of simulated time. The frequencey of saves has decreased in relation to model size as supercomputers have been growing at faster rates than their associated file systems. In situ gives simulation scientists the opportunity to sample at different rates on different axes to control the total data being stored to represent their simulation. Flexibility created by in in situ libraries like ParaView Catalyst and VisIT libsim allow domain scientists to pick and choose data products that meet their needs while managing total quantities and times to produce the data products stored on disk.

Not all data reduction exercises need to be complicated and domain specific. De facto file formats used by ParaView such as the XML VTK file formats allow for zlib compression, simply by adding a single line of code to turn on the compression the vtkUnstructuredGridWriter object. Other types of lossless compression could also be developed and injected into the work flow. Lossy compression, like wavelets [18] , could also be employed. This type of compression though lossy, works very well on simulations that have continuous scalar fields, it also enables other algorithms like Level of Detail, and demand driven capability where only the data that matters to a given visualization is supplied to the visualization pipeline.

Data Reduction systems can also be developed where feature catalogs are generated

from the native data and that catalogs are stored rather than the raw data. An example of feature catalogs can be found in our recent work [6] on identifying and counting ocean eddies and comparing the in situ work flow with a post hoc work flow. The sensibility of performing the eddy finding and cataloging in situ rather than post hoc is clearly shown in figure 1.3. This plot from the above cited work is comparing daily eddy analysis over one simulated year for both the the in situ and post hoc methods. Each pair of lines shows the hours it takes to perform the analysis for a given size global ocean model run on varying number of processors. The plot is log-log. Notice the upturn in the performance for larger core counts. The in situ capability shows no such upturn and scales well with the simulation code itself. It is performance issues in the file IO that create such poor performance for the post hoc paradigm. It is worth noting that feature identification and acceptance is a process that starts with recognizing the features, evaluating their importance to the community and for evaluation such as was done with our ocean eddy work [19].

## 1.3 Data Triage

The ability to make decisions about data that is important and data that is not important is a burgeoning field of necessity. Statistical methods applied to raw simulation data in situ can be used to show that the simulation state maintained itself between data product generation events. It can also be used to dynamically generate more output when the state is changing rapidly or in predefined unexpected ways. Triage techniques could also be used to select spatial regions within a simulation and not repeatedly store unchanging spatial regions. Techniques like this could increase the information content in the data that is saved while reducing the total quantity of data being saved. A method for choosing a camera position to ensure that changing data in a simulation is captured in the visualizations is shown in Figure 1.4. The simulation space is divided into a set of spatial bins, each of which generates a histogram for one of the scalar fields in that bin. Successive time steps are compared. The histograms are subtracted and the magnitude difference for each bar in the histogram is summed. The total sum produces a single scalar value that can be compared against a threshold to cause a camera to move. This process could, for instance, be performed across a number of scalar fields for a number of cameras. Metrics could be developed that split and merged cameras as necessary. This type of algorithm running in an autonomous mode demonstrates a relatively inexpensive straight forward method of triaging spatial data and preserving time steps with significant change in some scalar field, while not preserving data that is not changing and maintaining a guarantee of a maximum change between data saves.

## 1.4 Summary Of My Results

Starting with Chapter 2 only new results will be presented and these results will now be summarized here.

**Chapter 2: In Situ Visualization with Ghost Cells**
Ghost cells are important for distributed memory parallel operations that require neighborhood information, and are required for correctness on the boundaries of local data partitions. Ghost cells are one or more layers of grid cells surrounding the local domain,

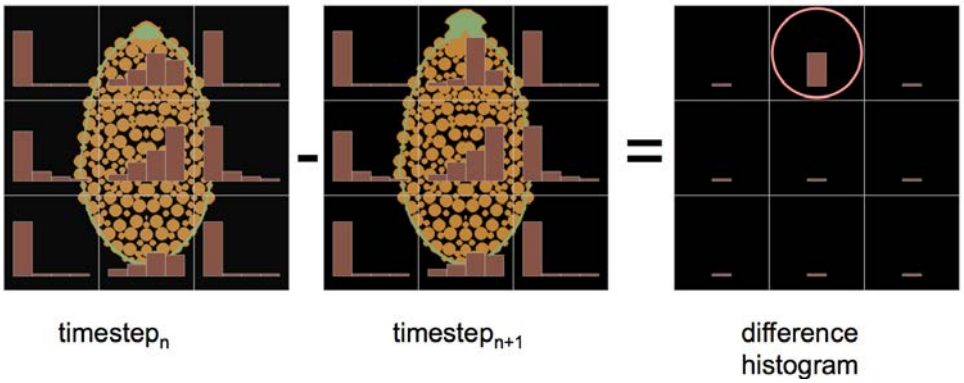



Figure 1.4: Spatial data sets can be divided into spatial bins and have histograms of their scalar values generated for each bin. These histograms can be compared between time steps using a simple subtraction of the histogram and then summing the difference magnitudes. Where the differences are non zero, the data has changed, if the sum meets a threshold, events can be triggered such as moving a camera to that position.

which are owned by other data partitions. They are used by the local domain when neighbor information is required.

Finding ghost cells in structured data is trivial and can normally be done by calculating on cell indices. Obtaining ghost cells for unstructured grid data, however, is a nontrivial task. It requires an analysis of the connectivity of the grid in order to find neighbor cells. When the grid is distributed, the operation is further complicated by the need to determine which processes own the neighbor cells, and coordinating communication with them. This is a problem when operating on unstructured grid data sets that do not already have ghost cells. Parallel visualization algorithms will usually assume that a cell does not exist if it is not in the local data partition. Without ghost cells, this leads to operations that need neighborhood information, such as point data interpolated from cell data, being calculated incorrectly at partition boundaries. Production visualization tools generally support the existence of ghost cells, but not their generation, especially for unstructured grids. In the literature, there is no documented algorithm for generating in parallel one or more layers of ghost cells for unstructured grid data.

A new algorithm to compute ghost cells in parallel on distributed unstructured data sets is presented. Given a partitioned data set without ghost cells, this algorithm is capable of producing any number of layers of ghost cells necessary to support parallel operations. Performance results and timing comparisons to ParaView's D3 filter are presented. A number of optimizations to the algorithm are also presented, with a space and performance analysis showing the benefits and trade-offs enabled with this ghost cell generator. The algorithm is currently available in the Visualization Toolkit (VTK) and ParaView as the vtkPUnstructuredGridGhostCellsGenerator filter.

**Chapter 3: In Situ and Post Processing Work Flows Case Study: Asteroids** Simulation scientists need to make decisions about what and how much output to produce. They must balance their ability to efficiently ingest the analysis with their ability to get more analysis. This balance as a trade-off between flexibility of saved data products and accessibility of saved data products is studied in this chapter. One end of the spectrum is raw data that comes directly from the simulation, making it highly flexible,

but inaccessible due to its size and format. The other end of the spectrum is highly processed and comparatively small data, often in the form of imagery or single scalar values. This data is typically highly accessible, needing no special equipment or software, but lacks flexibility for deeper analysis than what is presented. A user driven model and analysis that considers the scientists' output needs in regards to flexibility and accessibility is presented. This model allows for the analysis of a real-world example of a large simulation, lasting months of wall clock time, on thousands of processing cores. Though, the ensemble of simulation's original intent was to study asteroid generated tsunamis, the simulations are now being used beyond that scope to study the asteroid ablation as it moves through the atmosphere. With increasingly large supercomputers, designing work flows that support an intentional and understood balance of flexibility and accessibility is necessary. In this chapter, a new strategy is presented, developed from a user driven perspective to support the collaborative capability between simulation developers, designers, users and analysts to effectively support science by wisely using both computer and human time.

**Chapter 4: Optimizing File Access Patterns using Spatio-Temporal Parallelism**

For many years now, I/O read time has been recognized as the primary bottleneck for parallel visualization and analysis of large-scale data. In this chapter, a model that can estimate the read time for a file stored in a parallel file system when given the file access pattern is presented. The file access pattern will be dictated by the type of parallel decomposition used. Spatio-temporal parallelism, which combines both spatial and temporal parallelism, to provide greater flexibility to possible file access patterns is employed. This work enables the configuration of the spatio-temporal parallelism to design optimized read access patterns that result in a speedup factor of approximately 400 over traditional file access patterns.

**Chapter 5: Cinema**

Extreme scale scientific simulations are leading a charge to exascale computation, and data analytics runs the risk of being a bottleneck to scientific discovery. Due to power and I/O constraints, it is expected that in situ visualization and analysis will be a critical component of these work flows. Options for extreme scale data analysis are often presented as a stark contrast: write large files to disk for interactive, exploratory analysis, or perform in situ analysis to save detailed data about phenomena that a scientists knows about in advance. A novel framework for a third option is presented– a highly interactive, image-based approach that promotes exploration of simulation results, and is easily accessed through extensions to widely used open source tools. This in situ approach supports interactive exploration of a wide range of results, while still significantly reducing data movement and storage.

**Chapter 6: Analysis Driven Refinement**

Prioritization of data is necessary for managing large-scale scientific data, as the scale of the data implies that there are only enough resources available to process a limited subset of the data. For example, data prioritization is used during in situ triage to

scale with bandwidth bottlenecks, and used during focus+context visualization to save time during analysis by guiding the user to important information. In this chapter, *ADR* visualization, a generalized analysis framework for ranking large-scale data using Analysis-Driven Refinement (ADR), which is inspired by Adaptive Mesh Refinement (AMR) is presented. A large-scale data set is partitioned in space, time, and variable, using user-defined importance measurements for prioritization. This process creates a *prioritization tree* over the data set. Using this tree, selection methods can generate sparse data products for analysis, such as focus+context visualizations or sparse data sets.

# Chapter 2

# In Situ Visualization with Ghost Cells

**Abstract**

Ghost cells are important for distributed memory parallel operations that require neighborhood information, and are required for correctness on the boundaries of local data partitions. Ghost cells are one or more layers of grid cells surrounding the local domain, which are owned by other data partitions. They are used by the local domain when neighbor information is required.

Finding ghost cells in structured data is trivial and can normally be done by calculating on cell indices. Obtaining ghost cells for unstructured grid data, however, is a nontrivial task. It requires an analysis of the connectivity of the grid in order to find neighbor cells. When the grid is distributed, the operation is further complicated by the need to determine which processes own the neighbor cells, and coordinating communication with them. This is a problem when operating on unstructured grid data sets that do not already have ghost cells. Parallel visualization algorithms will usually assume that a cell does not exist if it is not in the local data partition. Without ghost cells, this leads to operations that need neighborhood information, such as point data interpolated from cell data, being calculated incorrectly at partition boundaries. Production visualization tools generally support the existence of ghost cells, but not their generation, especially for unstructured grids. In the literature, there is no documented algorithm for generating in parallel one or more layers of ghost cells for unstructured grid data.

A new algorithm is presented to compute ghost cells in parallel on distributed unstructured data sets. Given a partitioned data set without ghost cells, this algorithm is capable of producing any number of layers of ghost cells necessary to support parallel operations. Performance results and timing comparisons to ParaView's D3 filter are presented. A number of optimizations to the algorithm are also presented, with a space and performance analysis showing the benefits and tradeoffs enabled with this ghost cell generator. The algorithm is currently available in the Visualization Toolkit (VTK) [20] and ParaView as the vtkPUnstructuredGridGhostCellsGenerator filter.

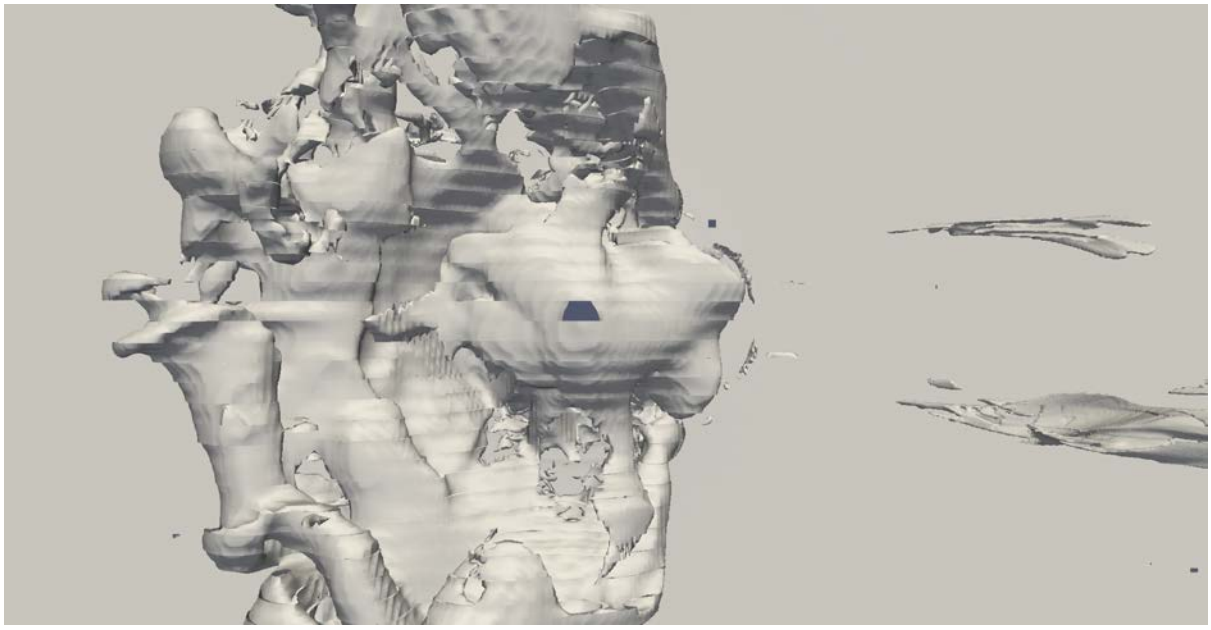


Figure 2.1: An isocontour produced on the yA31 data set with 128 processing cores and no ghost cells, there are clearly visible holes in the isocontour and seams are also visible in areas with high gradient.

## 2.1 Related Work and State of the Art

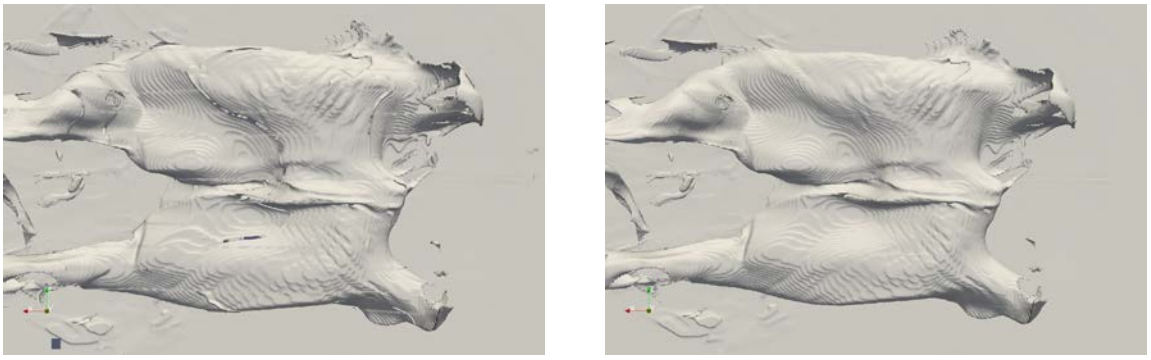
### 2.1.1 Ghost Cells

Ghost cells are necessary for parallel stencil operations on the boundaries of local domains. If ghost cells are absent, each processor will be unable to decipher whether its hull cells are part of the global hull or just the local hull. Many operations treat cells on a local boundary differently than cells on a global boundary. Algorithms, where this matters, will typically assume boundary cells are global boundary cells unless there is an indicator otherwise. Figure 2.1 shows the effects of an isocontour operation on a data set that contains no ghost cells, visualization artifacts in the form of holes in the surface and seams on boundaries are visible.

In parallel visualization systems, ghost cells are replicated cells that are owned and maintained by another processor and copied into a local domain to support stenciled calculations on the boundary. Ghost cells are cells that share points with the local domain but are owned and operated on by one or more other domains. Ahrens et al. first look at ghost cells for visualization algorithms and describe them as necessary for unstructured algorithms that require neighborhood information [21]. Ghost cells will allow the local domain to differentiate its local hull from the global hull during a parallel operation.

### 2.1.2 State of the art

Ghost cells are supported by parallel visualization pipelines like those in ParaView and VisIt. Each filter in a visualization pipeline is capable of leveraging the existence of the ghost cells to determine if the local boundary is the global boundary, and if not the global boundary, leverage the ghost cell data for stencil operations on the local boundary. Ghost



(a) No ghost cells (b) 1 layer of ghost cells

Figure 2.2: An isocontour calculated across 512 processors with and without ghost cells.

cells are passed along the visualization pipeline and leveraged when they are available. If ghost cells are not available in the data source, only expensive options exist. ParaView has an expensive D3 filter that re balances unstructured grid data across processors and can generate ghost cells. An end user could also modify a source to generate ghost cells.

The Distributed Data Filter, commonly called the d3 filter was developed at Sandia National Laboratories and has been in the VTK code base for more than a decade. It has been the state of the art for distributing and balancing vtkUnstructuredGrids across a number of processors and generating ghost cells to support stencil operations. It works by generating a parallel kd-tree to spatially balance the data set. The edges in the kd tree are used to separate the cells by assigning cells to one partition or the other based on the cell centroid.

The details of the parallel decomposition with the gay KD tree are very important to the next step of calculating ghost cells. If the cut line goes through a cell but the centroid location puts the cell in another domain, that sell would be a ghost cell for the local domain. This is the default in paraview. Include all intersecting cells is turned off. A cell is only included if the centroid is in the local domains side of the cut.

Global node IDs and cell IDs are also very important. Without global node IDs, points have to be compared to find out if they are the same point. Points on different processors have to become shared and then compared to find out if they are the same points. Redistribution and go sell calculation are combined in the case of the D3 filter. This redistribution, and with every processor having information about the local domains of every other processor. This is not the case in a pre-domain decomposed data set such as those handed to visualization algorithms for in situ use cases.

Global node and cell IDs are very useful in the general case when cells and points are being distributed even if it is just ghosts sell redistribution. This allows for easy methods to deal with duplicate cells when they show up in the local domain.

### 2.1.3 Problems with the Current State of the Art

Ghost cells are hard to get if you don't already have them. Both the ParaView and VisIt tools support the existence of ghost cells in a local domains data set. However, they don't provide efficient ways to calculate ghost cells if they don't already exist. Filters depend on previous filters in the visualization pipeline to produce ghost cells and filters will assume if they don't have the ghost cells that their local domain contains the global boundary. Users who are working with data sets that do not contain ghost cells really

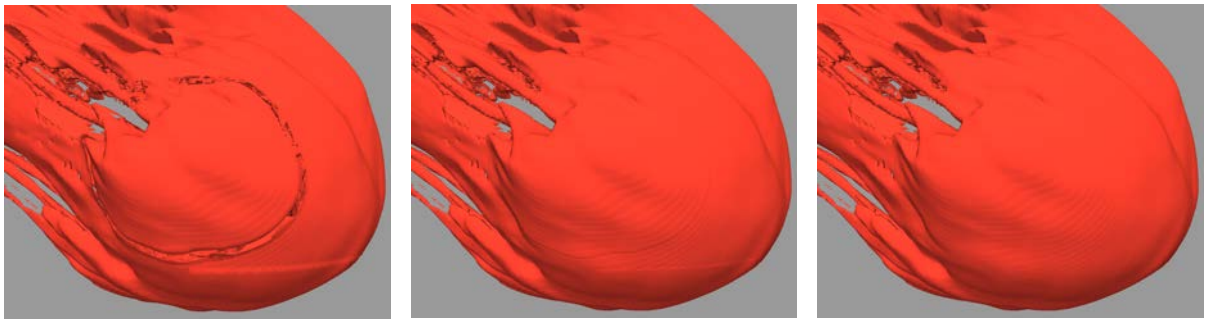



Figure 2.3: Three isocontours with no ghost cells on the left, one layer of ghost cells in the middle and two layers on the right.

have only 2 choices:

- Use the D3 filter.
- Develop ghost cell generation in the data source.

The D3 filter is expensive in both time and memory. This makes it unacceptable for many practical applications, especially for in situ, where memory is frequently limited as the simulation is also running and each individual node is already supporting two representations of the simulation grid, the simulation itself and the visualization tool. This memory expensive double grid maintenance problem could be solved with zero copy but the ghost cell problem would remain.

The other solution is to develop the source so as to produce ghost cells. This is certainly possible and probably optimal from the computation perspective. This can sometimes be difficult when simulations are written in languages unfamiliar to computer scientists and extravagant grids are defined. frequently simulations maintain their own set of ghost cells and if they ghost cells are complete, meaning every cell in the global domain that shares a point is available to the local domain there is no problem. Many simulations only maintain ghost cells for shared boundary faces, meaning that only cells with shared faces are available, not cells that share only an edge or only a point on the boundary. Developing ghost cell generation capability in every source can be expensive in terms of developer time and may even consume many lines of code that simulation developer balk at checking into their source.

Visualization without complete sets of ghost cells can leave rendering artifacts that distract from the visualization itself and can be hard to reconcile as an accurate representation of the simulation. For instance, Figure 2.1 shows an isocontour of a water volume fraction data set with no ghost cells at 0.95 after a 250 meter diameter asteroid impacts the ocean. Visualization artifacts in the form of holes and seams are clearly evident and are directly caused by the lack of ghost cells.

The D3 filter and its partitioning scheme, is initially described by Moreland, et. al. [22] to KD tree partitioning scheme was used for load balancing in the context of volume rendering in paraview. The results in this initial research were scaled to only 64 processors.

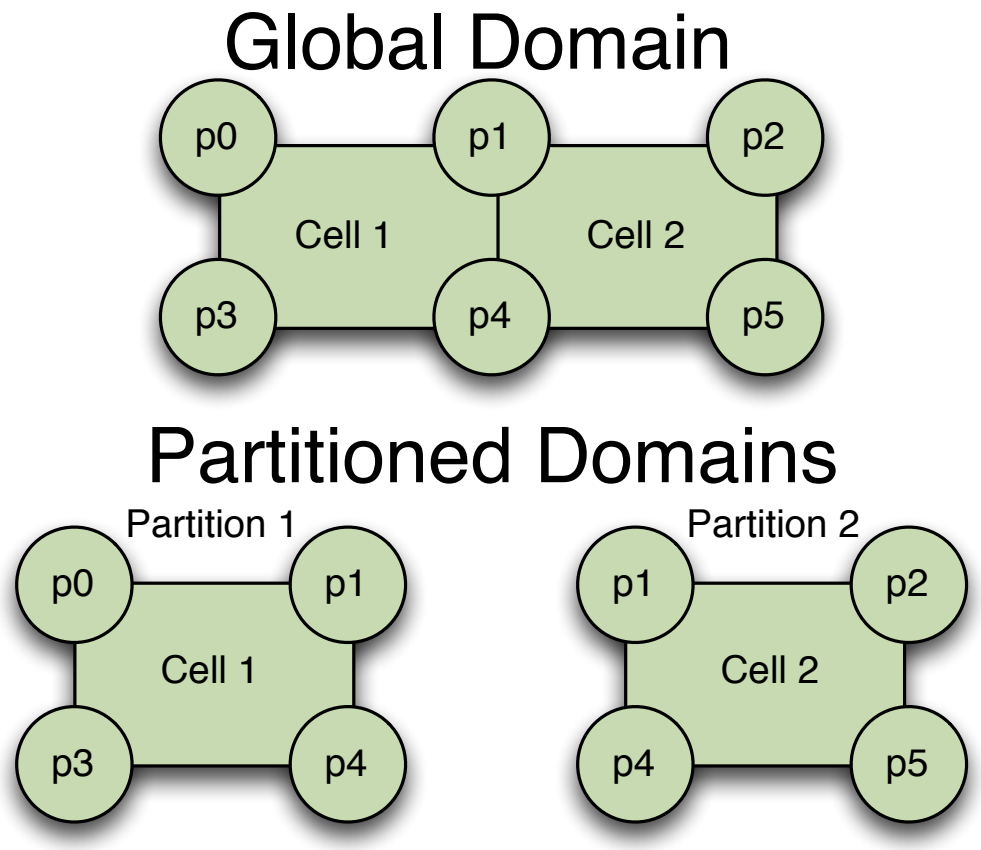



Figure 2.4: A grid partitioned into two memory spaces, faces and points on the divided boundary are replicated in both domains.

## 2.2 Solution - Basic Concept

In order to support simultaneous parallel processing of data on distributed memory supercomputers, the data often needs to be partitioned across a number of memory spaces. Each process owns one discrete partition and is responsible for all operations on that data. The partitioning generally splits on cell boundaries of the global geometry. An example simple decomposition is shown in Figure 2.4, where only two cells exist. The decomposition will assign one cell to one partition and the other cell to another partition. Note that points p1 and p4 are stored in both Partition 1 and Partition 2. Both partitions will be unaware of each other, without more information. Ghost cells solve this problem. Figure 2.5 shows the two partitions with ghost cells. Now when Partition 1 is operated on, it knows that Cell 1 shares a face and two points with Cell 2. Likewise, when Partition 2 is operated on, it will know that it shares a face and two points with Cell 1. Both partitions have sufficient information to differentiate between their local and global domains. Algorithms that require neighbor information, such as stencil operations, will have access to that information. Ghost cells support parallel operations on partitioned data, by ensuring that neighborhood information is available to all points and cells.

This chapter solves the problem of missing ghost cells in distributed unstructured grid data sets. Obtaining ghost cells for structured grids is an easier problem as ghost cells can be calculated in index space. The nature of structured grids, and one of their major

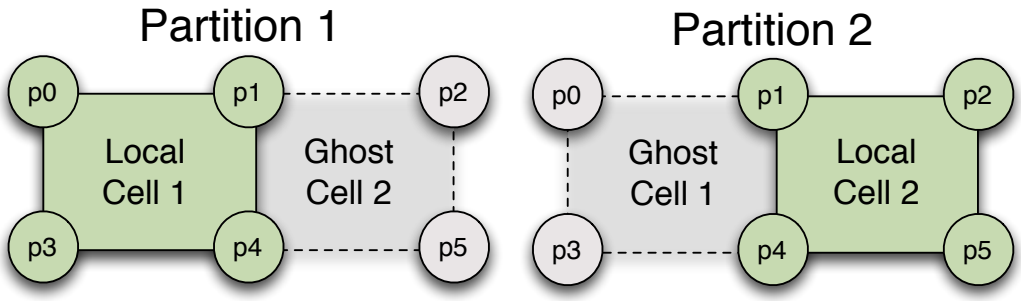



Figure 2.5: The partitioned grid with ghost cells.

advantages, is that neighbors are so easily found, even in structured parallel partitioned data. Unstructured grids, even when stored in a monolithic memory space, are difficult to decompose into well ordered, discrete partitions. A Los Alamos National Laboratory (LANL) simulation code, xRage [23], drove this work during in situ adaptor development. The first attempt at obtaining ghost cells was written in the in situ adaptor itself. During development, it was realized that the problem being solved was a rather complex ghost cell generation problem in the native simulation code. Such a complex task would need to be repeated for any future simulation codes needing in situ adapters. The need for multiple layers of ghost cells was also identified, after witnessing rendering artifacts in large parallel partitioned data. The data sets from xRage come pre-partitioned based on the memory contents of each processor at the time the grid and data is translated from the native tree adaptive mesh refinement (AMR) of xRage to VTK unstructured grids. These unstructured grids can be either operated on in situ or saved to disk. The quantity of partitions in the xRage output can number into the tens of thousands. The partitions do not come with ghost cells, but many of the desired visualization and analysis operations require ghost cells for accurate results.

Figure 3.1 shows the effects of an isocontour operation exposing an asteroid speeding through the earth's atmosphere, extracted from a member of the Deep Water Ocean Impact data set [24]. The figure shows the resulting isocontour when using no ghost cells, one layer of ghost cells and two layers of ghost cells. These three images expose the problems with not using enough ghost cell layers to supply neighborhood information across partition boundaries. The leftmost image, containing no ghost cells, shows a clear break in the isocontour, making it topologically incorrect. In addition, rendering artifacts are visible along the partition boundaries of the middle image containing only one ghost layer. The image on the right, using two layers of ghost cells, is correct, with no artifacts due to the parallel partitioning.

Ghost cells for unstructured grids are generally supported by parallel visualization pipelines like those in ParaView [25] and VisIt [26]. Support for creating ghost cells when they are missing is lacking. ParaView does provide the Data Decomposition (D3) [22] filter, which is primarily used to redistribute and repartition the grid, but can provide ghost cells in the output. D3 can be an expensive operation, as it is a complete redistribution of the global grid across the processor domains, and it is known to not scale well. Even though D3 performs data redistribution, the ghost cell generator here does not. The ghost cell generator algorithm is compared to the D3 filter, as it is currently the only mechanism in ParaView to obtain ghost cells for distributed unstructured grids. Alternatively, the source of the data can be modified to produce ghost cells. This would be

required for every unique source. Our solution simplifies the ghost cell generation process by making it feasible to simply use a general solution. In situations where more speed is necessary, a custom solution closer to the data source could be developed. To summarize the state of the art, the options are to use the D3 filter in ParaView, or develop code to make the source data provide ghost cells.

This work contribute a solution to the general unstructured grid, missing ghost cell problem, by developing an algorithm and an implementation in VTK that produces parallel multi-level ghost cells for unstructured grids. A description of the algorithm and the implementation is provided in this work. In addition, a number of algorithmic improvements to the general algorithm are described. Performance measurements on a LANL supercomputer using real-world data is provided as evidence of the efficacy of our solution.

## 2.3 Motivation Leading To Our Method and Our Results

The Paraview manual [27] contains a simple introduction to ghost cells using a two dimensional partitioned unstructured grid for an extract surface algorithm. This introduction is built upon, with more sophisticated examples of the need for ghost cells. Ghost cells are necessary for a number of reasons when working with parallel partitioned data sets. Ghost cells offer the availability of neighborhood information across partition boundaries. They allow operations on individual partitions without needing extra communication, when global versus local boundary differentiation needs exist, or when neighbor data is needed for stencil operations.

One common operation which requires neighborhood information is calculating point data from cell data (cell-to-point operation). Many visualization and analysis operations, such as isocontouring, require point data. When only cell data is available, as could be the case with output from a simulation, point data must be derived from the available cell data. The standard way to calculate point data from cell data is to find all cells that use a point as a vertex, and calculate the average over all those cells (see Figure 2.6).

The problem with partitioned data sets when performing cell-to-point calculations is that points on the edge of the local domain do not have access to all their neighbor cells. A single layer of ghost cells solves this problem by ensuring that every point in the partition has access to every cell that uses that point as a vertex. Figure 2.6 shows a simple case of calculating point data from cell data. The partition has access to every cell that contains the point P1 as a vertex. The red points on the local domain's external surface, without extra information, are unaware of their status as a local or a global domain boundary.

In real world implementations of operations requiring neighborhood information, the typical behavior is to assume that any needed ghost cells are already available. When ghost cells are missing, this can lead to problems for points along boundaries, since different partitions will calculate various point values based on the cells that are local to that partition.

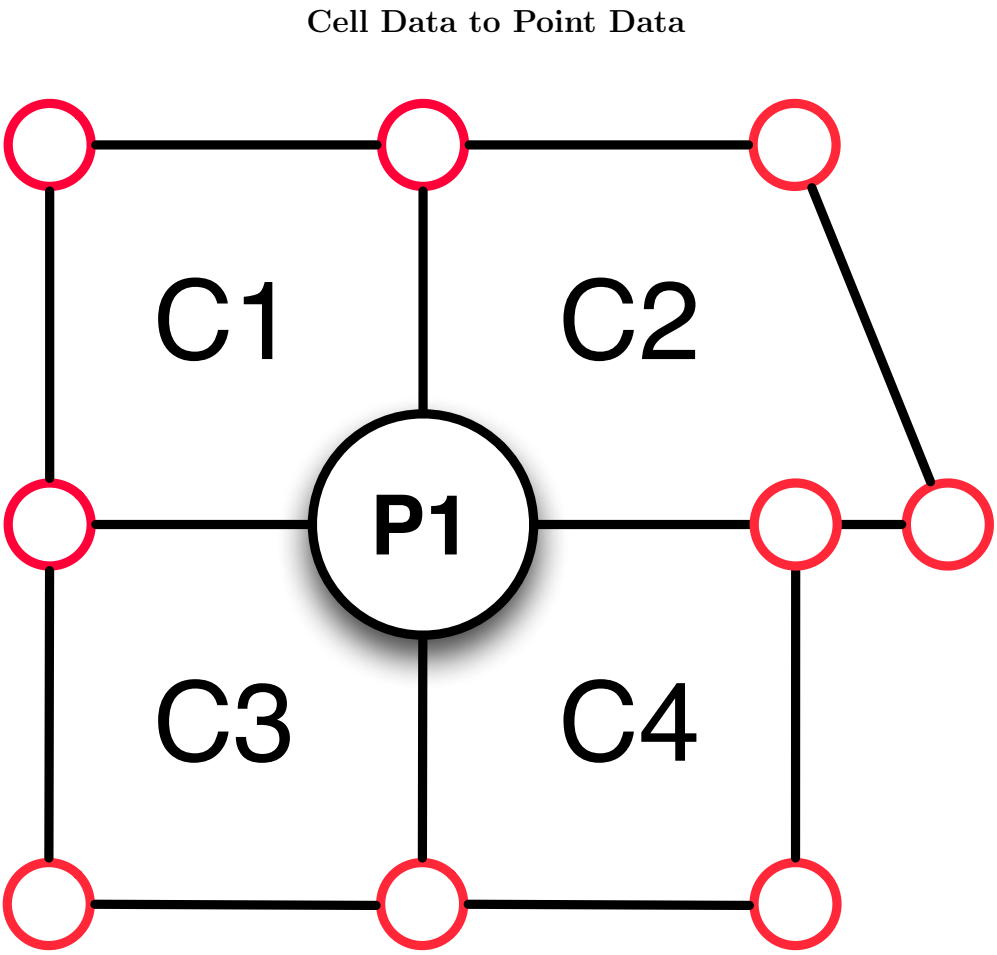



Figure 2.6: These four cells in a partition without ghost data can calculate point data correctly only at point P1, since all cells that use that point as a vertex are available. The points on the partition boundary (in red), may or may not have complete information depending on whether or not they are on the local or global boundary.

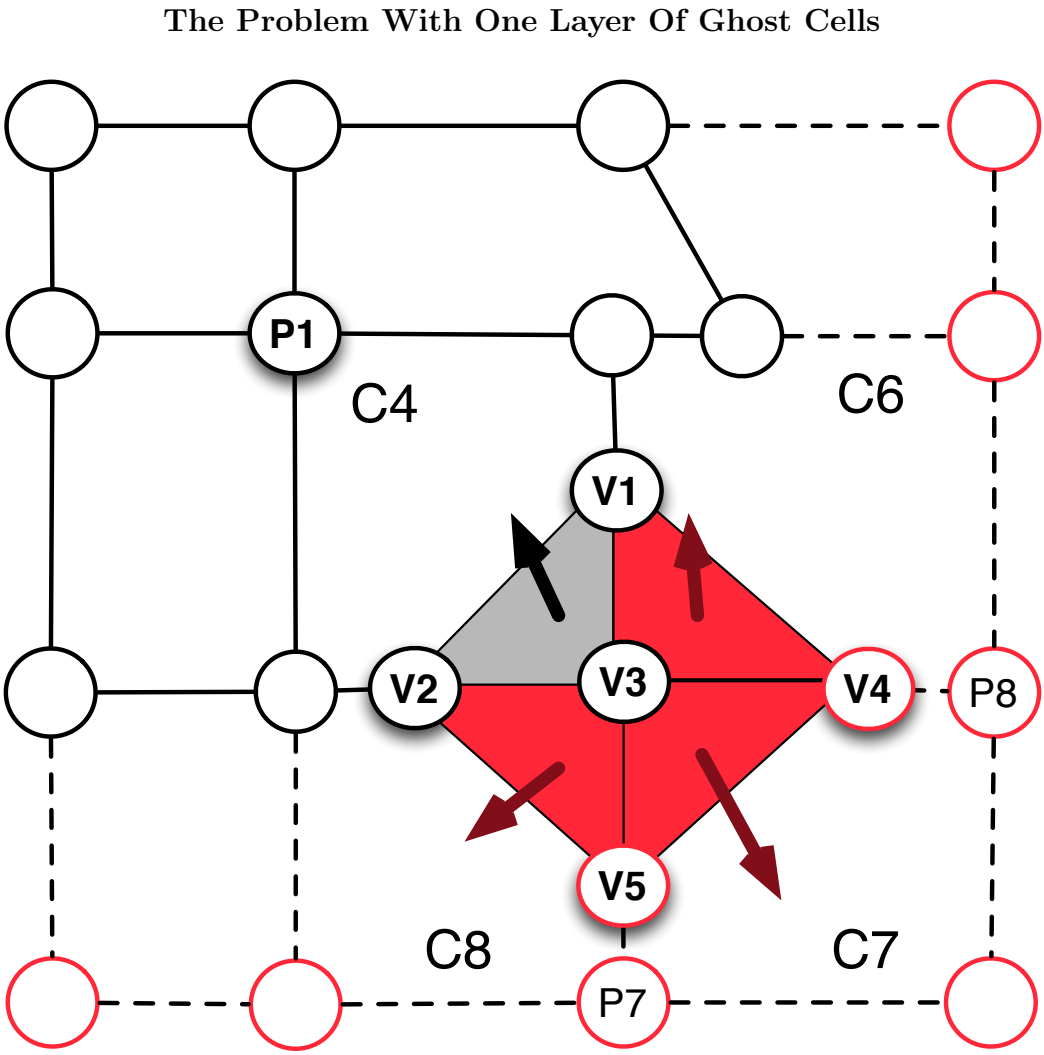



Figure 2.7: Ghost cells are indicated by dashed lines. With one layer of ghost cells, isocontouring produces correct geometry (gray triangle) in the local domain. The geometry in the ghost cells, though, is incorrect (red triangles) because the values at points P7 and P8 are faulty. The normal calculated for vertex V3 will be inaccurate, since the triangles in C6, C7 and C8 are all incorrect. This will result in lighting artifacts when the isocontour is rendered.

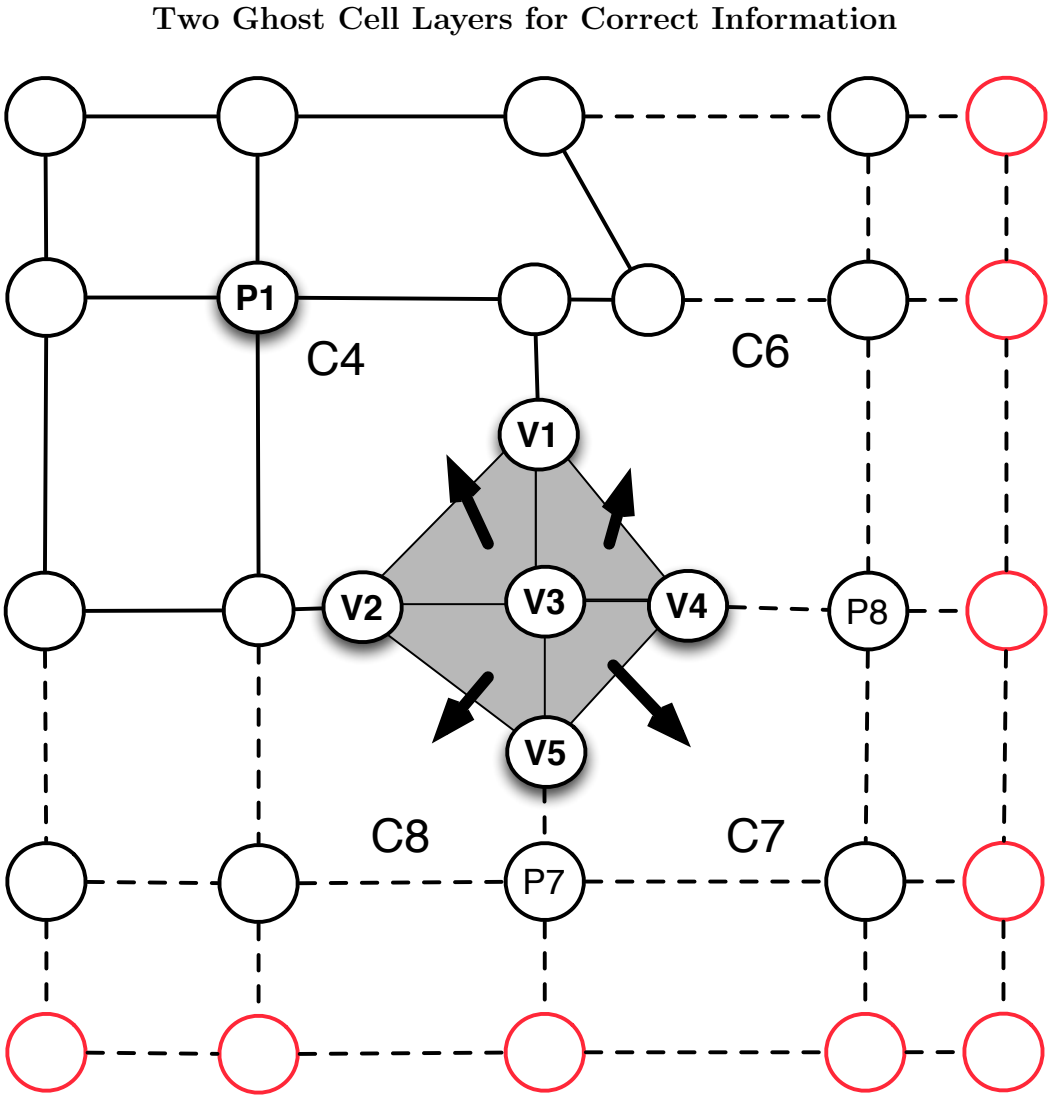



Figure 2.8: By adding a second layer of ghost cells to the one layer problem shown in Figure 2.7, correct values are calculated for points P7 and P8. This results in proper interpolation of vertices V4 and V5, making all isocontour polygons correct (gray triangles), along with their normals, in cells C6, C7 and C8. This information can then be used to produce a correct normal for vertex V3.

**Cutout of Center Isocontour from Figure 3.1**

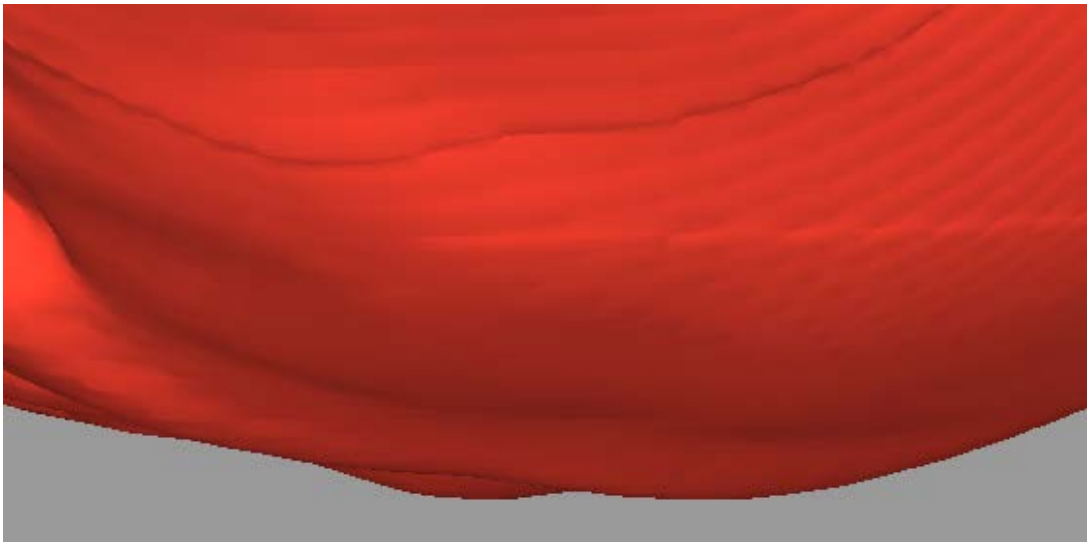


Figure 2.9: A cutout of the center image from Figure 3.1, which is an isocontour from a data set containing one layer of ghost cells. Note the lighting artifact running horizontally through the center of the image and arching across the top of the image.

### 2.3.1 Use Case: Parallel Isocontour

This section will focus on a specific use case of ghost cells, namely calculating an isocontour in parallel. In the case of a data set with only cell data, one layer of ghost cells is required to correctly compute the geometry of the isocontour. In order to obtain correct normals for points and prevent any lighting artifacts, two layers of ghost cells are necessary.

Isocontour algorithms are particularly sensitive to missing ghost cells and the resulting incorrect cell to point operation on the local domain boundaries. The most obvious error is the visual output not being topologically correct. That is an isocontour should be a continuous surface embedded in the 3D space. When grid points shared on partition boundaries are different values in different partitions, isocontour polygons which result from interpolating those points will not align. Anecdotally, this behavior is evident with increasing probability in data sets with high gradients and many partitions. Strong scaling studies performed at Los Alamos National Laboratory utilizing thousands of partitions began exposing the lack of ghost cells on data sets that only contained ghost cells on faces [11, 28].

Partitions that share the same points on boundaries must derive the same values during the cell-to-point operation. A single layer of ghost cells supplies enough information for a cell-to-point algorithm over a distributed data set to calculate the correct values for points. Figure 2.7 shows the result of using one layer of ghost cells. A topologically correct isocontour can be produced with this single layer of ghost cells. There is still insufficient information, though, to produce a partitioned geometry of the isocontour suitable for quality rendering using smooth shading. Smooth shading requires the geometry produced by the isocontour filter to have normals interpolated at the vertices. This is done in a similar way that cell-to-point is performed. Every vertex in the surface calculates the average of the normals of each polygon that shares that vertex.

When using only one layer of ghost cells, normals for the points in the isocontour will

be incorrect. This is evident in the center image from Figure 3.1 which has a closeup cutout in Figure 2.9. The problem lies with the faulty triangles that are produced within the first layer of ghost cells. This would not be a problem for rendering geometry, as all geometry produced within ghost cells can be simply discarded when rendering. The problem is that these faulty triangles are used to help calculate normals of point in the isocontour. In the same way an isocontour produces incorrect triangles on the boundary without a first layer of ghost cells, the first layer of ghost cells will produce incorrect triangles without another layer of ghost cells. This problem can be seen in Figure 2.7.

In essence, in order to obtain correct normals for all points in the isocontour, all geometry must be correct within the local domain *as well as the first layer of ghost cells.* The solution, as shown in Figure 2.8, is to add a second layer of ghost cells. Adding the second layer will result in correct geometry within the first layer of ghost cells, which will then result in correct polygon normals. Finally the proper point normal can be calculated from the polygon normals.

## 2.4 Related Work

Ahrens et al. looked at ghost cells for visualization algorithms and described them as necessary for unstructured algorithms that require neighborhood information [21]. Ghost cells will allow the local domain to differentiate its local hull from the global hull during a parallel operation.

The ParaView Guide [27] describes the ParaView data parallel computing paradigm. It provides an example of the need for ghost cells using an external faces algorithm. Without ghost cells, each processor does not contain enough information to know which parts of the external hull belong to the global hull and which belong only to the local hull. The ParaView Guide also suggests to use the D3 filter to create a distribution of unstructured cells and generate their ghost cells, when ghost cells are not present in the source. Our work builds on the existing ParaView infrastructure and provides a method to generate ghost cells for unstructured grid data sets that do not require redistribution.

The VisIt User's Manual [29] indicates there is support for ghost cells and provides a filter, the InverseGhostZone operator, that makes ghost cells visible and removes non ghost cells for the purpose of debugging simulation data and reader plugins. There is no mechanism to produce ghost cells for data sets being operated on that do not already contain ghost cells. The work presented here could be leveraged by the VisIt tool to augment data sets in their pipeline with ghost cells.

Isenberg, et. al. [30] presents an algorithm for decomposed structured grids that works in parallel distributed memory by streaming blocks without ghost cells and preserving the known needed ghost cells while maintaining an ordered method of communication made possible by working with a structured decomposition of structured grids. The work presented here is different in that it works exclusively with "pure parallelism" of unstructured grid data, where an entire time step is loaded into memory simultaneously and is not necessarily structured.

Weber, et. al. [31] utilized ghost cells to generate stitch cells, which are then used to solve grid resolution differences on block structured AMR block boundaries for the purpose of topologically correct isocontour calculations. They provide a parallel algorithm to calculate ghost cells for grids derived from the block structured AMR. Our work is originally inspired by unstructured grid representations of tree based AMR data found

in the xRage simulation code, but is sufficiently generalized to support arbitrary well formed unstructured grids. The Distributed Data Filter, commonly called the D3 filter was developed at Sandia National Laboratories and was originally published as part of a parallel unstructured volume rendering scheme [22]. Since then it has been the state of the art for distributing and balancing unstructured grids across a number of processors and generating ghost cells to support stencil operations. It works by generating a parallel kd-tree to spatially balance the data set. The edges in the kd-tree are used to separate the cells by assigning cells to one partition, or the other, based on the cell centroid.

The details of the parallel decomposition with the kd-tree are very important to the next step of calculating ghost cells. If the cut line goes through a cell but the centroid location puts the cell in another domain, that cell would be a ghost cell for the local domain. This is the default in ParaView. Include all intersecting cells is turned off. A cell is only included if the centroid is in the local domains side of the cut.

Global point IDs and cell IDs are also very important. Without global points IDs, points have a much more expensive comparison to identify shared points. Points on different processors have to be shared and then compared, to find out if they are the same points. The work presented here uses data sets that do not have global cell or point IDs. Comparisons of points are performed on point coordinates to find duplicates.

Redistribution and ghost cell calculation are combined in the case of the D3 filter. This redistribution ends with every processor having information about the local extents of data contained on every other processor. The parallel kd-tree implementation of D3 produces a very nice way of each processor being able to sort cells and identify processor ownership without extra communication. This is not the case for some arbitrary partitioned, unstructured grid. If a grid could meet the D3 decomposition invariants imposed by the parallel kd-tree, the D3 filter could be used to generate ghost cells without redistribution. Our target data sets do not lend themselves to having a kd-tree imposed upon them without redistribution.

## 2.5 Ghost Cell Generator

In this section, a parallel multi-level ghost cell generator algorithm for distributed unstructured grids is presented. A naive implementation, capable of producing a single layer of ghost cells that surrounds the local partition is the beginning foundation. It is then shown how to use the information gained from that first step to iteratively generate an arbitrary number of ghost levels.

Optimizations to the base algorithm will be presented. Global sharing of local extents, which enables each processor the ability to reduce the number of potential neighbors to only those that are actually in the neighborhood is presented first. Other optimizations that build on the details of sending and receiving multiple ghost layers and reduce the stress on the core algorithm by keeping track of which cells the neighbors already have is then covered.

### 2.5.1 The Basic Algorithm

The topological building blocks for a 3 dimensional grid are points, edges, polygons and polyhedrons. Points are the fundamental, most basic, topological unit in the grid representation. Any edges, polygons or polyhedrons that share a point are neighbors. Points on the boundary of a local partition can be used to identify cells on other partitions

which also use that point. These cells are therefore neighbors for another partition. Our algorithm hinges on this observation that boundary points, when existing on two or more processors, are the defining element which indicates which cells need to be shared across partitions. Points are therefore used to identify shared domain boundaries. Any cells on these shared boundaries must be shared with any other partitions along the shared boundary. It is assumed that the input data set is partitioned, and each partition is owned by a processor. VTK is leveraged as a framework for the implementation presented here.

The ghost cell generator (GCG) algorithm is divided into seven major parts. These are summarized in Algorithm 1.

The first step is to for each processor to calculate the external surface of its local domain. In 3D, this is the surface of the volume, that is every polyhedral face that is not shared with another cell in the local domain. All of the points in the surface are then shared with every other processor. Each point in the local domain's hull is compared against each point from each of the other processor's external surfaces. If a point in the local surface matches with a point from another surface, all cells which use that point are sent to the processor which owns that surface.

**Algorithm 1** Calculate Ghost Cells

1: *Extract the Local Surface*
2: *Share External Surface With Potential Neighbors*
3: *Calculate Actual Neighbors*
4: *Create Cell Lists to Send to Each Neighbor*
5: *Send Cells to Neighbors*
6: *Receive Cells from Neighbors*
7: *Integrate Cells into Local Domain*

### Extract the Local Surface

The ghost cell generator starts with each processor finding all the points that exist on the external surface of the local partition. These are the points that are candidates for having shared cells on other partitions. No other points in the local domain can possibly share a cell remotely, since these points do not lie on the boundary of the local domain.

A practical algorithm to find these points is to find the surface of the local domain and make a list of the points that are used in the surface geometry. This involves a loop over each cell in the local domain. Therefore, its runtime is proportional to the number of cells in the local domain.

The vtkdata setSurfaceFilter is used to extract the surface of the local geometry and the surface points are harvested from the polygonal data output. It is worth noting since our algorithm supports a wide range of partitioned inputs, not every input is friendly to this algorithm. For instance, a partition could have every point in the local domain also be a point on the local external surface, maximizing the runtime for this step. If this partition has a relatively large amount of geometry, this could create a bottleneck.

### Share External Surface With Potential Neighbors

The ghost cell generator algorithm assumes each partition starts with no information regarding its neighbors. As a parallel processing algorithm using MPI the ghost cell

generator only has its rank, total ranks, and local partition information, but nothing regarding any other data, if any, on any other rank. In our naive implementation, all other ranks are considered potential neighbors and therefore the list of points on the local external surface are shared with all other ranks. This is an expensive all-to-all communication pattern where each rank must send to every other rank.

The worst case here, is that one of the processors has a very large count of points to send to its potential neighbors. In practice this can be exacerbated because that same rank might have taken an abnormally long time to calculate the surface as well, causing a potential cascade of poor performance as all other processors must wait for the slowest one to send its points before they start the next step. High performance computing networks are sufficiently fast today that the sharing itself is not a bottleneck.

### Calculate Actual Neighbors

Waiting for communication to finish can be a bottleneck, however, as all the neighbors must be calculated using all of the global information retrieved in the previous step. This step involves finding the intersection of a partition's local surface points with every other partition's surface points. At this point in the algorithm each processor has its own partition's surface points and the surface points of every other partition. The algorithm must simply compute the intersection of its surface points with each of the other partitions. A local kd-tree or a hashing mechanism can be used to iteratively compare each of the possible neighbors points with the local points. If a match is found, that point is a neighbor point, and the local cells that share that point are added to a list of cells to be sent to the processor that owns that partition.

Our implementation uses an acceleration structure to do this computation. A bottleneck can arise when neighbors have a large number of surface points. If $n$ is the total number of partitions composing the global data, and m is the size of the local surfaces, this should have a worse case runtime of $\mathcal{O}(mlogm)$, where the $logm$ represents the acceleration structure. Data sets with a large number of partitions, with large $m$, will get processed correctly but it will take longer.

### Create Cell Lists to Send to Each Neighbor

The cells need to be packaged for sending across the network. They must have a complete set of their own geometry defined. They must be completely self contained so they can be understood by the receiving rank.

The worst case here is that every neighbor would need all of the cells in the local data set. This will not generally be the case for a well partitioned data set.

The implementation presented here uses vtkExtractCells to obtain a subset of the local domain. That grid is marshaled and given to the communicator. Performance measurements presented here are coupled with the *Send Cells to Neighbor* operation since this is generally very fast in relation to the other times.

### Send Cells to Neighbor

Once neighbors are identified and packaged cell lists are created for each, the cells can be sent.

Sending cells to neighbors, in the worst case, has every processor sending data to every other processor. In a normal case where this algorithm is chosen over the D3 filter,

the data is already reasonably partitioned and the cells are not randomly distributed. An asynchronous send is used to transfer these data sets to their neighbors as they are produced. Timings presented here for this send are coupled with the packaging of the cells to send for two reasons. First they logically belong together as a prepare and send asynchronously operation. And second, the performance is relatively fast, so combining them produces a more useful plot.

#### Receive from Neighbor

All of the self contained data sets containing the ghost cells for a given partition must be received. Depending on the size of the send, and when the send was performed, this process exposes ranks that were later than others in getting cells sent or received. It is also dependent on the supercomputers network performance.

#### Integrate Ghost Cells into Local Domain

Once cells are received from the neighbors they need to be integrated into the local data set and tagged as ghost cells.

Each cell must be merged into the existing data set. This again requires an identification of duplicate points. All points in the incoming data set need to be reconciled with the existing points in the local domain. When a cell is merged into an existing data set, it must use the indices of the data sets points rather than its own. This part of the algorithm will, therefore, run in direct relation to the size of the incoming data set, which is somewhat bounded in practice by the number of the local domain's own surface cells. Poor relative performance can occur here when the data sets are large or structured in such a way that they interfere with the acceleration structures ability to quickly locate points.

Our implementation uses the vtkAppendFilter to accumulate all of the ghost cell filled data sets into one unstructured grid. The filter vtkMergeCells is then used to join the ghost cells data set with the local domain's geometry and data, merging any duplicate points that are present.

### 2.5.2 Multiple Ghost Cell Levels

After the first layer of ghost cells has been generated, each processor knows its neighbors. Also, every cell that was sent by a processor, was a boundary cell, and it also has a complete set of neighbors, ghost or locally owned. Therefore, every cell that was sent can simply calculate every one of its neighbors, which will contain the second neighbor (and more), of the partition it was sent to in the first round. Since the set of neighbors and cells are known, the next step is to follow the initial algorithm from step 4, *Create Cell Lists to Send to Each Neighbor*.

This process is repeated for each additional ghost level requested. Performance results of producing up to three layers of ghost cells are presented in the Results section.

### 2.5.3 Optimizations

#### Global Information Sharing

This optimization has each partition share its extents first in order to efficiently decrease the number of possible neighbors. As part of an initialization process similar to that found in [31], the algorithm starts with a global communication to share each processor's extents with all other processors. This provides enough global information for each processor to identify potential neighbors and share points only with those potential neigbors. This optimization has some cost up front in an all-to-all communication pattern, but it is a small quantity of data that must be shared: a six-tuple representing extents. The cost of the communication must be considered against the value of the information. Each processor ends the initialization with a list of every other processor's extents, which allows each processor to inspect every other processor's extents and find the neighbors. The complexity of this algorithm is $\mathcal{O}(n)$, where $n$ is the number of processors.

A fundamental problem with the naive implementation of our ghost cell generator is the lack of any global information at the beginning. Because any given processor only knows its fundamental information of being a process amongst $n$ processes, the set of potential neighbors that it must receive and process, the external surface points for intersection grows unwieldy with scale. An optimization to the algorithm that shares global information is introduced, in the form of extents, that allows each processor to identify all potential neighbors based on their extents. Although the potential worst case of an unstructured grid would have a complete overlap of extents, the ghost cell generator algorithm presented here was designed for a well formed structured grid that has bundles of cells partitioned spatially.

Sharing local extents globally allows the algorithm to substantially narrow the set of possible neighbors, reducing the network traffic and the number of times the point intersection algorithm must be run. The cost of communicating an all to all is expensive, but the pervasive time savings from using the information is substantial as can be seen in section 6.3.2.

#### Send Only Once

Successive layers of ghost cells leverage the observation that any processor that sends any number of cells to another processor for the first layer of ghost cells, is guaranteed to have a second layer of ghost cells for a processor if they exist. Any cell that touches a point sent in the first round is a potential second layer ghost cell for the destination processor. This optimization to the ghost cell generator algorithm improves on the naive implementation by remembering the set of cells already sent to a partition and not sending them again. It saves network traffic and remote integration time of cells that will be identified as duplicates anyway. The cost is in local memory and a small amount of compute.

#### No Send Backs

A processor/partition that has sent a ghost cell, should not have that cell sent back as a potential ghost cell when the next layer of potential neighbor ghost cells is calculated. The Ghost Cell Generator filter uses neighbor information from the initial first layer calculation to calculate succesive ghost cell layers. A naive algorithm simply finds all points from the cells sent in the last round, and sends all cells connected to those points in the next round as described in Section 2.5.2. This is a problem, because cells received

in the last round all share points with cells sent in the last round. This means that a processor will naively send cells back to its trade partner. Even though that trade partner obviously already has those cells. So, this opimization keeps a list of cells received from a processor and removes any cells slated to be sent back to that processor before sending them back in the next round of ghost levels.

## 2.6 Experimental Setup

### 2.6.1 Implementation of Ghost Cell Generator

An implementation of the Ghost Cell Generator (GCG) as a parallel filter in VTK was developed. The algorithm and an implementation is described in Section 2.5. The filter is exposed in ParaView.

### 2.6.2 Snow Supercomputer

For our timing results, a Los Alamos National Laboratory supercomputer called Snow was used. It is, an ASC Program Commodity Technology System Phase I cluster (CTS-1), running the Tri-Lab Operating System Stack (TOSS). Snow has a total of 368 compute nodes with 36 processing cores in each for a total of 13,248 processing cores. Each core has 3.5 GB of memory. Snow is configured with the Intel OmniPath interconnect, and has a peak operating speed of 445 Teraflops.

### 2.6.3 Data sets

Time steps from the Deep Water Impact Data Set [24] are used. It is a simulation of an asteroid falling through the atmosphere and impacting the ocean. In particular, the xA50 ensemble member cycle number 41760, is heavily leveraged. It contains nearly 600 million total cells. The time step contains 1024 VTK Unstructured Grid Files. Each file represents the domain from each of the simulation processes. There are 11 scalar fields, each of which are floating point values.

A generated well structured, unstructured grid data set was also created using the ParaView wavelet source. It was an 800 cubed data set containing one floating point scalar field, totaling 500,000 cells. A threshold filter was run that returned the entire data set as an unstructured grid.

## 2.7 Results

Weak scaling performance of the ghost cell generator algorithm for both our naive and fully optimized versions in relation to the performance of the D3 filter, all producing one ghost layer, can be seen in Figure 2.10. The naive ghost cell generator makes no special attempt at optimization for any of the stages. The sharp rise in time, between 512 cores and 1024 cores, for the naive ghost cell generator can be attributed to the increased communication and compution that grows in relation to the number of processors. Both the naive ghost cell generator and the D3 filter are in stark contrast to the performance of the optimized ghost cell generator. The only optimization exposed here is the global sharing of extents, which reduces the number of potential neighbor partitions from every

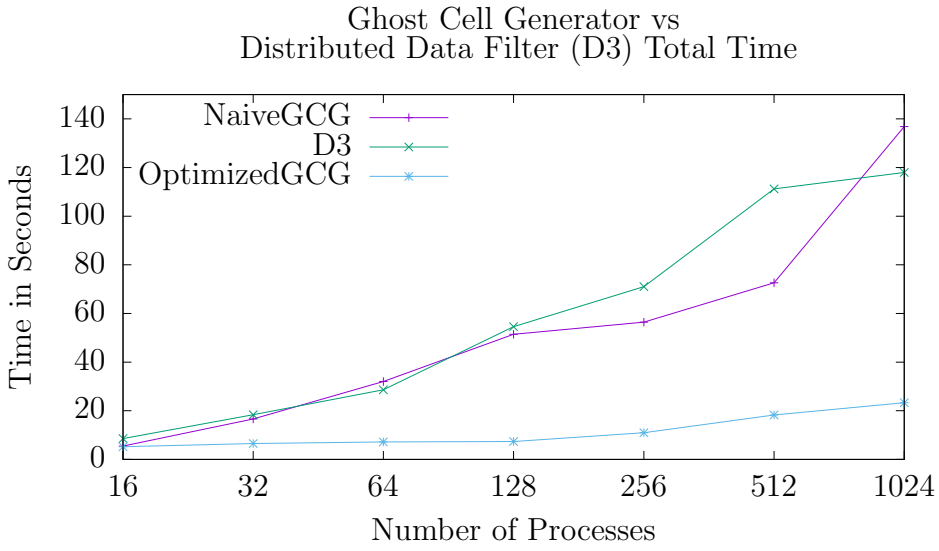



Figure 2.10: Weak scaling performance of the naive implementation of the ghost cell generator filter versus the data decomposition filter. Note the degrading performance with scale.

partition, to just those that are actually nearby, based on their respective local extents. The time saved by sharing extents is substantial.

A breakdown of the performance of the optimized ghost cell generator from Figure 2.10's 32 process, "OptimizedGCG" number can be seen in the stacked bar chart in Figure 2.11. The extract surface, in this run, is very consistent and is responsible for a small fraction of the overall time. The "ShareSurface" value includes the cost of sharing the extents. Calculate neighbors has a wide range of times over the set of processors. Those with processors with many neighbors who send many cells are slow. You can see the receive times for many are very long. Many of the longer receives appear to be waiting for the input from the slower processors such as process 20, which seems to have stalled a large number of neighbors, while calculating its own neighbors. Once process 20 completes its calculation and sends, its neighbors quickly start integrating their ghost cells into their local domain. The performance breakdowns seem to follow this pattern. There is always a worse group of neighbors responsible for the longest run time. In fact, there is usually a single process that can be identified, like process 20 here, for causing all processes to wait.

Figure 2.12 shows strong scaling performance using a 500 million cell data set using 256 to 4096 partitions. The data set has 4096 partitions, neighbors were merged to produce the smaller partition counts. The scaling looks very good between 256 and 1024 processes. The performance flattens between 1024 and 4096. The D3 filter also shows very good strong scaling as well, however, it failed to run on 4096 cores. This plot shows the functionality of the ghost cell generator across a wide spectrum of local partition sizes.

The ghost cell generator algorithm supports the calculation of multiple ghost cell levels. A summary of a two ghost level calculation on our weak scaling data set can be found in Figure 2.13. Both the *send only once* and the *no send backs* optimizations save a modest amount of time, if any. For larger processor counts, the expected performance appears to settle down with both optimizations being the fastest and no optimizations

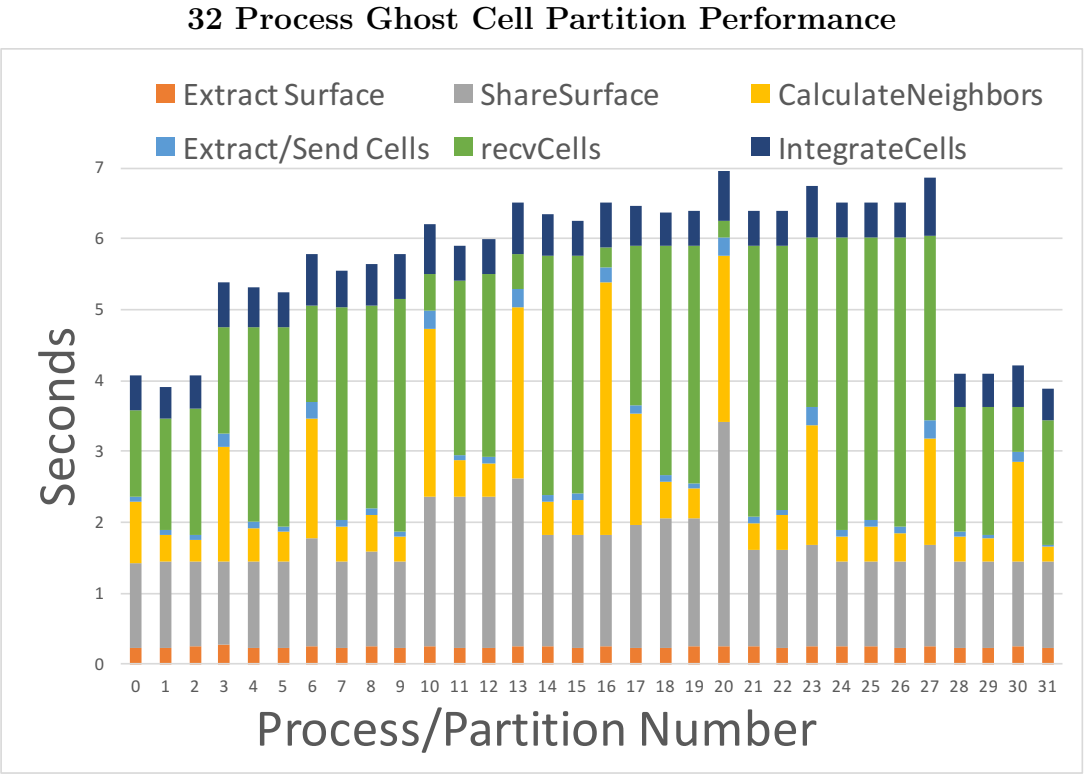



Figure 2.11: Break down of the ghost cell generator performance.

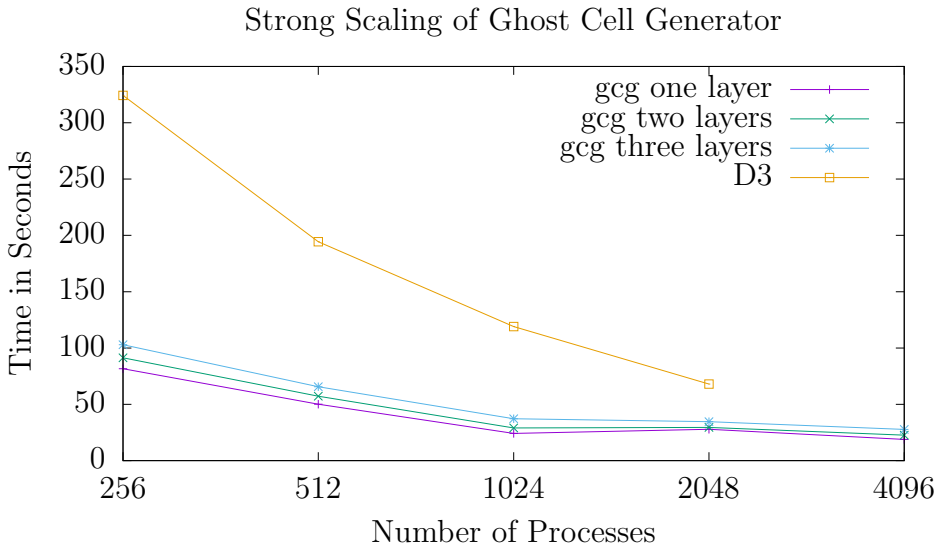



Figure 2.12: Strong scaling performance of a 500 million cell data set distributed across up to 4096 partitions. Although D3 shows excellent strong scaling, it failed to run at 4096 processes.

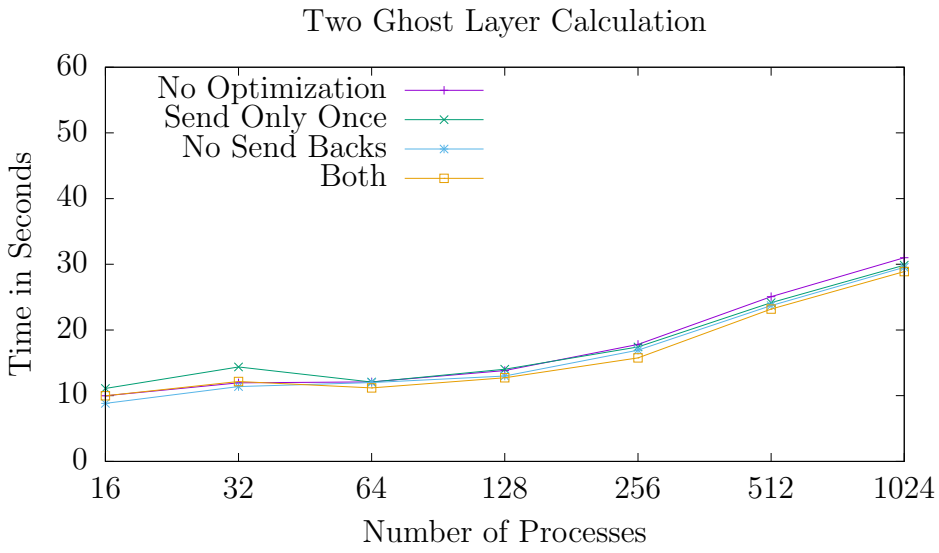



Figure 2.13: Performance of ghost cell generator using shared extents to compute 2 layers of ghost cells. For this case the multilayer optimizations do not contribute significantly.

being the slowest, together they save the most time. This is also true for the 3 Layer case found in Figure 2.14, where a 20 percent time savings is recognized by the optimizations on the largest core count.

Finally, the total number of cells in a data set increase as the number of ghost levels increase. Figure2.15 shows how the number of cells increases regularly with increasing cell counts for each of our weak scaling values. The 1024 processor case, the total number of cells increases by 218 million. The number of ghost cells produced by any given data set will be data dependent and can vary quite widely.

## 2.8 Conclusion

A parallel multi-level ghost cell generator algorithm and performance results on two different data sets was presented. The need for ghost cells was described, as well as a specific use case which required multiple layers of ghost cells. The multi-level ghost cell generator is available in ParaView version 5.2.0 and beyond.

### 2.8.1 Using the solution

The ghost cell generator filter is in ParaView 5.1.2 and beyond and can be invoked in the normal way in which ParaView deals with filters, it is available when exporting catalyst scripts and in ParaView's Python wrappings. It is also in VTK 7.0.0 and beyond and is available in both the C++ and Python interfaces. The implementation also allows for a second or more layer of ghost cells to be calculated. For layers of ghost cells greater than 1, the algorithm simply remembers which processors it sent its cells to and which cells, the algorithm then simply sends to those processors again all of the cells associated with the points it already sent. 2 layers of ghost cells are important when dealing with cell data that needs to be transformed to point data for certain algorithms which then produce geometry for rendering. The geometry for rendering calculates normals at each vertex by averaging all the triangles touching that vertex. The problem is the triangles calculated

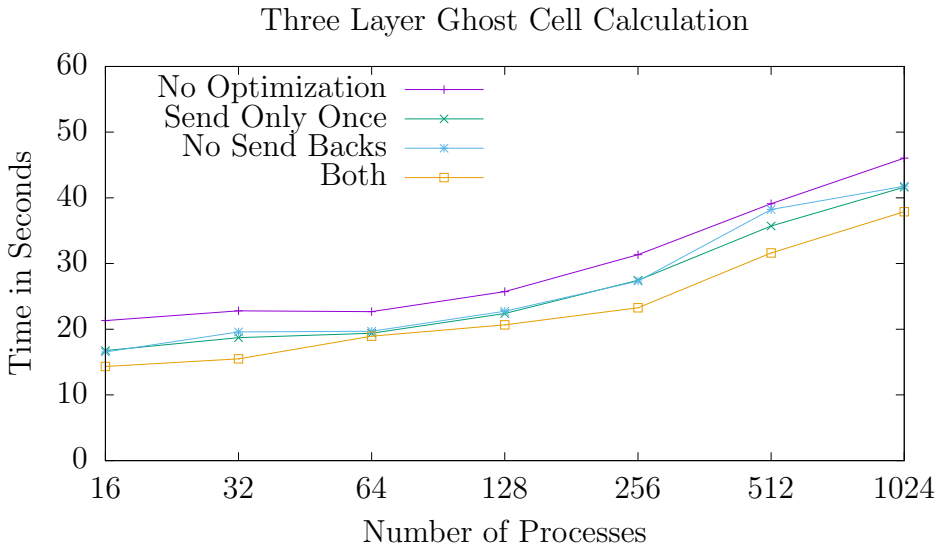



Figure 2.14: Performance of ghost cell generator using shared extents to compute 3 layers of ghost cells. For this case, the multilayer optimizations provide a 20 percent improvement.

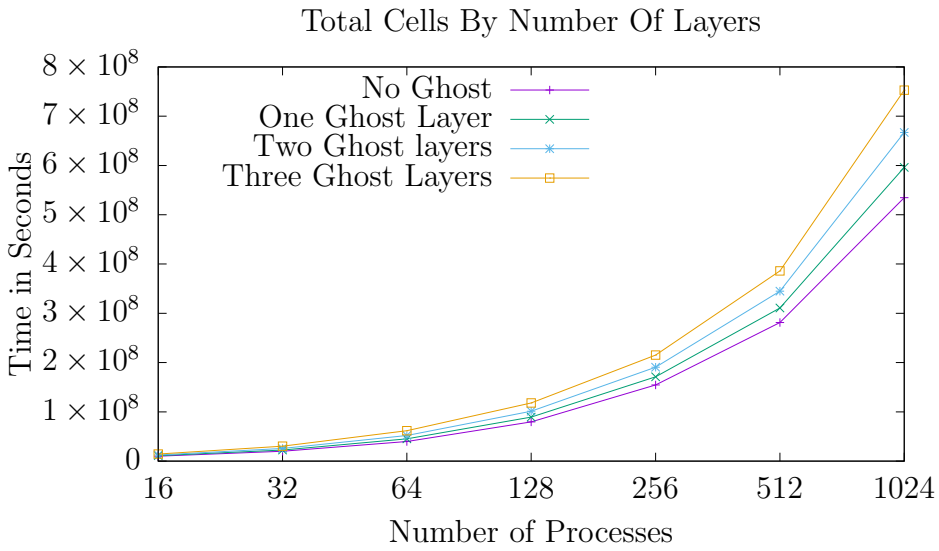



Figure 2.15: Total Cells in the data set increase subststantially as more ghost levels are added

in the first layer of ghost cells are incorrect after the cell to point operation made the external hull points of the ghost cells incorrect. Therefore the normals on the local domain boundary vertices will be incorrect. The visualization pipeline artifacts, created by the lighting model at points of incorrect normals on boundaries, are most evident in the high gradient areas and can be seen in Figure 2.2b. A recent movie submitted to the Supercomputing 2016 Visualization showcase [15] could have leveraged ghost cells and saved a vast amount of time in post processing and merging of data to be processed in a single memory space.

## 2.9 Core References

John Patchett, Boonthanome Nouanesengsy, Joachim Pouderoux, James Ahrens, and Hans Hagen. Parallel multi-level ghost cell generation for distributed unstructured grids. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on*. IEEE, 2017

# Chapter 3

# In Situ and Post Processing Workflows Case Study: Asteroids

**Abstract**

Simulation scientists need to make decisions about what and how much output to produce. They must balance their ability to efficiently ingest the analysis with their ability to get more analysis. This balance as a trade-off between flexibility of saved data products and accessibility of saved data products is studied in this chapter. One end of the spectrum is raw data that comes directly from the simulation, making it highly flexible, but inaccessible due to its size and format. The other end of the spectrum is highly processed and comparatively small data, often in the form of imagery or single scalar values. This data is typically highly accessible, needing no special equipment or software, but lacks flexibility for deeper analysis than what is presented. A user driven model and analysis that considers the scientists' output needs in regards to flexibility and accessibility is presented. This model allows for the analysis of a real-world example of a large simulation, lasting months of wall clock time, on thousands of processing cores. Though, the ensemble of simulation's original intent was to study asteroid generated tsunamis, the simulations are now being used beyond that scope to study the asteroid ablation as it moves through the atmosphere. With increasingly large supercomputers, designing work flows that

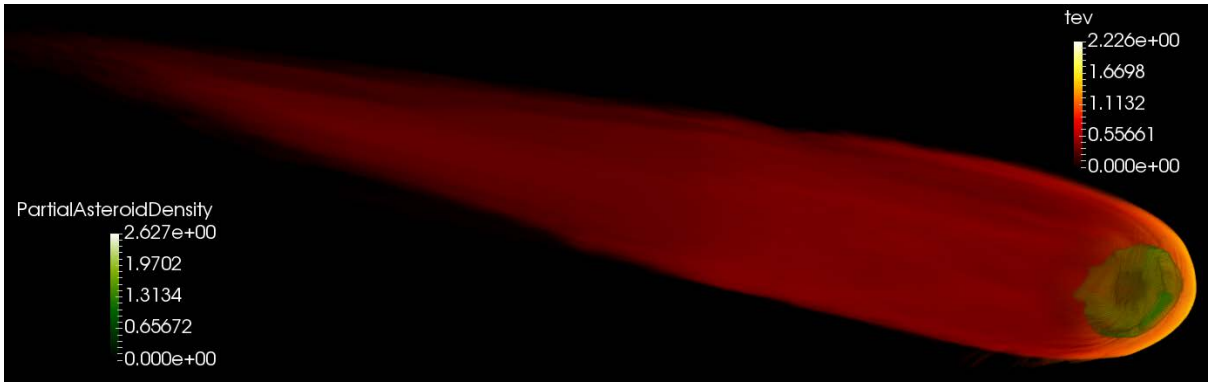



Figure 3.1: Volume renderings of two thresholds of partial asteroid density. One colored yellow to red by temperature and showing the ablation material the other colored by partial density in green shows only the cohesive asteroid flattened from the entry.

support an intentional and understood balance of flexibility and accessibility is necessary. In this chapter, a new strategy is presented, developed from a user driven perspective to support the collaborative capability between simulation developers, designers, users and analysts to effectively support science by wisely using both computer and human time.

## 3.1 Related Work and State of the Art

Asteroids are potentially deadly objects racing through our solar system with one, a meter in diameter, entering the earth's atmosphere every other week [32]. Small asteroids are common, while larger are more rare. Asteroid $TC_3$ [33] entered the atmosphere over the Northern Sudan in 2008, it marked the first time scientists were able to detect an asteroid prior to it entering the earths atmosphere. Scientists did not see the 2013 Chelyabinsk meteor [34] which disturbed a substantial population area, damaging buildings, mostly broken windows in the middle of the Russian winter and sending many to the hospital for related injuries. If scientists are fortunate enough to detect an asteroid prior to its entering the earths atmosphere, scientists need to have studied the problem and provide decision makers with potential solutions.

Variables to consider when studying an asteroid impact include size, composition, speed, angle of entry and whether or not there is an air burst. Our work supports scientists studying asteroid impacts using ensembles of simulation results generated with in situ capabilities to make decisions about how, how much and which data to save. The simple in situ versus post processing argument is surpassed and a study of an actual work flow that leads to a user driven model that can support decision making with domain scientists to meet user needs over time is contributed. Scientist analysis needs change over time and the requirement of systems to support the balance of flexibility and accessibility of results needs to be maintained.

## 3.2 Background

The data product summary in Figure 3.2 shows typically expected output sizes and their effects from the three main operations of the supercomputing work flow that are studied here: simulation, reduction, and visualization and data analysis (VDA). Simulation output is the largest and least refined, VDA output is the smallest and most refined and reduction output is somewhere in between, on the spectrum generated by these two extremes. Reduction operation examples include compression or feature extractions, they typically are lossy operations, sacrificing precision or exclusion of certain subsets.

Flexibility refers to the total information content in the data product. For instance a simulation dump contains the most general information while a single image contains very specific information. Accessibility refers to the ease of access of the data product. A very general data product coming directly from a simulation usually requires sophisticated tools to access usable information, while a highly refined data product from a VDA operation produces data that can be viewed in a shell or web browser. It is easily and quickly ingested by the domain scientist.

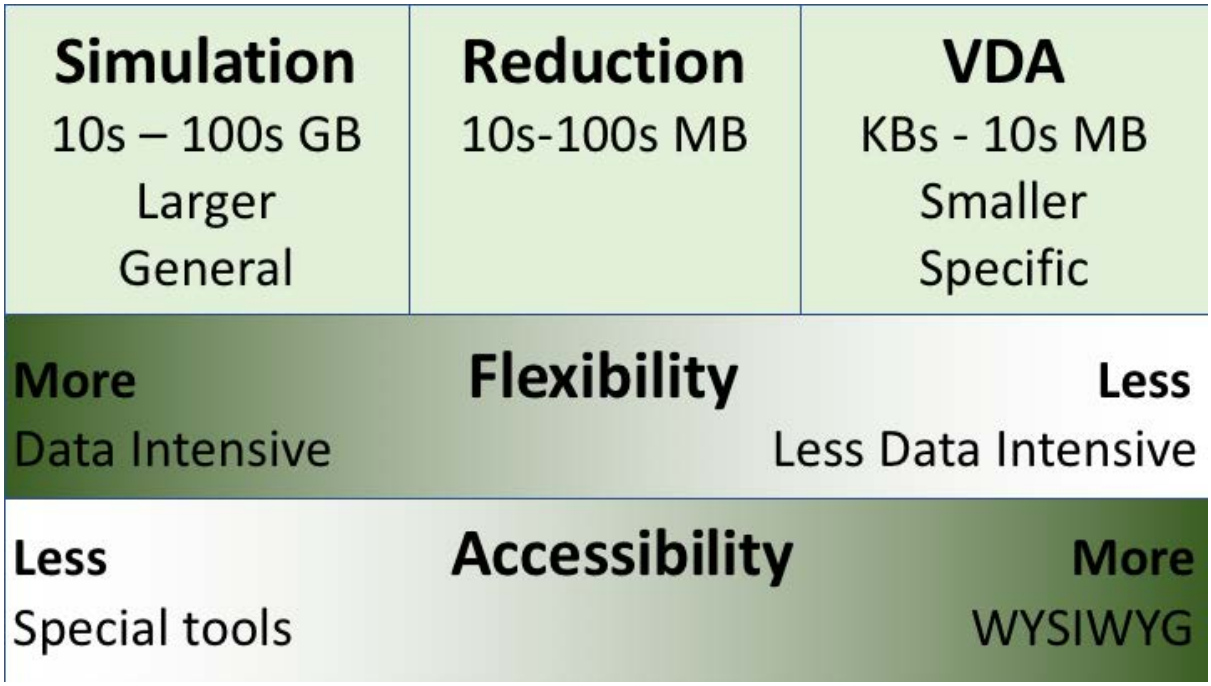



Figure 3.2: A summary of the supercomputing work flow operations. Reduction refers to data reducing operations. VDA stands for visualization and data analysis.

## 3.3 Related Work

Work presented here is empirically derived directly from work on in situ for supercomputing applications [35,36], particularly while preparing a domain scientist for the the Second International Workshop on Asteroid Threat Assessment: Asteroid-generated Tsunami (AGT) and Associated Risk Assessment [37] which supported decision making by the attendees and was also presented at Supercomputing 2016 [38]. This work seeks to improve on the analysis capabilities afforded by saving defensive checkpoint restarts at some regular interval [39]. This work also part of a search for ways to move toward a smaller footprint on file systems and is inspired by research like the Cinema project [40], in situ feature detection and preservation [6, 41, 42], and compression [43, 44] all of which enables users to make trade-offs between accessibility and flexibility and still manage practical constraints like storage, space and time. Our models and strategies also attempt to consider, or at least not exclude, emerging hardware and software technologies like burst buffers [12, 45] and VTKm [46]. Many observations have been influenced by the data, information, knowledge and wisdom hierarchy described in [47].

## 3.4 Approach

The specifics are limited to a single simulation of a 500-meter diameter asteroid, initially at a 20-kilometer elevation ripping through the atmosphere at 17 kilometers per second on a simulation grid containing three materials: air, water and asteroid as seen in Figure 3.3. The simulation begins with a modest 150 million cells, but quickly ramps up to 500 million cells when the asteroid impacts the water and then quickly goes to 1.3 billion cells. Our initial goal is to analyze the effect on the deep ocean water after the asteroid impact. A later goal, generated after the initial simulation run, is to then study the asteroid itself,

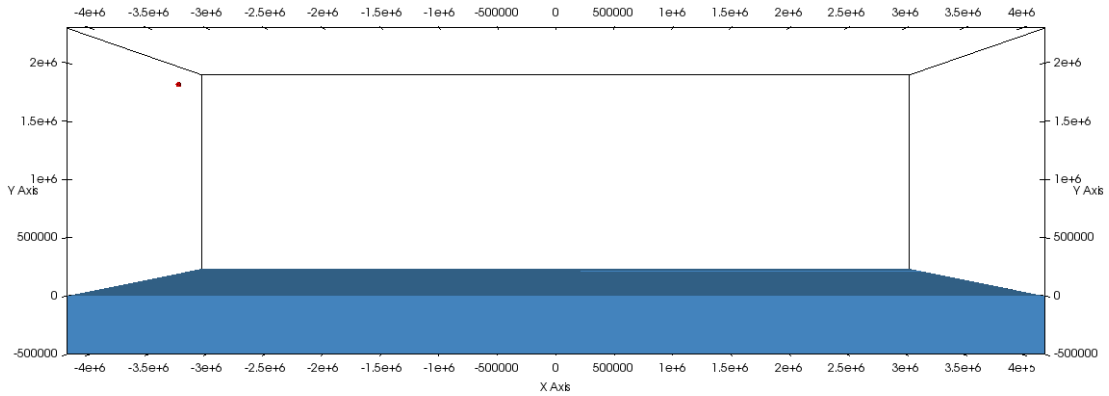



Figure 3.3: A visualization of the initial conditions. The asteroid is red, water is blue and air is everything else in the bounding box. The axis values are in centimeters.


Figure 3.4: A simple work flow that begins with a simulation, extracts a feature and performs visualization and data analysis on that feature with the decision to store or not store data between each processing element.

prior to impact.

The computational simulation is performed by xRage [23], a parallel multi-physics Eulerian hydrodynamics code that is developed and maintained by the ASC program at the Los Alamos National Laboratory. xRage uses a continuous adaptive mesh refinement (AMR) technique that allows smaller computational cells in areas of interest and larger, thus fewer, cells in other areas, which enables more efficient use of the supercomputer. The simulation is outfitted with an integrated ParaView Catalyst capability that translates the computational grid into a VTK unstructured grid representation, then hands control to ParaView which is capable of executing a Reduction or VDA pipeline before returning control back to the simulation code.

A supercomputer that contains 2 GB of memory per processing core with a high performance interconnect between nodes (16-36 cores per node) was used. Access to a VDA cluster which has many fewer nodes but contains 196 GB of memory and 12 cores per node was also secured. Both of these machines have access to large multi petabyte shared file systems. Many desktop computers with modern graphics hardware, 64 GB of RAM and a single terabyte SSD storage was part of the inventory.

Figure 3.4 shows the basic work flow that is being studied here. The simulation produces data that can either be persistently stored or can be handed directly to a feature extracting algorithm. The algorithm will produce a reduced data set which can be either persistently stored or passed directly into the visualization and data analysis algorithm. This algorithm, in turn, will produce easily accessible data products like imagery or numbers. There are four paths through this work flow. The full in situ stores no intermediate data and produces only final visualization and data analysis products. The full post processing stores intermediate data products at each juncture. There are two hybrid approaches. The first passes the data directly to the reduction algorithm in

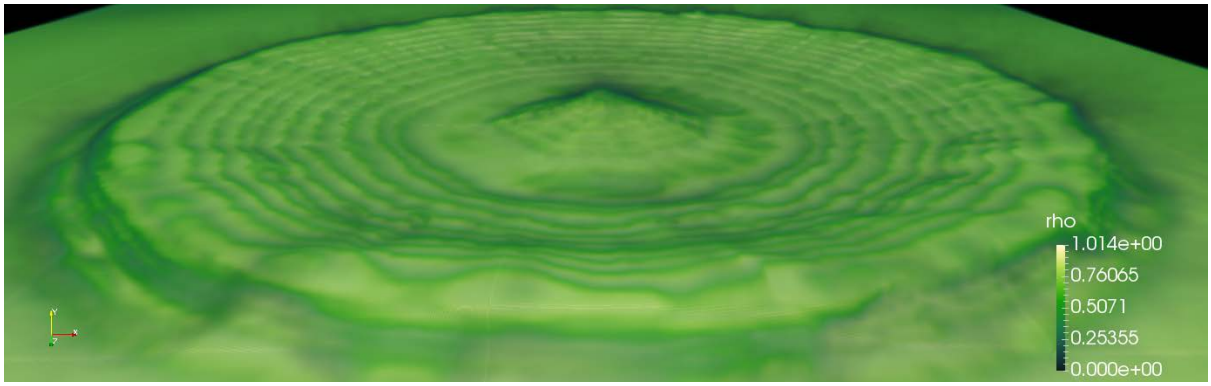



Figure 3.5: Volume rendering that shows cean surface extracted.

situ and stores the much smaller output from that which can be quickly and efficiently read by the visualization and data analysis operation. The second starts as a traditional post processing algorithm with a large data set dropped from the simulation and the feature extraction and visualization and data analysis are combined with no intermediate writes.

$$T_{total} = S + S_{out} + R_{in} + R + R_{out} + V_{in} + V \tag{3.1}$$

Formula 3.1 shows the cost model derived from the possible work flows. Let $T_{total}$ be the sum of all costs, $S$ be the cost of the simulation operation, $R$ is the data reduction operation and $V$ be the cost of the visualization and data analysis. $S_{out}$ is the cost the simulation's data output which could be a disk write or an in situ adapter cost. $R_{in}$ is the cost for the reduction operation to access the data, this could be near zero in the case of in situ or the time to read from disk for the post processing. $R_{out}$ is the cost of the reduction operation producing data and either storing it or passing it to the VDA operation. Finally $V_{in}$ is the cost of ingesting data for the visualization and data analysis operation.

## 3.5 Results

Four fundamental use cases based on the model are identified here. They are presented in the natural order that they appear to occur that aligns with the maturing of a simulation coupled with a topic and a group of domain scientists. One goal would be to achieve the fourth use case as quickly as possible.

$S$, $R$, and $V$ are static costs, regardless of work flow. The computation expense doesn't change and can simply be removed from the analysis, leaving only the details of the data movement decisions. $S_{outInsitu}$ and $S_{outFile}$ are the two main classes of expense coming out of the simulation. $R_{inInsitu}$ and $R_{inFile}$ represent the cost of getting data into the reduction operation, and $R_{outInsitu}$ and $R_{outFile}$ are the costs of getting data out of the reduction operation. Finally $V_{inInsitu}$ and $V_{inFile}$ are the costs for getting data into the VDA operation.

### 3.5.1 Full Post Processing: $S_{outFile} + R_{inFile} + R_{outFile} + V_{inFile}$

The first data product that is typically and persistently stored from simulations is the checkpoint restart (CR). This is the entire state of the simulation required to restart

from that state. Analysis can be performed with the CR as a source. Then some other type of output that is more accessible usually comes next: the viz dump. The viz dump is classified in this bin as it usually contains the full grid and a selection of scalar values. This work flow maximizes flexibility at the cost of disk space and user/computer time. Interactive volume rendering was desired. This is a difficult task using production software on unstructured grids with hundreds of millions of cells. The solution was to read the visualization dumps, sample them onto structured grids and save those, which effectively turned 30-100 GB per time step data sets into .6-1.2 GB per time step data sets. This data reduction not only made interactive exploration feasible, it made movement of the data to local disks that didn't require supercomputers feasible. This is a vast improvement in accessibility. An output from such data can be seen in Figure 3.5 which shows the ocean surface after the impact, which is important when analyzing tsunami generation.

### 3.5.2 Hybrid: $S_{outFile} + R_{inFile} + R_{outInsitu} + V_{inInsitu}$

Improving the reduction work flow to increase the accessibility and specificity of the output, starting from the vis dump all the way to distinct data products, is a natural progression in scientific inquiry. Output files which were produced to answer more general questions generated more specific questions. These specific questions required more specialized data in order to obtain an answer.

In our case, volume rendering the entire mesh generated questions of how much of the asteroid was actually being lost while it went through the atmosphere. The sampled data clearly showed that a large quantity of asteroid material was left behind and disintegrated in the atmosphere before the impact. The sampled data, although great for making macro level, qualitative imagery accessible, was not fully appropriate for measuring the size of the solid asteroid. The native grid needed to be accessed again. This required loading the time step data, which were in the tens of gigabytes. Once the data was loaded on the supercomputer using hundreds of processors, the asteroid could be extracted, requiring a fraction of the original grid, on the order of tens of megabytes, which is easily accessible for interactive volume rendering. The exact values required to extract the solid asteroid were not confidently known. Density values that were well into asteroid dust range were included. Since basalt density is on the order of 2.7 $g/cm^3$ a threshold on computational grid cells of .5 $g/cm^3$ was run to produce smaller data sets. These data sets are sufficiently small to transfer with high accessibility. Results of this reduction in size can be seen in Figure 3.6. Full resolution for the volume of interest is preserved in the reduction. The image in Figure 3.1 used such a refined work flow and the relatively small asteroid seen in Figure 3.3 helps to explain such a reduction.

### 3.5.3 Hybrid: $S_{outInsitu} + R_{inInsitu} + R_{outFile} + V_{inFile}$

Data representing the asteroid between the originally saved time steps, see Figure 3.7, is needed to better visually explain the mushrooming between 1.0 and 1.6 seconds. Value in higher temporal, full resolution dumps from the simulation are not expected to be of great value. The scientist is now searching for very specific features in his data, but he still wants some generality for exploration within the subset of the new study. Improving the in situ dump to provide only known needed data is the first jump back to the domain scientist who is normally responsible for running the simulation.

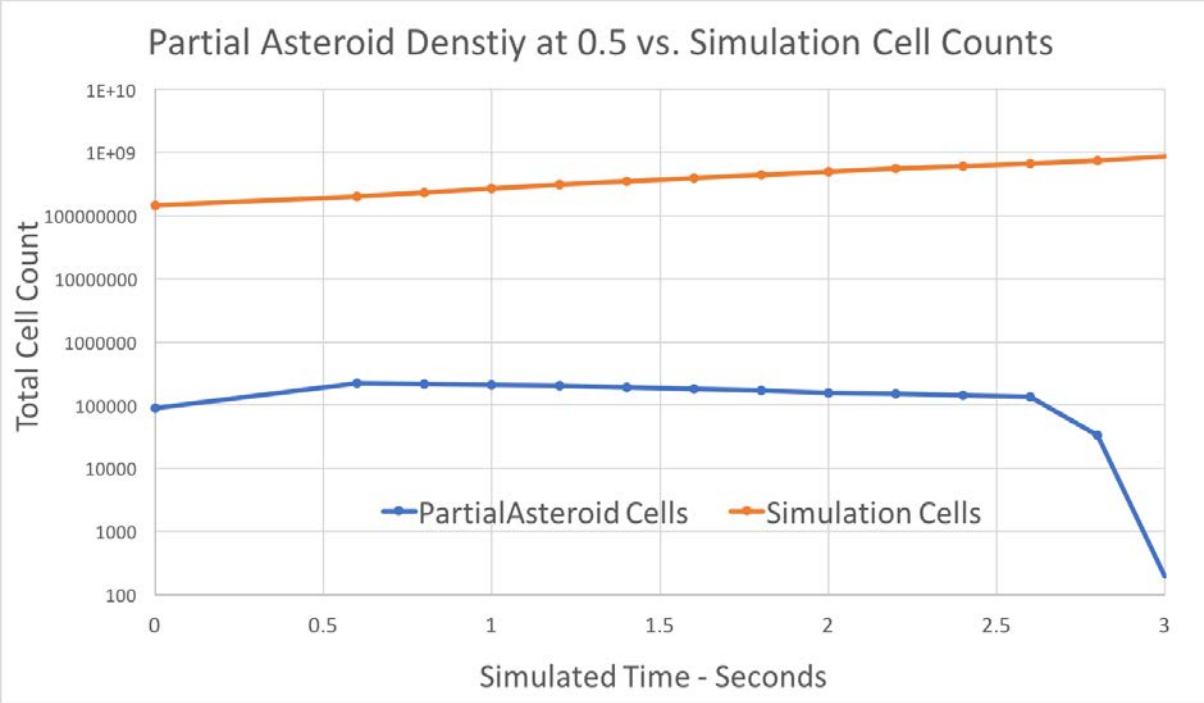



Figure 3.6: A plot showing total cells in the full simulation grid versus the total cells in an asteroid partial density threshold of $> 0.5$

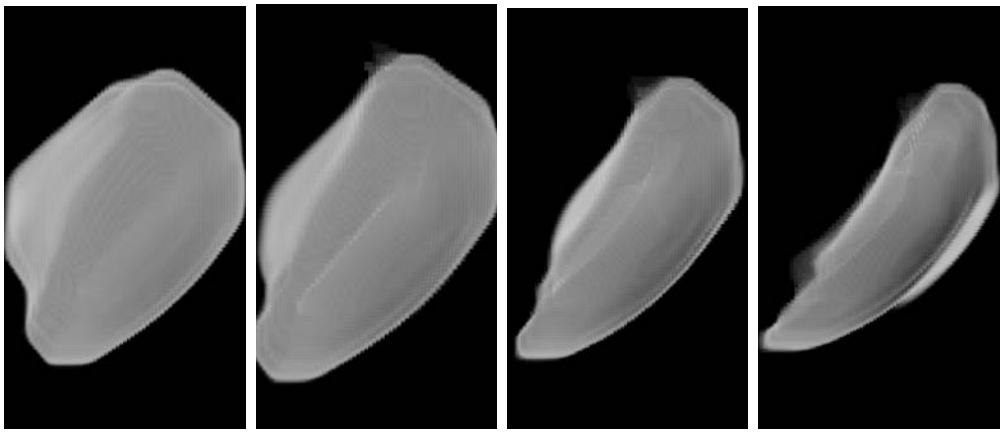
Having developed a pipeline of asteroid extraction, given the integration of the VDA tool with the simulation, it was trivial to simply rerun the simulation and have the simulation output high resolution asteroid. It is still expensive in terms of time to run the simulation again. Spending time doing defensive checkpoints or vis dumps is not necessary. The effects on the file system are minimal and hundreds of easily accessible time steps can be saved in the disk space consumed by a single vis dump.

This is the first step in really removing flexibility in support of accessibility. If the generality of the dump is needed again it will have to come from the running simulation which would have a greater cost in terms of time, typically, more than simply reading from disk. The improvement in accessibility, though, is large, as the data is much more refined, more information dense in an area of interest, potentially much smaller, and therefore potentially accessed with fewer resources and fewer specialized tools and definitely more quickly. Accessibility to an end-user should not be underestimated. Of course they want both accessibility and flexibility.

The decision by a domain scientist to sacrifice the generality found in saving full spatial resolution dumps comes only with an increased maturity of understanding what the implications are of sacrificing that loss for the accessibility. They know what they're looking for and already have a good sense of what they'll be missing.

### 3.5.4 Full in situ: $S_{outInsitu} + R_{inInsitu} + R_{outInsitu} + V_{inInsitu}$

The full in situ comes at the end of the pipeline. The simulation is well understood. It is unlikely that any other information will be needed in the near future regarding the simulation. Some number of checkpoint restarts and all of the data necessary to rerun the simulation must still be preserved. This is a worthy goal of achievement. The risk associated with the full in situ is potentially mitigated by projects like Cinema which can



(a) 1.0 seconds (b) 1.2 seconds (c) 1.4 seconds (d) 1.6 seconds

Figure 3.7: A sequence of asteroid at the temporal resolution saved from the simulation partial density threshold of $> 0.5$

potentially save a large quantity of refined information, which is stored in a searchable database.

## 3.6 Conclusion

A model was presented with an example, including the context of a real-world use. The ideas can be leveraged by individuals and groups in designing simulation runs to ensure accessibility and flexibility needs are met. This is important to manage, not only for compute and storage resources, but importantly the scientist time. This leads us to a full delivery of in situ capability to end users.

## 3.7 Delivery of In Situ Capability to End Users

### 3.7.1 Introduction

ParaView Catalyst [48] was chosen to deliver in situ visualization and analysis capability to end users. Although the capability had been created, only the most dedicated users could find a path to making it work, this was the status quo beginning this project. Ease of use was lacking for the end users of the xRage [23] code. xRage is a parallel multi-physics Eulerian hydrodynamics code that is developed and maintained by the Advanced Simulation and Computing (ASC) program at the Los Alamos National Laboratory (LANL). xRage uses a continuous adaptive mesh refinement technique that allows smaller computational cells in areas of interest and larger, thus fewer, cells in other areas, which enables more efficient use of the supercomputer. The LANL ASC program created a milestone to ensure the technical capability of in situ was delivered to end users. This paper documents the work and observations that followed the simple milestone of "deliver in situ to end users" that occurred over a one year period. In particular, the first successful delivery using the Cinema [40] project was achieved. This generated new user requirements, improved software engineering processes and a more rigorous release schedule of ParaView.

### 3.7.2 The In Situ Work Flow

Simulations of arbitrary physical phenomena don't have a standard template for necessary output. The general solution is to save a high spatial resolution "dump" of the simulation's state at a low temporal resolution. In situ libraries enable alternatives. ParaView Catalyst is an open source, in situ, data visualization software maintained by Kitware, Inc. It is based on ParaView [27] and is designed for distributed memory parallel environments. Catalyst exposes ParaView as a library. Simulations call the Catalyst library at runtime and predefined visualization and data analysis products are produced. Several things need to occur to enable a ParaView Catalyst work flow.

1. In situ adapter needs to be created for the simulation.
2. Simulation needs to be built with the adapter linked to ParaView Catalyst.
3. Visualization definition file needs to be created.
4. Simulation runs with Catalyst pointing at visualization definition file.

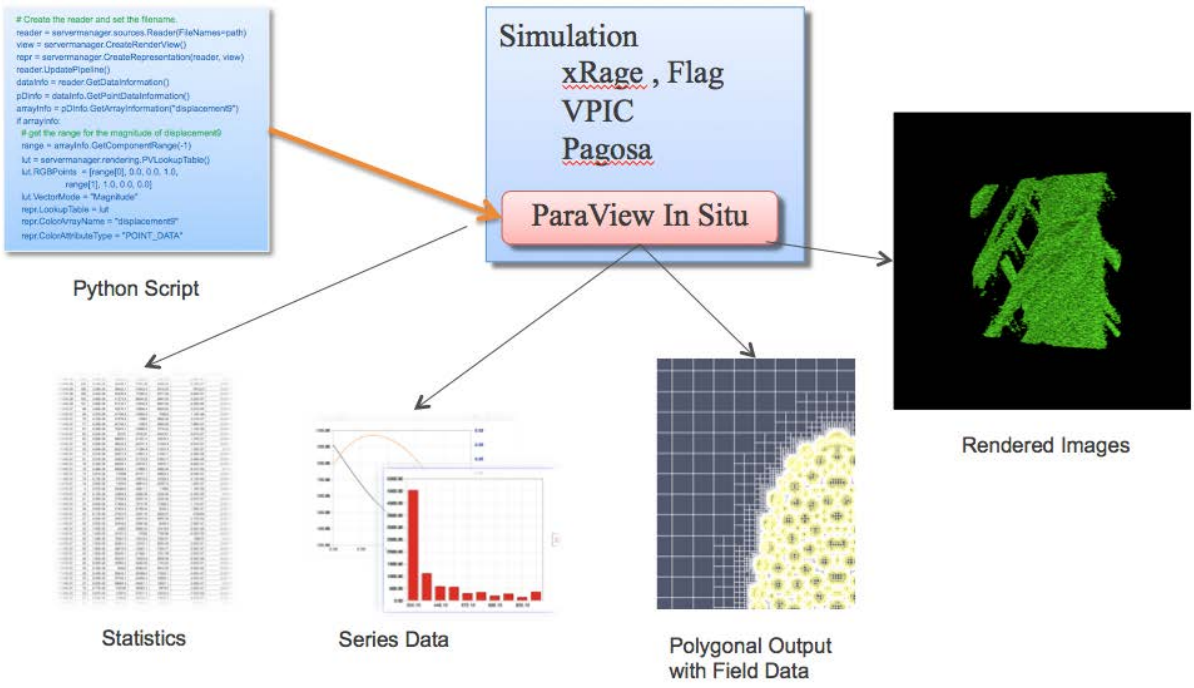



Figure 3.8: This diagram shows the general design of a functioning catalyst system. The simulation runs, reads in a python based visualization pipeline description and outputs visualization products.

ParaView Catalyst has now been linked to many codes at Los Alamos [11]. The usual mechanism to build a simulation code against the ParaView Catalyst library is to first build a ParaView with the same compiler and MPI that will be used by the simulation for each necessary supercomputer. Once a compatible ParaView is built, an in situ adapter can be built and linked. The in situ adapter is custom for each code. It translates a simulation's state to a VTK data set that can be operated on by Catalyst. Prior to our work, in situ adapters were being stored in separate repositories from the simulation

code and sometimes even in ParaView itself. In situ adapter code now resides with the simulation code to be co-developed. This was a requirement for the milestone. This was difficult as code teams protect from scope and code creep. Prior to our work, code teams were unaware of the capabilities of an integrated in situ library. Code teams generally buy into the idea with time and user pressure. The powerful possibility of debugging the simulation code by having high resolution access to the grid is a winning argument.

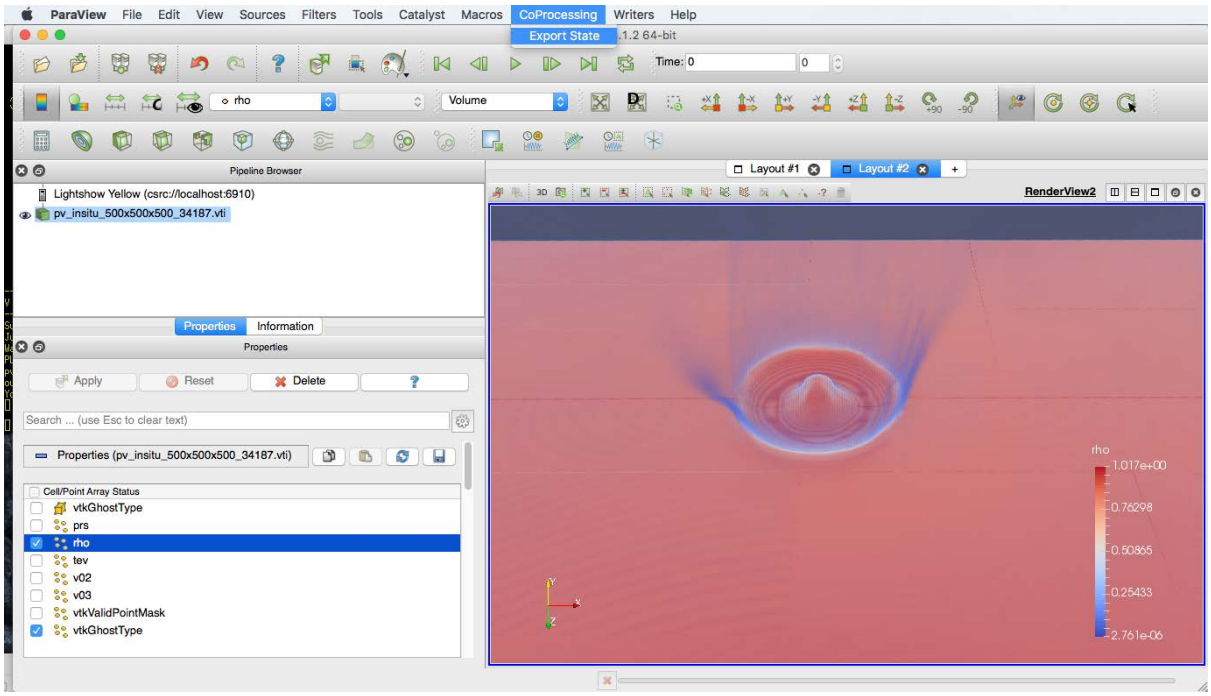



Figure 3.9: The ParaView Graphical User Interface about to export a ParaView Catalyst pipeline definition script.

Each simulation stores data differently. A Catalyst in situ adapter transforms simulation grids and data into a data set structure understood by ParaView. Every simulation will need to produce its own in situ adapter so that mappings of in memory simulation data structures are properly mapped to the VTK based data structures.

The simulation must link to a ParaView built with the same Message Passing Interface (MPI) that it itself was built against. This is required because the MPI handle is passed from the parallel simulation code to the ParaView libraries. ParaView Environment Modules were developed that set the PARAVIEWLIBPATH environment variable, which can trigger the build and linking of the in situ adapter. If it is not set the simulation will simply build without the in situ capability.

The major stakeholders in ParaView Catalyst agreed on a standard work flow for ParaView Catalyst simulation end users. The primary difficulty was with the visualization definition file. Although a savvy user could produce a python based visualization definition, most users would be unable to do this. Generic visualization definition files are made somewhat difficult as there is a need to register variable names. That is scalar and vector fields need to have their names consistent so they can be referenced in the visualization definition script. For instance 'pressure' might be called 'pressure', 'prs', 'Prs', 'rho', 'press', etc... The process to register variables between the simulation itself, the in situ adapter, and the visualization definition file requires the user to:

1. Run simulation with in situ to produce data sets on disk.
2. Make visualization definition scripts with ParaView Catalyst Plug-in.
3. Run in situ.

A user must first use a python visualization definition file that writes the representation provided by the Catalyst in situ adapter to disk. Once that is written to disk, it must be post processed to develop a visualization pipeline to produce desired in situ output. This is done using the ParaView client. If the data set is sufficiently small, ParaView on a local machine is sufficient. For larger data that won't fit in the memory of a single computer, a client-server connection must be established between the ParaView Client and a parallel ParaView with access to a time step. In either case, the development of the pipeline will be the same once the data set is loaded into the ParaView tool.

ParaView's co-processing plug-in allows the export of a visualization definition file. ParaView Catalyst provides a straightforward means to export the visualization definition file. A series of dialogs guide the user through the process of making choices as to the desired output for the in situ capability and then will save a python based ParaView Catalyst definition file.

An in situ adapter integrated into a code will have a mechanism for communicating the name of the visualization definition file to the Catalyst capability. This will likely occur through the input deck of the simulation.

### 3.7.3 Move To Production

ParaView is a parallelization of the Visualization Toolkit (VTK) [49]. The premise is a parallel visualization tool that supports distributed memory parallelism throughout the entire visualization pipeline. In particular data sets that are otherwise too large to fit in the memory of a single computer can be partitioned into pieces that can each be operated on independently and analysis products can be recomposed when necessary. Management support for the milestone provided sufficient drive by the stakeholders from desktop and high performance computing to support the roll out of ParaView on their respective systems. Section 3.7.3 will discuss the ParaView Desktop client and issues bringing it to production, while Section 3.7.3 will discuss those issues for the HPC side and finally, Section 3.7.3 will discuss issues surrounding the joining of these two production capabilities.

#### Production Desktop ParaView

Since its inception, ParaView has been heavily used by the visualization research community. A substantial amount of the work presented here is based on moving this tool from a predominantly research software package with an individual sophisticated user base to full on production visualization tool with documentation, training and consistency.

A ParaView client is required to run on the user desktop as a necessary part of the in situ work flow. The ParaView client version must match the parallel ParaView server version running on the supercomputer. A problem was identified on all of the LANL networks: not all end users were able to easily get a proper ParaView client on their desktop. The result was that every user needed to make a custom request to their desktop support staff for a specific version of ParaView to be installed. The solution was to coordinate with the institution and ParaView packagers at Kitware to

support the institutional desktop computing systems' ability to roll out multiple and simultaneously supported versions of ParaView. The effort of coordination involved in delivering ParaView to High Performance Computing (HPC) systems and Desktops has resulted in establishment of a more rigorous release schedule for ParaView in addition to regular meetings and long term planning amongst the stakeholders to ensure that capability and roll out of future ParaView versions will be increasingly smooth.

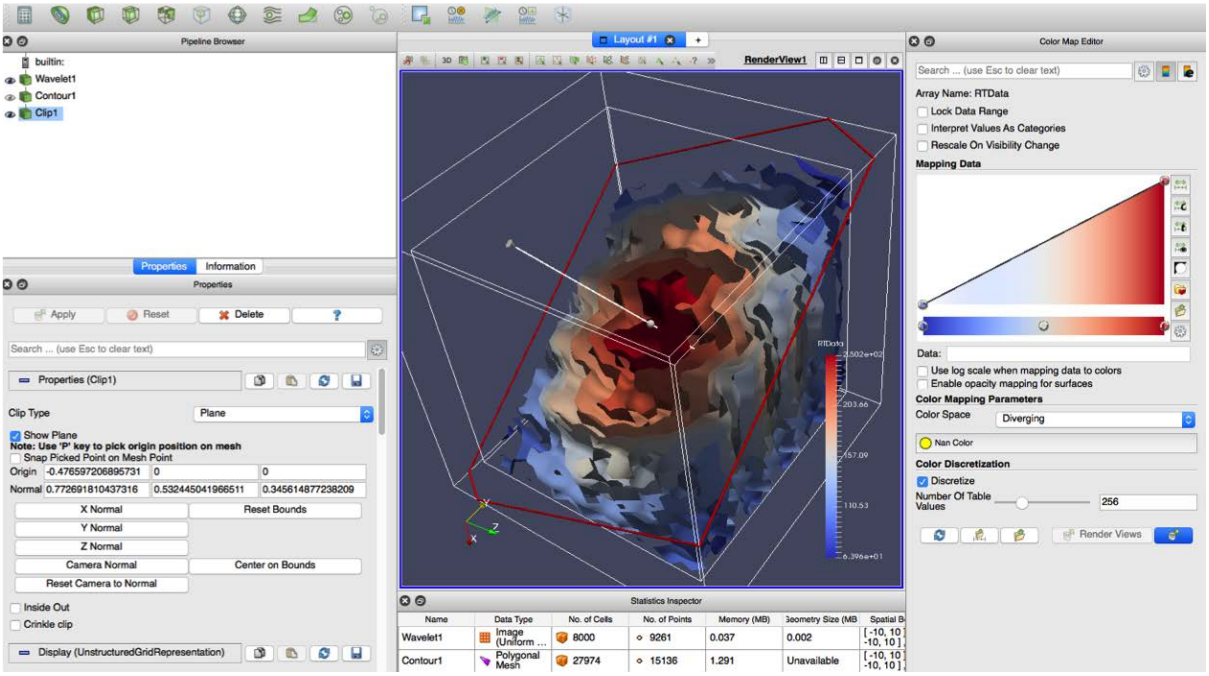



Figure 3.10: The ParaView Graphical User Interface provides the view and the control to the ParaView model.

The ParaView client is a sophisticated piece of software that can connect to parallel servers and drive them to produce visualizations with exacting control. Figure 3.10 shows the ability to change color maps, see spreadsheet views of selected raw data and control a vast amount of details regarding the finished visualization.

### Production HPC ParaView

LANL supercomputers provide environment modules for ParaView built against a variety of compilers and MPI versions that are deemed important. This work enables an end user to load a compiler, an MPI, and then a ParaView which simply loads the ParaView environment for the compiler and the MPI.

A "super build" of ParaView is used for generating the cross product of necessary builds against compilers and MPI versions. The super build contains an automated mechanism for building a variety of ParaView dependencies to easily integrate the parallel ParaViews into the production-computing environment. Code for producing super builds is stored in git repositories. Parallel builds are now the responsibility of the Parallel Tools Team, who provide production parallel software and support for supercomputers. Once a ParaView version has been built and its modules are installed on each of the supercomputers, the Parallel Tools Team will ensure that software is available for the lifetime of the supercomputer. Simulation code teams can depend on those builds existing.

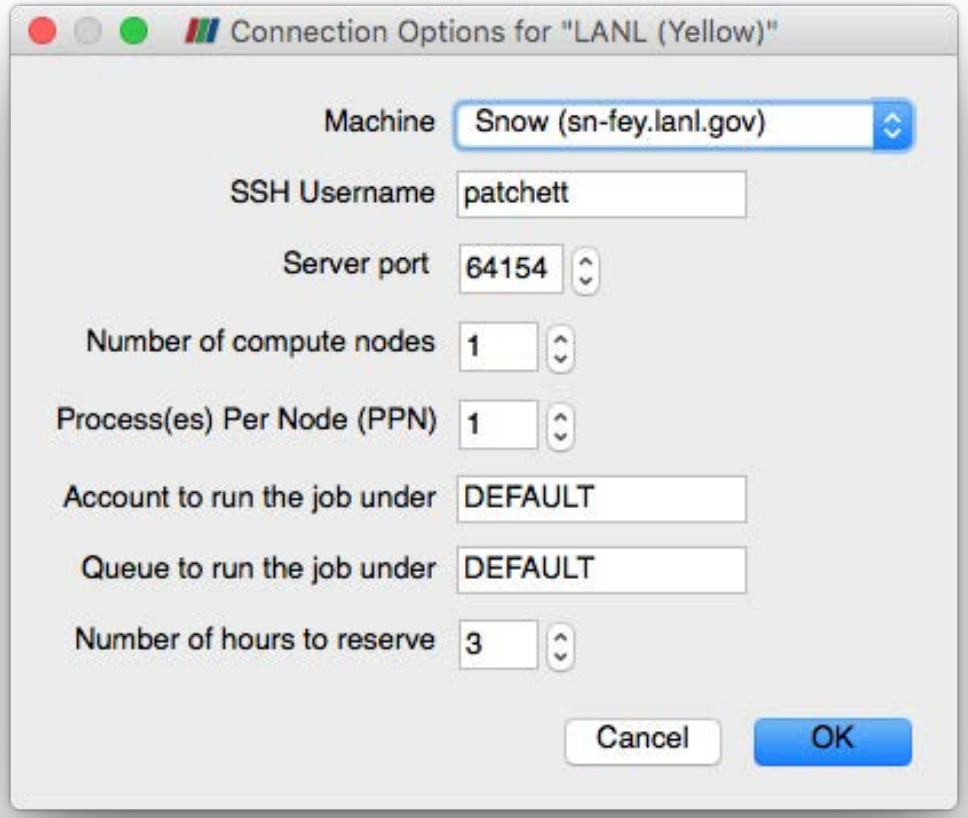



Figure 3.11: The ParaView remote connect dialog for LANL's network requires a one time setup by the user.

LANL HPC Consultants and Parallel Tools teams developed an automated connection system between the desktop and the supercomputer. The configuration of this system is documented on internal web pages easing the burden on users requiring parallel visualization. Websites with documentation have been instrumental in our success and user training sessions have been full. If users are having problems configuring, starting up, or otherwise using the interactive client server capability they can now call standard technical support, accessible to all supercomputing and desktop users. Technical support staff can triage ParaView issues. Figure 3.11 shows the interactive dialog the LANL ParaView GUI uses when attempting to connect. The supercomputer, accounts, queues and the total run time are configurable. Versions of the ParaView client, the parallel server and the Catalyst library linked to by the simulation must match.

Desktop and Information Technology support are completely independent of the supercomputing maintenance and support. Management for coherency is currently done in an ad hoc manner, but it is fully expected that improvements will be made as more iter-

ations of production ParaView software are performed. LANL prefers specific packaging for software distributions that allow their automated systems to automatically install, update, validate, and remove specific software packages. There are many details that have or are being dealt with including proper signing of mac binaries using a certified Apple Developer ID and produce packages with version control systems that allow multiple versions rather than upgrades through LANL's infrastructure. Windows clients automated package testing uses virtual machines with insufficient OpenGL library versions. Coordinating software versions across HPC and desktop environments is a new problem for the LANL support personnel. LANL is using its influence through a contract with Kitware, Inc. and its relationships with other facilities and institutions that use ParaView to start releasing packaged software for these environments.

Although there is a lot of work yet to do and consistency to establish, ParaView is available for an interactive, post processing use case at LANL today. It is also available for a large assortment of MPI and compiler combinations to support the array of simulation codes that wish to use it for the in situ use case. Software management systems allow end users to select ParaView as an installation option and supercomputers supply environment modules to match.

#### Production Simulation

Simulations were run using xRage [23], a parallel multi-physics Eulerian hydrodynamics code developed and maintained by the ASC program at LANL.

### 3.7.4 Use Case I

A friendly user was found wanting to pursue the ability to study opacities at the locations of Lagrangian tracers during the simulation run. The opacity information is a large variably sized vector that makes it difficult to produce full resolution data sets for post processing. The ParaView Catalyst adapter was augmented to support information extraction for the set of Lagrangian tracers that exist in the simulation, in particular the opacity information used in that time step to advance the simulation. Combined, this allowed the creation of visualizations showing both a tracer location and opacity information tied together for each Lagrangian tracer, a new capability for users. A set of imagery was provided in the form of a Cinema database [40]. Figure 3.12 shows an example of the problem. There are many tracers, each of which is represented as sphere glyphs and colored by ID (1-63). The large red sphere represents the tracer chosen and Figure 3.13 shows the detailed opacity absorption and scattering information at that tracer location. With this solid foundational capability in place, user training to create self sufficiency could occur. The ParaView Catalyst in situ adapter already contained the ability to iterate over the Lagrangian tracers and translate their locations and associated data as fields in the VTK multi block data set produced by the adapter. The opacity information was not available to the adapter. Even if the opacity information was available, the end user was interested in looking at a large number of simulation cycles and the simulation decisions that were getting made based on the opacity information which is non linear and difficult or impossible to calculate post facto. This creates a viewing and analysis problem which was solved by creating a Cinema database and an associated Cinema viewer.

This work marks the first ever, LANL, official delivery of data products via a Cinema

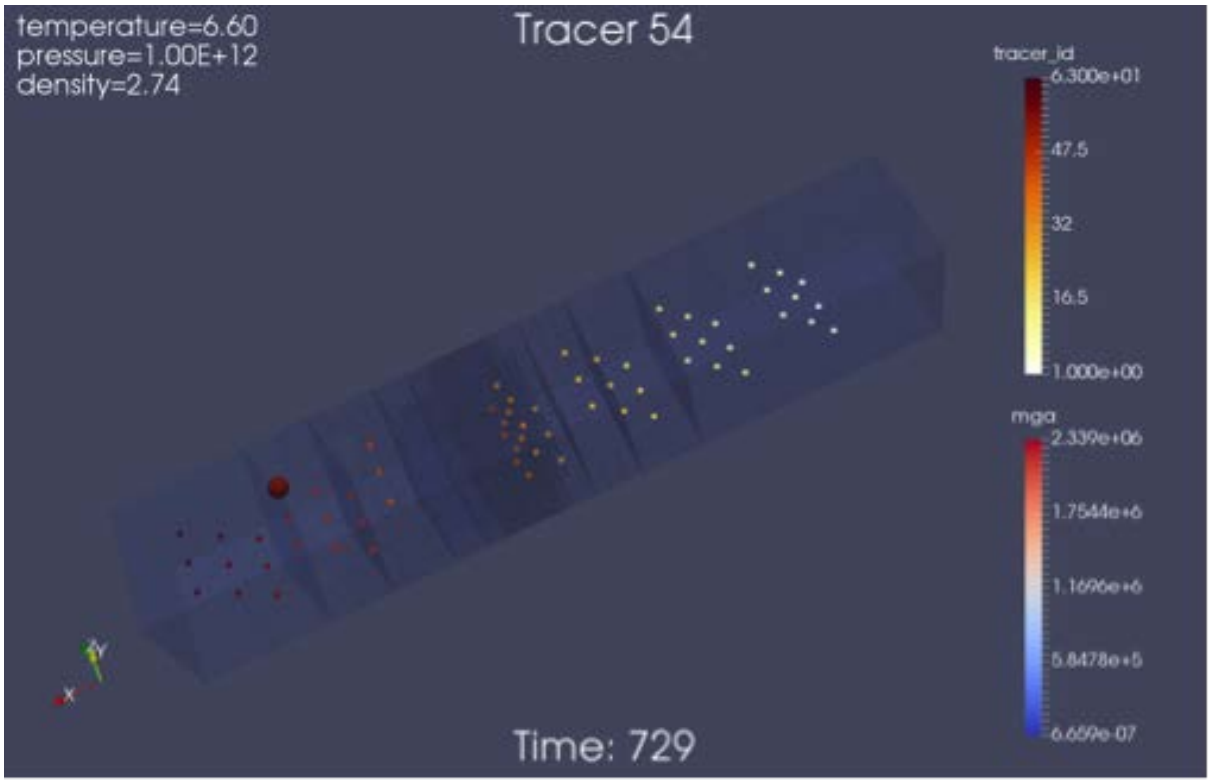



Figure 3.12: Plot of selected tracer, displays location, temp, pressure, density and location of other tracers in the simulation space.

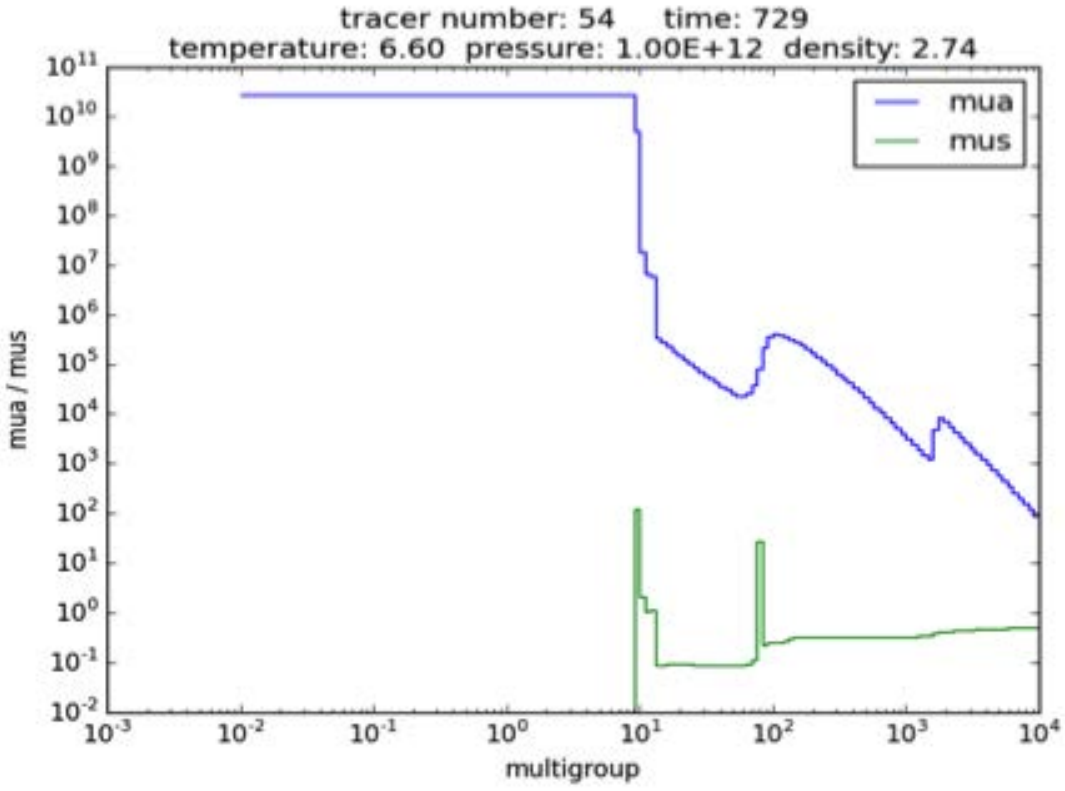



Figure 3.13: Plot of absorption and scattering at the highlighted Lagrangian point.

database. Each time step in the data set contained two blocks of a multi block data set, the first block was the raw simulation grid and simple variables like pressure and temperature. The second block had a list of 63 tracers, plus their associated simple variables and their opacity information.Although ParaView can rapidly render 2D plots such as those shown in Figure 3.13 and it is designed to rapidly render 3D imagery like that shown in Figure 3.12, it did not natively have the capability to tie the tracer selection in the 3D view to the 2D plot showing information about that tracer. In order to deliver the data base, the visualization dumps were post processed to create cross products of imagery for each time and tracer location for both the 3D visualizations and the 2D plots.

This delivery helped expose design requirements for future Cinema based system. This included fast rendering of 2D plots and support for 2D plot information. It also made clear a need for a spreadsheet view with linking across arbitrary axes. These issues were discussed with the Cinema specification development team and are now fixed in the most recent heads of ParaView and in the Cinema specification.

### 3.7.5 Use Case II

A user from the LANL asteroids team integrated our in situ ParaView Catalyst capability into his work flow to help in analyzing simulations of asteroids entering the Earths atmosphere and impacting deep ocean water. The user used visualizations, generated in ParaView with data produced by the ParaView Catalyst capability in xRage, to explain his results and show that simulation results supported his conclusions. The capability enabled by the in situ adapter generated much excitement amongst our collaborators and a small team from the University of Texas, with Office of Science funding, a student, with LANL ISTI funding, and a variety of LANL scientists worked together to produce an SC 2016 Visualization Showcase (SC16, 2016) submission based on science research and simulation of these asteroid impacts in deep ocean water [15] in addition to others [5, 8].

Simulations run for the asteroid impact studies produced over 226 TeraBytes of data and ran on up to 2048 processors for up to 6 weeks each per ensemble member. In addition to the normal checkpoint restarts for each simulation, VTK multi block files were produced for several hundred time steps representing multiple states per simulated second. The ParaView catalyst in situ adapter translated the native adaptive mesh of xRage to VTK unstructured grid representations. One VTK Unstructured grid XML file with binary uncompressed data was produced per processor and written to a High Performance File System (HPFS). For the purposes of volume rendering, the unstructured data was coalesced on a large memory compute node and merged into a single unstructured grid representation. A cell to point algorithm was run placing data on the vertices of the unstructured grid. This grid was then sampled onto a variety of structured grid resolutions that enabled both interactive low resolution volume rendering for exploration and high resolution batch rendering for final imagery for video production and publication.

Our efforts with asteroid impacts has significantly paid off. Large quantities of data have now been run through the situ adapter. Bugs in our in situ adapter, dealing with ghost cell generation, were found and fixed after rendering artifacts became immediately apparent. The efficacy of the existing ParaView parallel writers was validated. The ability to produce imagery at run time and at scale was validated. This work has generated a lot of interest because of the topic area, the spatial and temporal resolutions from the simulation. Ongoing work with the domain scientist is progressing as evidenced by the open sourcing of the simulation outputs [24] which will allow researchers from around

the world access to these large unstructured grids, improve algorithms that work on them and improve on the visualizations that have already been done. Patchett, et al. [15] supplies notable results from this use case: the paper that accompanied the Best Scientific Visualization at Supercomputing 2016.

### 3.7.6 Conclusion of Delivery

An in situ capability was successfully contributed over a one year period at LANL. The delivery was successful because of buy in from management, code development teams, production desktop, the production supercomputing, training, and users. At least one LANL simulation code is regularly built with an in situ capability and a small subset of users are now demanding the code be built and linked with ParaView Catalyst. This work has made the catalyst libraries available for building simulation codes. Effort is still being made to ensure that various compiler and MPI versions are supported. ParaView Catalyst has been driven by LANL users to be improved. In particular the promise of the Cinema specifications have driven further development in the PareView Catalyst support for generating cinema databases in situ. As the technology of catalyst becomes more mature, the simple use case of writing VTK files for post processing is a big win because the simulation code teams do not have to maintain writers that catalyst provides.

## 3.8 Core References

John M Patchett, Boonthanome Nouanesengsy, G Gisler, J Ahrens, and H Hagen. In situ and post processing workflows for asteroid ablation studies. In *Eurographics Conference on Visualization (EuroVis)*, 2017

John M Patchett, Boonthanome Nouanesengsy, James Paul Ahrens, Michael Kenneth Lang, David Honegger Rogers, Jennifer Kathleen Green, Francesca Samsel, Giovanni Antonio Cone, and Hans-Jurgen Hagen. Delivery of in situ capability to end users. *Visualization in Practice*, 2017

# Chapter 4

# Optimizing File Access Patterns using Spatio-Temporal Parallelism

**Abstract**

For many years now, I/O read time has been recognized as the primary bottleneck for parallel visualization and analysis of large-scale data. In this chapter, a model that can estimate the read time for a file stored in a parallel file system when given the file access pattern is presented. The file access pattern will be dictated by the type of parallel decomposition used. Spatio-temporal parallelism, which combines both spatial and temporal parallelism, to provide greater flexibility to possible file access patterns is employed. This work enables the configuration of the spatio-temporal parallelism to design optimized read access patterns that result in a speedup factor of approximately 400 over traditional file access patterns.

## 4.1 Related Work and State of the Art

The visualization and analysis of large-scale data in a timely manner has been a recognized problem for many years now. With computational power increasing and the introduction of more sensitive instrumentation, data from both simulations and experiments are expected to continue to grow. The result of these trends are ever larger data sets containing higher spatial and temporal resolutions. The standard method to tackle the large-data problem is to use parallel processing. In practice, the main bottleneck in large-scale parallel analysis is the I/O read step. One of the main factors in I/O performance is how the file is accessed, and what the read pattern is. The chosen parallel decomposition strategy determines the read pattern.

For many of the common visualization tools used by the community, parallel processing implies using a data-parallel approach employing only spatial parallelism. In this approach, the data is partitioned spatially and spread out across several processes. Each process then applies the same computation on their piece of data. Another option for data decomposition, temporal parallelism, involves processing data from different time steps simultaneously. The same computations are applied to each time step. Often overlooked is how different decomposition approaches affect the read pattern in files, despite the fact that I/O performance is usually the performance bottleneck.

Attempts have been made to address I/O performance for visualization of large data sets. Some have improved I/O performance for post processing by using systems connected to faster storage such as solid state drives (SSDs) [50] [51]. Mitchell *et al* [52],

using VisIO, realized performance gains over the traditional high performance parallel file system by extending the ParaView system to support the Hadoop file system. Pre-processing data before post-processing has been shown capable of lessening I/O bandwidth requirements. Woodring *et al* [18] encode raw data in a JPEG 2000 format to enable multi-resolution streaming over low bandwidth connections. In-situ, in-transit, and hybrid combinations of these two paradigms have been used to lessen the necessity of post-processing, mitigating associated I/O issues [35] [53] [54].

Parallel post-processing of climate data is of major concern. Woitaszek *et al* [55] gained performance by parallelizing a post-processing work flow for climate data using the Swift scripting language, with parallelism only scaling to 32 processes. On the other hand, our work is focused primarily on I/O performance scaling to thousands of processes.

Both spatial and temporal parallelism have been studied previously. Yu *et al* [56] [57] use a spatio-temporal scheme to achieve high I/O performance, but does not model the read step. Childs *et al* [58] showed through a series of large parallel visualization experiments that pure parallelism analysis operations work at extreme scales, but I/O times became very large, suggesting that the I/O performance required further study. The TECA project [59] reports good temporal parallelism performance for climate data, but has little parallel I/O or native spatial parallelism support, which creates a problem for data with large spatial bounds. While spatio-temporal parallelism is not new, our work focuses on the connection between the resulting read access pattern and the I/O performance. Our model explains why end-to-end scaling of pure spatial or pure temporal pipelines are suboptimal.

Work has been performed to enable developers and end users the ability to alter data decompositions and thus affect read times. Kendall *et al* [60], using their BIL (Block I/O Layer) software, show considerable performance gains by aggregating smaller requests into larger requests which cover more contiguous regions of the file on disk (two-phase collective I/O). Peterka *et al* [61] provide a generalized library for building visualization algorithms on top of configurable domain decompositions, impacting I/O access patterns which are executed using a variety of library access methods, including BIL. Our contribution of a model will help users of these tools and systems make more wise data decomposition decisions.

Biddiscombe *et al* [62] introduced the concept of time to the ParaView pipeline. While this work enabled the serial processing of spatially decomposed time steps, our work enables decomposition in both space and time, allowing for the simultaneous processing of multiple time steps.

## 4.2 Approach

In this chapter, we contribute a model which estimates the total running time of a parallel visualization and analysis pipeline. We also contribute the implementation of a pipeline capable of spatio-temporal parallelism, called the spatio-temporal pipeline, in a general purpose visualization tool. Our model can estimate the time needed to load a file stored in a parallel filesystem when given the file access pattern. The goal of our model is to provide a useful way of comparing various work flow choices, rather than accurately predicting exact running times. Using this model, we can determine which parallel approaches are best suited for a given file format and pipeline. To offer more control over the file access pattern, we implement a method that employs both spatial

and temporal parallelism, called the spatio-temporal pipeline. In the spatio-temporal pipeline, processes are separated into groups called time compartments. Temporal parallelism is employed as time compartments independently process a time step concurrently. Each time step is partitioned spatially over all processes within the time compartment, thus the spatio-temporal pipeline uses both spatial and temporal parallelism. The spatio-temporal pipeline is a fully integrated feature in ParaView and UV-CDAT (Ultrascale Visualization - Climate Data Analysis Tools). ParaView is a general-purpose parallel visualization tool, and UV-CDAT [1] is a visualization and analysis tool specializing in large-scale climate-data analysis.

Time-varying data can be stored in a myriad of different formats. How the data is stored and what access patterns are used to read in the data play a critical role in I/O read performance. In this chapter, I focus the discussion to a very common file format: the time-varying data is composed of files, and each file represents one scalar field at one time step.

The performance differences between temporal, spatial, and spatio-temporal parallelism may not be readily apparent. Indeed, if all components of a parallel system were to scale perfectly, there should be *no difference in running time* between using spatial parallelism, temporal parallelism, and spatio-temporal parallelism, for equal amounts of parallelization. In practice, there is a difference between various parallelization schemes because each method creates a distinct read access pattern, affecting the I/O read time. Ultimately, using spatio-temporal parallelism allows for greater flexibility in selecting a file access pattern that will provide greater I/O performance. The model is used to determine which file access pattern maximizes performance.

### 4.2.1 The Spatio-temporal pipeline

Spatial parallelism is a decomposition in which data is spatially partitioned over all available processes. Each process then applies the same set of computations on its piece of data. When employing spatial parallelism, time steps are processed in serial, i.e. time step 0 is processed, then time step 1 is processed, etc. One side effect of spatial parallelism is that increasing the number of processes results in each time step being spatially partitioned into more pieces, and each piece becomes smaller. According to the read model outlined in Section 4.2.7, this behavior adversely affects the read pattern and impairs I/O performance.

Temporal parallelism is a method in which multiple time steps are processed in parallel. This is a form of pipeline parallelism, in which multiple pipelines are instantiated in order to process multiple inputs at once. Temporal parallelism usually requires large amounts of memory, as each process will load an entire time step.

patio-temporal pipeline was designed to utilize both spatial and temporal parallelism, which allows for more control of the access pattern used to read files. Spatio-temporal parallelism is accomplished by first partitioning all available processes into groups called time compartments. Each time compartment is responsible for processing time steps, and performs computations independently of each other. Each time step is spatially partitioned over all processes within the time compartment. If there are more time steps than time compartments, then a time compartment will process multiple time steps. For example, if there are two time compartments and six time steps, then each time compartment will process three time steps. Each time compartment loads one time step at a time, and when a time step is finished the next available time step is then loaded.

For our implementation, each time compartment contains the same number of processes.

In the spatio-temporal pipeline, the ratio between spatial and temporal parallelism can be changed by adjusting the number of processes in a time compartment. Assuming the number of total processes is constant, if the time compartment size is large, then there are few time compartments overall. This leads to lower temporal parallelism, since fewer time steps are processed concurrently, and higher spatial parallelism, since each time step will be partitioned into more pieces. On the other hand, if the size of a time compartment is lowered, the total number of time compartments becomes higher. This allows for more time steps to be processed in parallel, thus temporal parallelism is increased, while lowering the number of pieces each time step is split into, resulting in less spatial parallelism.

Because it is possible for multiple time steps to be processed concurrently, one restriction of the spatio-temporal pipeline is that time steps must be able to be processed independently. Only operations which do not require any communication between time steps can be used. Examples of such operations include computing the isocontour of each time step and creating an image for each time step.

Despite a large number of visualization and analysis algorithms requiring no communication, there are still some operations in which the spatio-temporal pipeline is incompatible. Time-dependent operations which are not associative and require processing through time steps in a certain order are currently not supported in the spatio-temporal pipeline. This class of operations include pathline advection and Finite-Time Lyapunov Exponent (FTLE) computation.

### 4.2.2 Models

As mentioned earlier, large-scale visualization and analysis tasks are usually bottlenecked by the I/O read step. The chosen parallel decomposition approach will determine what the file access pattern is, which greatly affects I/O performance. In order to illuminate how best to configure the spatio-temporal pipeline to get an optimized read pattern, we developed a model of a visualization pipeline. The goal of our model is to compare and determine which parallelization scheme will provide the best performance. A model for the spatio-temporal pipeline is introduced, as well as one for a pipeline using only spatial parallelism.

For modeling purposes, we use the following pipeline:

$$read \rightarrow isocontour \rightarrow write\ isocontour \tag{4.1}$$

We assume that there is a time-varying data set stored in the format of each file containing one time step of a scalar field. Each time step needs to be loaded from disk. Once the data is loaded into memory, an isocontour is generated. Then the resulting isocontour is written to disk.

### 4.2.3 Assumptions

Certain assumptions are made with the models:

- Each file is one time step containing one scalar field
- Isosurfacing and writes have perfect parallel scaling

- The number of processes allocated per node ($ppn$) is constant
- Each process is run on one core
- In the spatio-temporal pipeline, the total number of processes is evenly divisible by the time compartment size
- In the spatio-temporal pipeline, each time compartment spans the same number of nodes

### 4.2.4 Definitions

The following variables are used in the models.

- $n$ is the total number of nodes used
- $ppn$ is the number of processes per node used
- $p$ is the total number of processes, found by $p = n \cdot ppn$
- $sf$ is the size of each file
- $pf$ is the number of processes used to open one file
- $nf$ is the total number of files in the data set
- $mf$ is the maximum number of files any process will touch
- $bw$ is the bandwidth available to each node
- $bwp$ is the bandwidth available to each process, found by $bw/ppn$
- $tc$ is the time compartment size

### 4.2.5 Spatial Parallelism Model

The spatial parallelism model is based on a decomposition that uses only spatial parallelism, in which all processes are involved in processing each time step. Therefore, in the spatial parallelism model, $mf = nf$. The total time to compute the pipeline, $T_{total}$, is found by

$$T_{total} = T_{read} + T_{iso} + T_{write} \tag{4.2}$$

where $T_{read}$, $T_{iso}$, and $T_{write}$ are the time taken in each respective step in the pipeline.

Let us first consider the pipeline steps other than $read$ (modeling for the $read$ stage is addressed in Section 4.2.7). These steps are assumed to have perfect linear scaling. Since each file goes through the pipeline, each stage is encountered $mf$ times,

$$T_{iso} = mf \cdot T_{iso_p} \quad , \quad T_{write} = mf \cdot T_{write_p} \tag{4.3}$$

Where $T_{iso_p}$, $T_{write_p}$ is the time each respective step takes when $p$ processes are operating in parallel on one time step. Since these stages are assumed to have perfect linear scaling, the time for each of these stages can be computed with the following equations:

$$T_{iso_p} = \frac{T_{iso_1}}{p} \quad , \quad T_{write_p} = \frac{T_{write_1}}{p} \tag{4.4}$$

Therefore, $T_{total}$ can be characterized by the equation:

$$T_{total} = T_{read} + mf \cdot \left\lceil \frac{T_{iso_1} + T_{write_1}}{p} \right\rceil \tag{4.5}$$

### 4.2.6 Spatio-Temporal Parallelism Model

In the spatio-temporal pipeline, processes are divided into time compartments. Each time compartment runs in parallel and acts independently of each other. Therefore, the total running time will be the maximum time any time compartment takes. This is equivalent to a time compartment processing $mf$ files. In the spatio-temporal model, $mf$ is found using the equation:

$$mf = \left\lceil \frac{nf}{p \div tc} \right\rceil = \left\lceil \frac{nf \cdot tc}{p} \right\rceil \tag{4.6}$$

Similar to the spatial parallelism model, the total time is the sum of each step in the pipeline.

$$T_{total} = T_{read} + T_{iso} + T_{write} \tag{4.7}$$

For all stages except read, the time of each step is

$$T_{iso} = mf \cdot T_{iso_{tc}} \quad , \quad T_{write} = mf \cdot T_{write_{tc}} \tag{4.8}$$

Similarly to equations 4.4,

$$T_{iso_{tc}} = \frac{T_{iso_1}}{tc} \quad , \quad T_{write_{tc}} = \frac{T_{write_1}}{tc} \tag{4.9}$$

Therefore, $T_{total}$ can be written as

$$T_{total} = T_{read} + mf \cdot \left\lceil \frac{T_{iso_1} + T_{write_1}}{tc} \right\rceil \tag{4.10}$$

How to model $T_{read}$ is discussed in Section 4.2.7.

### 4.2.7 Read Performance Model

We now model the read times of both the spatial parallel model and spatio-temporal model. Note that $pf$, the number of processes used to open one file, is different for each model. For the spatial parallel model, $pf = p$, while for the spatio-temporal model, $pf = tc$.

First, we start with the read time for one file, $T_{read_1}$, assuming perfect linear scaling. In this case, the read time is the size of one file divided by the total available bandwidth.

$$T_{read_1} = \frac{sf}{bwp \cdot pf} \tag{4.11}$$

For both the spatial parallel and spatio-temporal methods, the maximum number of files read by any process is $mf$, so the total read time for $mf$ files is

$$T_{read_{mf}} = mf \cdot \left[ \frac{sf}{bwp \cdot pf} \right] \tag{4.12}$$

In general, the use of parallelism does not scale perfectly. There is always overhead associated with parallel algorithms, whether it is communication or load imbalance. Since each file is spatially decomposed and read in parallel, we expect there to be overhead for each file read.

$$T_{read_{mf}} = mf \cdot \left[ \frac{sf}{bwp \cdot pf} + overhead \right] \tag{4.13}$$

A parallel file system is a complicated system with many variables and parameters that could affect its performance. In general, a good rule of thumb is that the best I/O performance can be achieved by using contiguous reads. As the number of contiguous reads decrease and the number of file seeks increase, I/O performance will be impaired.

Thus, we base the overhead on the number of file seek operations required to read the file. We assume that the file is written to disk in such a way that spatial coordinates of the x-axis changes fastest, then the y-axis, and finally the z-axis. With this file format, partitions can be read row by row. Therefore the number of seeks can be estimated as the number of rows in a partition, which we denote as $ns$ (number of seeks). Though this is not necessarily the actual number of seeks the disk will perform in a parallel filesystem, we find it is a good estimate. We also believe that as the number of concurrent processes used to read a file grows, the read performance degrades due to increased contention. Because of this, the overhead is also based on the number of processes used to read in a file. Therefore, the final equation for overhead becomes

$$overhead = \alpha \cdot ns + \beta \cdot pf \tag{4.14}$$

Both $\alpha$, the time to perform a seek, and $\beta$, the amount of contention introduced per process, are free parameters that are based on the hardware characteristics of each machine. The final equation for the read step now becomes

$$T_{read_{mf}} = mf \cdot \left[ \frac{sf}{bw \cdot pf} + \alpha \cdot ns + \beta \cdot pf \right] \tag{4.15}$$

sectionresults

### 4.2.8 Analysis of Models

One of the most important inferences we can make from this model is how each method scales as the number of files and processes increase. From the models, we can infer the general performance trend of a weak scaling study (actual results of a weak scaling study are discussed in Section 6.3.2). Since it is assumed that the isocontour and write step scale linearly in both methods, the major difference will be how the read step behaves.

As the amount of work and number of processes increase, we expect the spatial parallelism method to incur more read overhead per file. This is because as the number of processes grows, the number of processes used to open a file increases. Each individual file will be spatially split into more partitions. This will increase both the $ns$ and $pf$ terms in Equation 4.14. Thus using spatial parallelism will result in worse file access patterns as the number of processes grows.

For the spatio-temporal pipeline, as weak scaling increases, the time compartment size is kept constant, which means more time compartments are added to process the increased number of files. For example, assume an initial configuration of 4 files and 8 processes with a time compartment size of 4. When doubling to 8 files and 16 processes and a time compartment size of 4, the processes are split into four time compartments, each composed of four processes. In this situation, each file is still read by four processes, so the spatial partitioning remains the same, thus the number of seeks needed remains unchanged. Therefore the resulting overhead value of Equation 4.14 remains constant. Overall, we expect the spatio-temporal pipeline to scale perfectly in a weak scaling study due to the fact that the read pattern remains unchanged.

This perfect weak scaling of the spatio-temporal pipeline implies that reading in multiple files by applying the same read pattern to each file does not create any additional overhead for the filesystem. This assumes that the number of nodes used per file remains fixed so that the amount of bandwidth per file is constant.

## 4.3 Results

In order to verify the accuracy of our model, we performed several timings tests. We used two different climate data sets for these tests. The first data set is output from the Parallel Ocean Program (POP), a simulation of the entire ocean, which we refer to as the *POP* data set. The data is composed of salinity values, and up to 256 time steps were used. The spatial resolution of each time step is 3600 x 2400 x 42. Each file is 1.4 GB, for a total size of roughly 350 GB. The other data set used is the output from a CAM (community atmospheric model) simulation, which we refer to as *ATM*. Each time step has spatial resolution of 1152 x 768 x 30, and a total of 16 time steps were used. For both data sets, files are stored in NetCDF 3 format, and are read using the vtkNetCDFReader class in VTK. The vtkNetCDFReader utilizes the standard NetCDF libraries without any parallel I/O optimizations.

The tests involving the POP data set were run on *Mustang*, a supercomputer at Los Alamos National Laboratory. Mustang features nodes with dual-socket AMD 12-core MagnyCours and 64 GB of memory. Mustang uses the Panasas filesystem.

All tests with the ATM data set were run on *Hopper*, a supercomputer at the National Energy Research Scientific Computing Center (NERSC). Hopper contains two 12-core AMD MagnyCours and 32 GB memory per node. Hopper uses the Lustre filesystem.

### 4.3.1 Weak Scaling

Weak scaling studies were conducted using both data sets. The POP data set was run on Mustang. Tests began at one file and 8 processes, and doubled until 256 files and 2048 processes were reached. The number of nodes allocated was equal to the number of files, with 8 cores per node being used. The time compartment size was set to 8 for all tests. With this configuration, each time compartment consisted of 8 processes all located on the same node, and each node held only one time compartment. Each time compartment was responsible for processing one file. Each file was first loaded into memory, then an isocontour was generated, and finally the isocontour was written to disk.

The weak scaling tests using the ATM data set were similarly configured. All tests were run on Hopper, and the time compartment size was chosen to be 24 for all tests. The number of files began at one and the number of processes started at 24. Subsequent

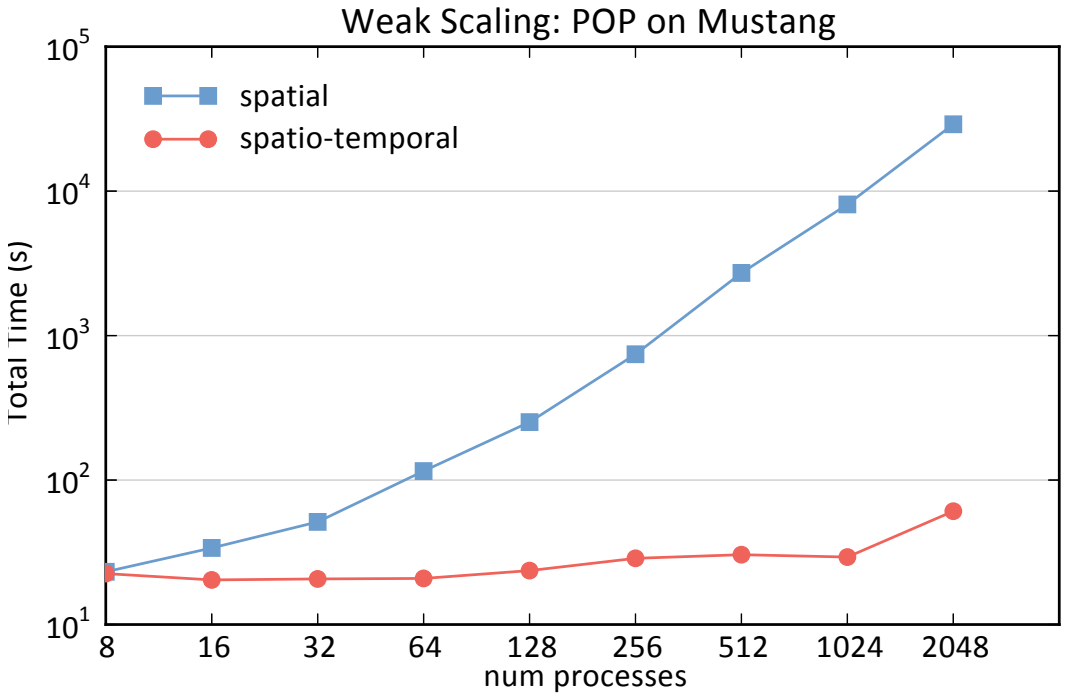



Figure 4.1: Weak scaling results between the spatio-temporal pipeline and spatial parallelism. Run on Mustang using the POP data set. The spatio-temporal pipeline with optimized read access patterns scale significantly better than the spatial parallelism method. At 2048 processes, there is a difference of a factor of over 400 between the two methods.

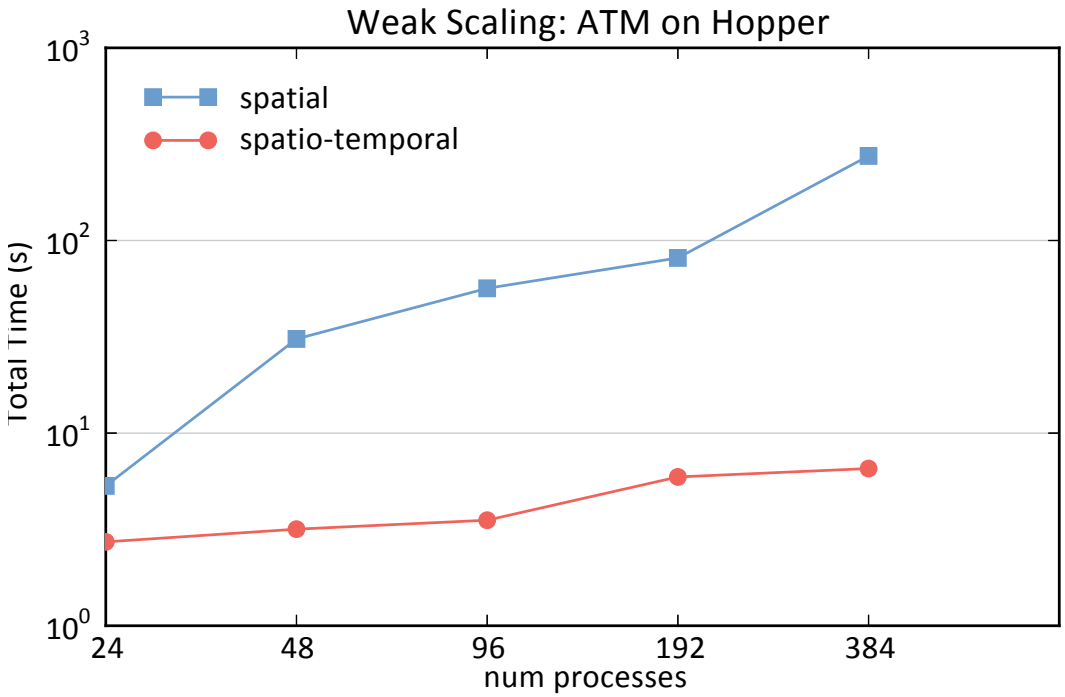



Figure 4.2: Weak scaling results between the spatio-temporal pipeline and spatial parallelism. Run on Hopper using the ATM data set. The performance of the spatio-temporal pipeline is orders of magnitude better than the spatial method due to the difference in file access patterns.

tests doubled these values until 16 files and 384 processes were reached. The number of nodes allocated was equal to the number of files processed, and 24 cores per node were used. Files were processed using the same pipeline as the POP tests described earlier.

Figure 4.1 shows the results from the POP tests on Mustang, and Figure 4.2 shows the results from the ATM tests on Hopper. For both tests, the spatio-temporal pipeline displayed significantly better performance and scalability. For the POP tests, at 256 files and 2048 processes, the spatial parallelism method required roughly 29,000 seconds (about 8 hours), while the spatio-temporal pipeline performed the same amount of work using only 60 seconds. This resulted in a speedup factor of over 480. The total time of the spatio-temporal method stayed relatively flat, beginning at 20 seconds, inching up to 30 seconds at 1024 processes, and jumped to 60 seconds at 2048 processes. We believe at 2048 processes, the maximum bandwidth of the system had been reached, thus the doubling of the time. Similar trends are shown for the ATM tests. The spatio-temporal method outscales the spatial method by up to two orders of magnitude. At 394 processes, the spatial method required 274 seconds, while the spatio-temporal method only took 6.5 seconds.

Given the three step pipeline of read, isocontour, and write, our models assumed that the isocontour and write step would scale perfectly. The models also predicted that the read step would increase in time when using only spatial parallelism, and would scale perfectly using the spatio-temporal pipeline due to differences between file access patterns. Figures 4.3 and 4.4 show the actual per component breakdown of the weak scaling results. The spatial parallelism results in Figure 4.3 show that the isocontour computation step does scale nearly perfectly. The write step begins to increase after 128 processes, but it never consumes more than 1% of the overall running time. As predicted by the model, the read step does steadily increase as the number of processes rise, dominating the overall running time. Thus, the read pattern that resulted by using spatial parallelism greatly impairs I/O read performance. For the spatio-temporal pipeline, Figure 4.4 shows all three steps scaling well until 2048 processes are used, in which case the read and write times increase. As mentioned earlier, we believe the increase in read times is due to bandwidth limitations, and the rise in write times may also be due to hardware limitations. Up to 1024 processes, the read times remained fairly steady, indicating that the spatio-temporal pipeline used a more optimal file access pattern. Overall, our models have predicted the general trends of both the spatial parallelism and spatio-temporal methods.

Our models not only let us discern the trends of different components, but also can be used to obtain an estimate of the total running time. Many variables, such as size of one file, number of files, and processes per node, are dependent on the run configuration. Other variables, such as *bwp* (the bandwidth available to each process), $T_{iso_1}$, and $T_{write_1}$, can be found by performing small timings tests on one node. The number of seeks, $ns$, can be calculated as the number of rows in each spatial partition. Once all these variables are obtained, they can be plugged into Equation 4.5 and Equation 4.10. The two free variables in Equation 4.15, $\alpha$ and $\beta$, are then found by finding the best fit of the modeled times to some actual results on the same machine.

Figure 4.5 compares the total time estimated from the model and the actual results of the POP weak scaling tests. It was empirically found that $\alpha = 7 \times 10^{-6}$ and $\beta = 1 \times 10^{-3}$ provided the best fit. For the spatial parallelism method, the modeled time tracks fairly close to the actual results. For the spatio-temporal method, the model predicts perfect scaling, so the modeled time is a flat line in the graph. The actual times track the

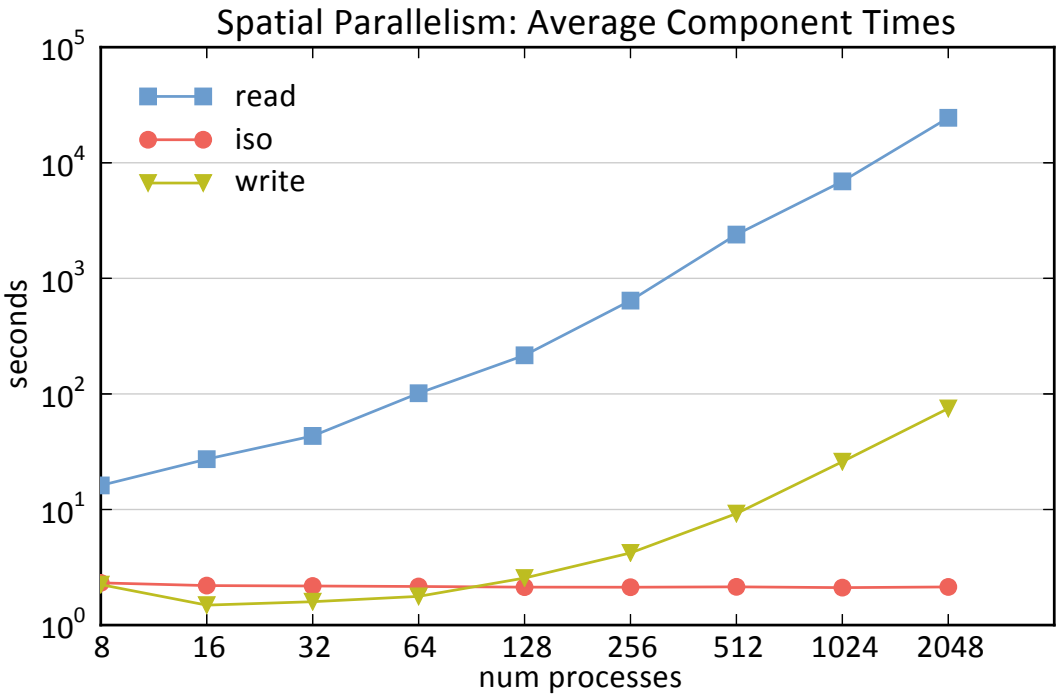



Figure 4.3: Per component breakdown of results of the weak scaling tests on the POP data set for spatial parallelism.

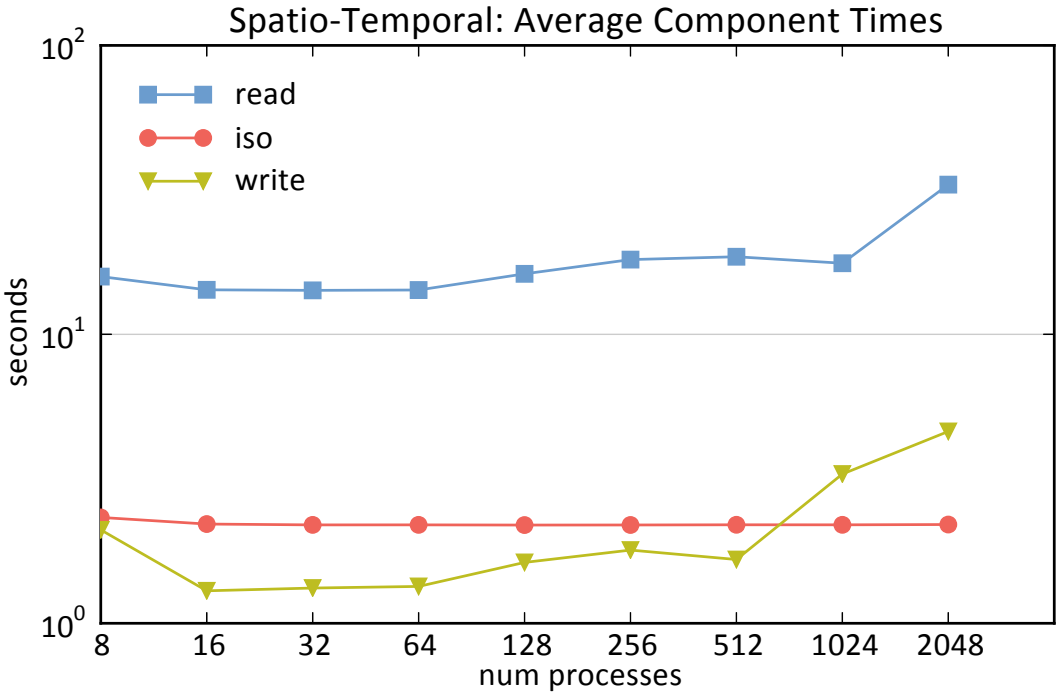
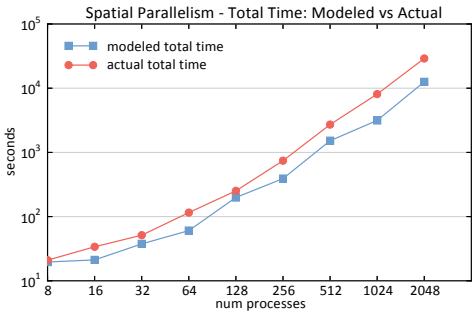
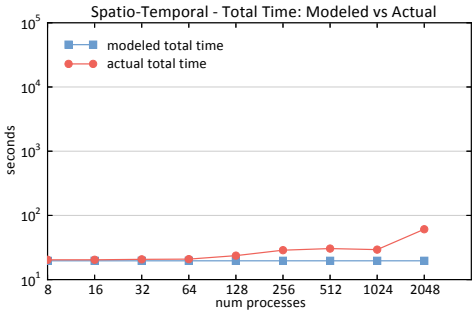



Figure 4.4: Per component breakdown of results of the weak scaling tests on the POP data set for the spatio-temporal pipeline.




(a) Spatial Parallelism




(b) Spatio-Temporal Parallelism

Figure 4.5: Modeled total times versus actual total times for the spatial parallelism and spatio-temporal weak scaling tests on Mustang for the POP data set. The model was used with $\alpha = 7 \times 10^{-6}$ and $\beta = 0.001$. Overall, the modeled time duplicates the trends seen in the actual times.

modeled times well, especially at low number of processes. At 2048 processes, the actual time spikes up, but as stated earlier, we believe this is due to bandwidth limitations, which the model does not account for.

The modeled and actual total time of the ATM weak scaling tests are shown in Figure 4.6. A best fit was found by using $\alpha = 5 \times 10^{-5}$ and $\beta = 1 \times 10^{-4}$. For the spatial parallelism method, the modeled times track well with the actual times. The greatest difference is at 48 processes, where the model estimate was 30 seconds and the actual time was 11.5 seconds. For the spatio-temporal method, the model always overestimates the total time, but the actual difference is small. Overall, our models predicted the total time of both the POP and ATM tests fairly accurately.

## 4.4 Core References

Boonthanome Nouanesengsy, John Patchett, James Ahrens, Andrew Bauer, Aashish Chaudhary, Ross Miller, Berk Geveci, Galen M Shipman, and Dean N Williams. A model for optimizing file access patterns using spatio-temporal parallelism. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 4. ACM, 2013

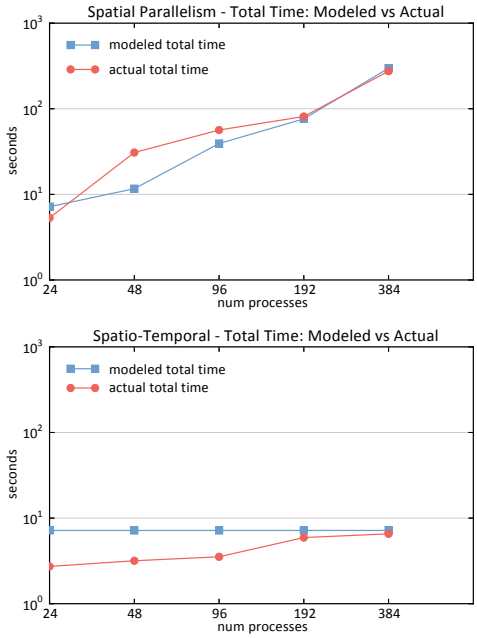



Figure 4.6: Modeled total times versus actual total times for the spatial parallelism and spatio-temporal weak scaling test on Hopper for the ATM data set. The model was used with $\alpha = 5 \times 10^{-5}$ and $\beta = 0.0001$. The modeled times are within the same order of magnitude, and track the actual times well at higher process counts.

# Chapter 5

# Cinema

**Abstract**

Extreme scale scientific simulations are leading a charge to exascale computation, and data analytics runs the risk of being a bottleneck to scientific discovery. Due to power and I/O constraints, it is expected that in situ visualization and analysis will be a critical component of these work flows. Options for extreme scale data analysis are often presented as a stark contrast: write large files to disk for interactive, exploratory analysis, or perform in situ analysis to save detailed data about phenomena that a scientists knows about in advance. A novel framework for a third option is presented– a highly interactive, image-based approach that promotes exploration of simulation results, and is easily accessed through extensions to widely used open source tools. This in situ approach supports interactive exploration of a wide range of results, while still significantly reducing data movement and storage.

## 5.1 Related Work and State of the Art

The supercomputing community has embarked upon a revolutionary path towards extreme scale ( 1015 FLOPS). Just as the massive computing power of these machines improve how we do simulation science, these new machines are changing how we analyze simulation results. Application scientists use visualization and analysis to understand and advance their science. At smaller scales, the data are stored or moved to another machine for post-processing. However, both the complexity and size of their scientific simulations continue to evolve as we advance en route to extreme scale computing platforms. With storage bandwidth significantly falling behind the rate needed to move data, standard postprocessing techniques will not be able to effectively scale in the future. Therefore, the fundamental extreme scale visualization and analysis challenge is there is too much simulation data and too little transfer bandwidth.

In situ techniques that analyze the data while it still resides in simulation memory show a promising path forward [63–66]. The same supercomputing resources that compute the simulation data are also used for the analysis and, as such, the data do not have to be moved. Typically, in situ approaches either are a predefined set of analyses or, rarely, make automatic decisions about which analyses and visualizations to create. Therefore, we see the goals of in situ visualization and analysis as multifaceted: 1) to preserve important elements of the simulations, 2) to significantly reduce the data needed to preserve these elements, and 3) to offer as much flexibility as possible for post-processing exploration.

Simulation results must be transformed from an extreme scale sized data space to a petascale storage system resulting in a massive compaction from sparse raw simulation data to dense visualization and analysis data. We envision the scientist using our framework to define which analyses are needed and a target data size bounds of the analysis results at the beginning of their simulation in situ run. Understanding the space of in situ visualization and analysis solutions within the context of data funneling process is a salient key to addressing the extreme scale challenge.

Our framework has number of contributions and, therefore, we review related work for these areas. In situ approaches are an important mechanism for visualization and analysis due to the cost of data movement and storage required for traditional post processing [67]. A key concern with in situ approaches is the need to maintain exploratory analysis capabilities despite the fact that data gathering occurs as a batch process at simulation run-time. For example, Woodring et al [11] saved a hierarchy of random samples from a particle-based simulation to create a flexible representation for later analysis.

Interactive Exploration Database. A key component of our solution is the creation of a large image collection from a structured sampling of camera positions, time steps and visualization operators. One option for managing these rendering is to compress them into a collection of movies. Chen et al [12] use this approach to accelerate interactive scientific visualization over the Internet. Kageyama and Yamada [13] applied this approach in situ to a simulation. Both create specialized movie players that support exploratory interactions accessing the linear movies to retrieve specific rotations, time steps and operators. Our solution extends these approaches by supporting the compositing of images to create new visualizations as well as metadata and image-based querying. Tikhonova et al in [14], [63], [15] represent the scientific data set to be visualized as a collection of proxy images. By retrieving images from this collection and applying image-based rendering techniques, interactive volume rendered results are produced. A range of view points, transfer functions, rendering, and lighting options are reproduced interactively. Our work is complementary to this image-based approach. Combining the approaches would support additional data compression, flexibility, and exploration possibilities.

Metadata Searching. Commercial multimedia databases such as Googles image search have brought the powerful non-traditional search/query techniques to the mainstream. Recent work by Subrahmaniam [16] identifies issues and future research directions for multimedia databases. For speed and flexibility in responding to our unique access patterns, we created our own image database. When populating an image database it is desirable to gather metadata about when and how the images are created, to enable querying of these parameters. Therefore, when we create imagery in situ, we save camera positions, time steps, details about the visualization operators, and statistics about the data. We highlight a connection to provenance systems, such as VisTrails, that directly store analysis results and how they are created. This is important so that results can be reproduced by others, improving scientific integrity [17]. Our approach could be evaluated as a visualization and analysis storage representation in such a system.

Creation of New Visualizations and Content Querying. Our approach supports the creation of new visualization by combining images with depth information. Tikhonova et al [14] also supports the creation of new visualizations from their database using interpolation. It is worth noting, when compositing images that contain opaque geometry, our results will be pixel accurate whereas with image-based interpolation, some loss is expected especially as the viewer moves away from sampling locations available in the database. Our idea for compositing visualization results evolved from the long history of

parallel compositing techniques that enable scalable interactive visualization. Moreland et al provides a recent overview of approaches [18] and apply additional optimization to this critical technique.

We are interested in supporting a variety of methods for interacting with our visualization. From an in situ starting point, we offer a traditional point-and-click interaction method, supporting the rotations, time steps, and visualization operations selected by the scientist in the setup of his/her simulation run. We also support interaction via an interactive database perspective. As mentioned above we support metadata searches. There are many image content search possibilities as well as search by color and search by similarly. In this paper, we focused on including image content queries that support querying about the visual weight of the objects in the generated visualization. This is a unique capability for an interactive scientific visualization framework and derives from the fact that we stored the visualization metadata, visualization objects, and their resulting 2D projections as images. There are many approaches to calculating the statistics of the 2D projection of a set of 3D objects [19]. Related to our approach, Jun Tao et al [20] computes a collection of streamline images and applies an image quality metric to select an optimal viewpoint. Our approach extends this work by virtue of being in situ and by our ability to change our evaluation metric dynamically with a scientist-generated query.

Complementary work would support indices on the original scientific data. These are referred to as data content-based queries and an overview is provided in [21]. An exemplar within this literature is Stockinger et al [22], which used efficient bitmap indices to support querying against the original data. Thus, we could save bitmaps with our visualization offering both image-based and data-based queries.

## 5.2 Approach

We present a novel framework  implemented in existing open-source tools [68] - to be used by a scientist to define a set of operations he/she finds to be most useful in exploring their data. The framework implements an image-based approach that results in a database of highly compressed data that is fundamentally different from what is currently available. Importantly, our framework effectively preserves the ability to interactively explore the same operation space defined at the start of the problem, so that data elements can be combined in much the same way they could in the original tool [69]. Thus, interactive exploration - so important to scientific discovery - is supported on a useful spectrum of operations.

Imagery is on the order of 106 in size, whereas extreme scale simulation data is on the order of  1015 in size. As an example, suppose we have an extreme scale simulation that calculates temperature and density over 1000 of time steps. For both variables, a scientist would like to visualize 10 isocontour values and X, Y, and Z cut planes for 10 locations in each dimension. One hundred different camera positions are also selected, in a hemisphere above the data set pointing towards the data set. We will run the in situ image acquisition for every time step. These parameters will produce: 2 variables 1000 time steps  (10 isocontour values + 3  10 cut planes)  100 camera positions  3 images (depth, float, and lighting) = 2.4  107 images. If we assume each image is 1MB (megapixel, four byte image), this results in approximately 24 TBs, which is a reasonable size for a large exascale simulation.

Thus, the image-based approach reduces the simulation output by storing a set of

output images directly from the simulation into an image database. One can think of this approach as the traditional in situ mode, but we are sampling the visualization and analysis parameter space, such as camera positions, operations, parameters to operations, etc., to produce a set of images [63, 70–72] stored in a data-intensive database. It is important to note these images are derived from full-resolution data with high accuracy.

The framework, implementing our image-based approach as a solution for extreme data visualization and analysis challenges, makes several contributions to the traditional in situ mode.

Interactive Exploration Database. Our image-based approach takes traditional in situ visualization and analysis and enables interactive exploration using an image database. This, in turn, creates a viable solution for extreme scale visualization and analysis. Our framework: Enables many different interaction modes including: 1) animation and selection for objects, 2) camera and 3) time, than we imagined possible with a set of pregenerated analysis. Creates an incredibly responsive interactive solution, rivaling modern post-processing approaches, based on producing constant time retrieval and assembly of visualization objects from the image database. Encourages the use of both computationally intensive analysis and temporal exploration typically avoided in post-processing approaches. Demonstrates the time to create an image collection is not of great concern.

Metadata Searching. By leveraging an image database, our image-based approach allows the analyst to execute metadata queries or browse analysis results to produce a prioritized sequence of matching results.

Creation of New Visualizations and Content Querying. Weve added composing of individually imaged visualization objects to our image-based approach to allow the analyst to reason about his/her simulation results from visualization space and create new content. This unique capability: Provides access to the underlying data to enable advanced rendering during post-processing (e.g. new lookup tables, lighting, ...). Makes it possible to perform queries that search on the content of the image in the database. Therefore, using image-based visual queries, the analyst can ask simple scientific questions and get the expected results. These image-based queries show promise of answering much more complicated questions.

Finally, we have exposed the framework of our imagebased approach to the scientist through an advanced selection interface that allows him/her to make sophisticated (time, storage, analysis, ...) decisions for the production of in situ visualization and analysis output.

In the sections that follow, we illustrate how our imagebased approach to extreme scale in situ visualization and analysis meets our goals for future post-processing exploration.

### 5.2.1 Overview of Approach

Though we are running into a bandwidth barrier, interactive post-processing visualization and analysis is still essential at extreme scale. When creating new simulations, scientists analyze support for exploration and debugging. Also, they need the capability to share data with colleagues that do not have access to their computing resources. Finally, there needs to be an analysis transition path for existing codes from terascale/petascale (1012/1015 FLOPS) to extreme scale.

In the image-based approach, we produce a multitude of analysis outputs, such as images and plots, that will show scientists about their data through interactions with an image database. Visualization and analysis operators typically produce results that

are many orders of magnitude smaller than the original data. Specifically, imagery is on the order of 106 in size, whereas extreme scale simulation data is on the order of 1015 in size. Our expectation is that the memory size, with associated burst buffers [23] and storage, will be on the order of 1015 (i.e., a petabyte). We believe a petabyte is a reasonable size for an extreme scale simulation output and, therefore, this means we can store approximately 109 (a billion) images of the simulation. This is on the order of the number of images uploaded to Facebook per year or the total number of photos hosted by Flickr [24]. The benefit of being able to store this many images is providing flexibility, similar to a data approach, for exploratory simulation analysis.

From our example present in the Introduction section, we would produce 2.4 107 images at 1MB per image for approximately 24 TBs, which is reasonable given our previous assumptions. We expect that massive computing power will be available on the supercomputer with associated burst buffer [23] and data intensive storage systems [25] to process these images.

## 5.3 Results

One interesting property of this approach is that the relatively fixed size of the output imagery, due to limits of human visual acuity [26], means that as we continue to scale simulation and supercomputer sizes we will be able to store more and more imagery as machine sizes grow. While the image size might grow as the simulation size grows, the analyst is less likely to increase the sampling of the corresponding parameters, operators, and camera space. This approach supports many different potential interaction modes, and may offer different insights than interacting with simulation data with traditional analysis and visualization tools. There are a number of useful image-based rendering approaches that may help us to sample and present generated images [63, 69]. The goal of this work has not been to produce better image-based rendering techniques. Rather, the goal is to understand how in situ methods are able to support flexible and accurate analysis of extreme scale data sets. In particular, we have focused on potential interaction modes with image data, but not necessarily those enabled by image-based rendering techniques.

Many scientific simulation communities produce image collections for later analysis and archival purposes. One of the best examples, the CESM (Community Earth System Model) [27], includes the diagnostics for all its component models including atmosphere, ocean, land, and sea ice. These visualizations are available on a webpage for analysis and archival purposes. They are generated in a post-processing manner after the simulations have completed. This collection represents a community consensus on a set of visualizations that are useful to the community. Other examples include astronomy [28], cosmology [29], and high-energy physics [30]. We recognize the current limitations of these collections. Specifically, most of these collections contain fully-rendered images that make it difficult to retrieve the original simulation data values. Optimizations to our approach will help to correct these shortcomings.

A. Simulation Data to Image Database Our image-based approach framework is built on top of ParaView, a modern visualization and analysis tool used around the world in post-processing for advanced modeling and simulation work flows. ParaView is an open-source, multiplatform data analysis application [31] developed to analyze extremely large data sets using distributed memory computing resources. Most modern post-processing

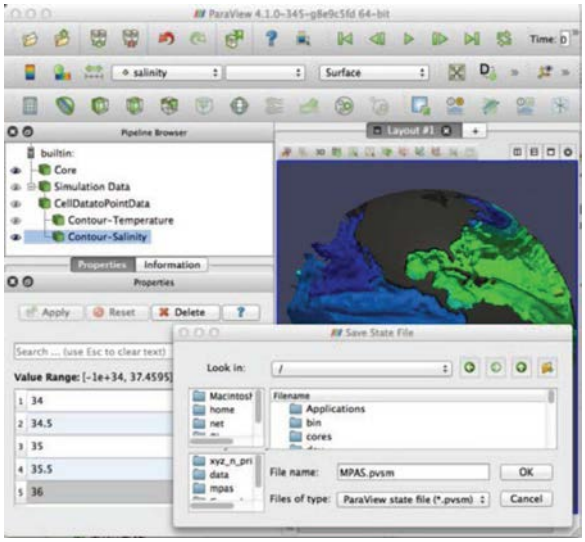



Figure 5.1: Once the computational scientist has defined a reasonable set of visualization and analysis operators in a familiar post-processing tool, he/she will simply save the current state of the application to a state file.

applications utilize a common pipeline architecture (e.g VisiT, ...). Thus, any of these tools could be easily adapted for use with our imagebased approach framework.

When starting an image-based analysis, the computational scientist will define a desired set of visualization and analysis operators using a test data set and a familiar post-processing application, as depicted in Figure 1.

Next, the scientist uses our advanced selection interface, shown in Figure 2, to make sophisticated prioritized decisions for the production of analysis output. By importing the state file, the advanced selection interface presents the visualization pipelines created previously using a familiar post-processing tool. Using the Pipeline section, the scientist determines: how often to perform in situ analysis, what visualization and analysis objects to create, and how to sample the visualization object parameter space.

Then, the scientist moves to the Camera settings section and describes how he/she will sample the camera space by defining a camera manager and making appropriate selections for  and  sampling. The tool instantly updates the total number of resulting viewpoints from the sampling selections. As simulations progress toward extreme scale, scientists must operate in a constrained storage environment. Hence, selection in the parameter space and sampling in the camera space will require prioritization to fit within a storage budget.

The final decision in the Image settings section particularly impacts storage. The scientist would select the image sampling (resolution) and the image type. The image types include: raw data (TIFF format) and compressed lossless data (PNG format).

The results of these choices are constantly updated in the Cost estimate section. The costs are reported for number of images, image size, collection size, and additional computational time.

The advanced selection interface could be very complex with many variables and operations from which to choose. The scientists selection of different outputs from the simulation are then presented (Figure 4), depicting the managed tradeoffs between additional computation, storage space usage, and visualization and analysis outputs. This selection could be optimized with intelligent selection capabilities, such as automatic iso-

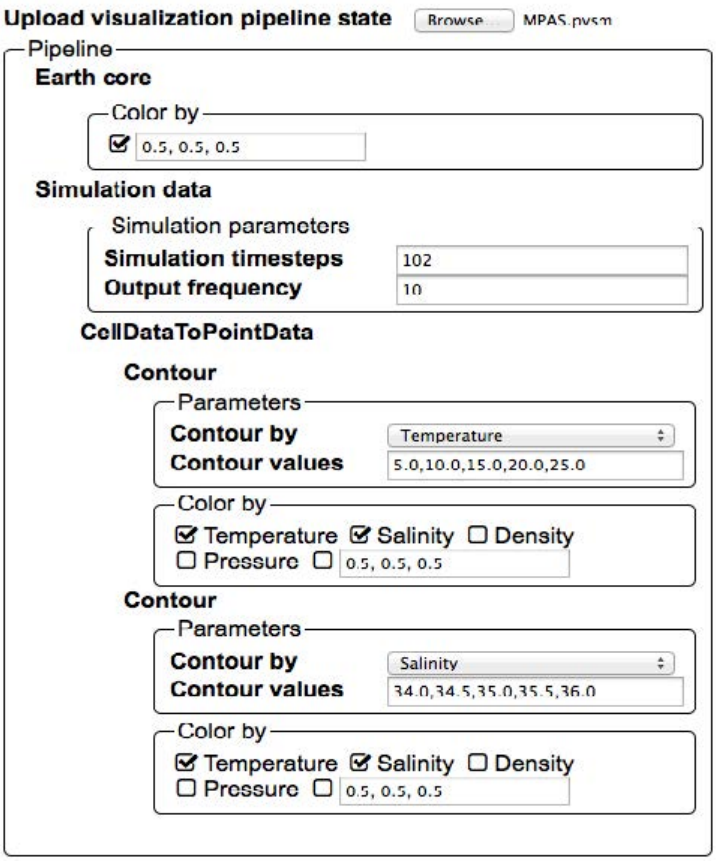



Figure 5.2: The advanced selection interface enables the scientist to adjust visualization and analysis operators and how to sample the parameters space.


Figure 5.3: By selecting camera space sampling, the scientists will receive instant viewpoint feedback in the Camera settings section.

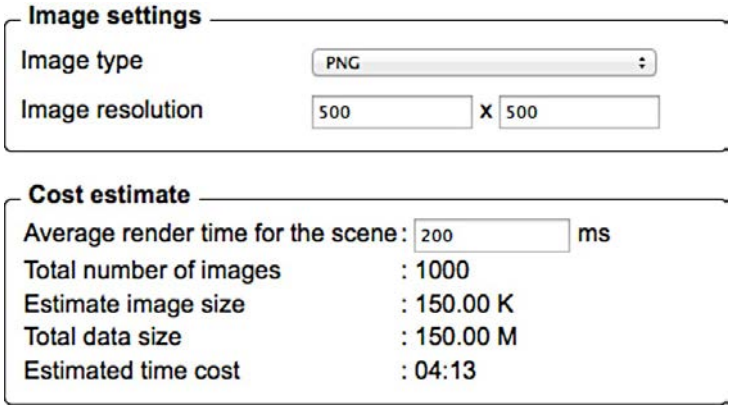



Figure 5.4: The Cost estimate section of the advanced selection interface enables the scientist to examine what if scenarios for costs in a constrained storage environment.

contour selection [32] and automatic camera selection [71], to help reduce the interface complexity.

The output of the advanced selection interface is an in situ analysis python script that implements the defined selections. Our framework uses ParaView Catalyst, an open-source in situ (and other use cases) visualization and analysis optimized C++ designed to be tightly coupled with simulation codes. As the simulation runs, the image results are ingested by the database. By this we mean, that the metadata, image provenance (i.e. a searchable description of how the image was created the simulation, the input deck, and which operators were applied), and the root image uniform resource locator (URL) avoiding the need to move the potentially large image data around.

Several of the contributions of our image-based approach are demonstrated using the Model for Prediction Across Scales-Ocean (MPAS-Ocean) [33] as an example. MPASOcean is an unstructured-mesh ocean model capable of using enhanced horizontal resolution in selected regions of the ocean domain. MPAS-Ocean is one component within the MPAS framework of climate models that is developed in cooperation between Los Alamos National Laboratory (LANL) and the National Center for Atmospheric Research (NCAR). The example runs are from real-world simulations with realistic topography, wind forcing, and temperature and salinity restoring. The horizontal grids are quasi-uniform over the globe, with simulations performed at nominal grid cell widths of 120 km.

B. Interactive Exploration Database The interactive exploration database enables a diverse set of interactions with a set (database) of pre-generated visualization and analysis results. The interactive exploration database supports essentially three elements and two modes of interaction, depicted in Figure 5. The three elements of interaction are time, (visualization and analysis) objects, and camera. The modes of interaction are: animation, where the interaction sequence through time, objects, and camera; and selection, where the analyst would select time, objects, and camera.

We have developed an interface for an interactive exploration database that supports the three elements and the two modes of interaction. From the top row of Figure 5, the analyst can select a time, an object (e.g., a three-dimension isocontour), and a camera, which requests the corresponding image to be fetched from the database. In fact, for each image in the top row a set describing time, object, and camera would be specified in the request. The bottom row of Figure 5 demonstrates animation of time, object, and camera, respectively, from the initial selection of the top row. We assume the sampling

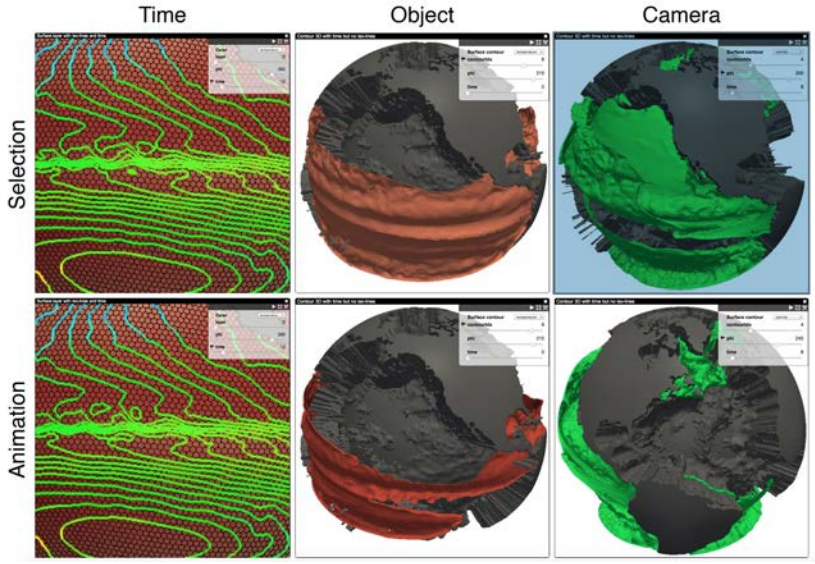



Figure 5.5: The interactive space enabled by the interactive exploration database.

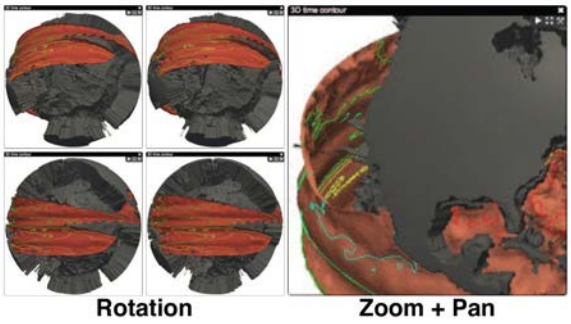



Figure 5.6: Mouse enabled interactive exploration such as rotation requires an image fetch from the database, while zooming and panning are image-based operations.

of the parameter space and camera space is dense enough so that interaction can be achieved on the three elements.

For the rotation images in Figure 6, we can see that the mouse-enabled rotation requires an image to be fetched from the database. Starting at the lower right image, if the analyst rotates up with the mouse, then the camera position change is queried and the upper right image is returned (likewise for rotations to the left and/or up and left). Mouse-enabled panning and zooming are simple image-based operations. Zooming displays the image by varying the pixel size to 11, 22, ..., n n, for zooming in, and removing necessary (row, column) pairs for zooming out. Panning simply shifts the image side to side or up and down. It is important to note that all querybased interactions are still valid and respect these image-based interactions. This is at the discretion of the scientist.

The image-based approach provides interactive (12+ fps) response from the interactive exploration database on typically available scientific network throughputs/bandwidths. In addition, adding more flexibility to the interactive exploration database only requires saving more images, which in turn more densely samples the parameter space and camera space.

One interesting advantage of this approach over the traditional interactive post-processing approach is that for the image-based approach, the time to display one image

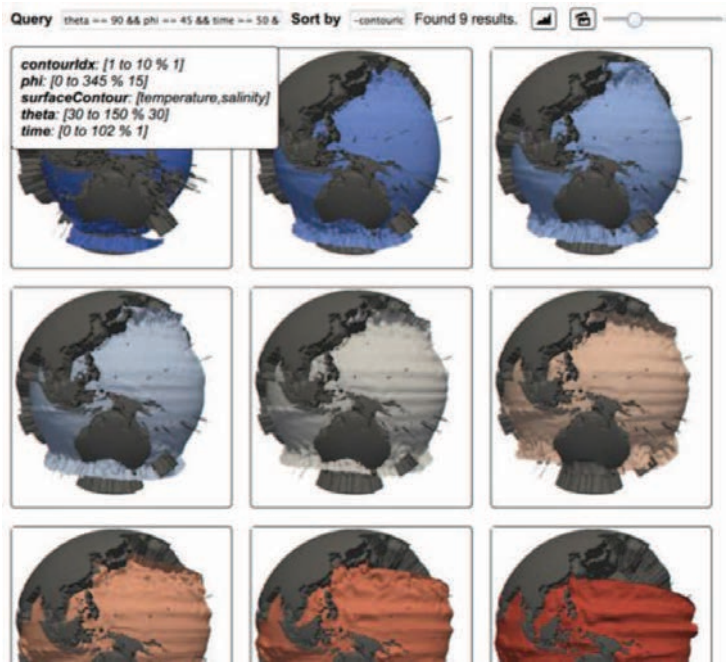



Figure 5.7: A simple example of a prioritized metadata query, where the temperature isocontours at time = 50 are ordered by increasing temperature isovalue for a particular camera.

is approximately the same time for any other image because the time to compute and render complex visualization and analysis objects has been amortized in situ within the simulation. For a traditional post-processing approach that computes visualization and analysis objects upon request, the wait time is extremely variable, ranging from seconds (rendering) to minutes (loading, pipeline selection, and computing). This inherent time to result bias produces a corresponding bias in what visualization and analysis objects are interactively explored. Specifically, because data sets are typically stored on disk as separate files for each time step, and the time to load a data set is typically long, very little interactive exploration in time is done. Our interactive exploration database addresses these issues, encouraging both computationally intensive visualization and analysis objects and temporal exploration typically avoided in post-processing approaches.

Although there may be a concern about the time it will take to create this image collection, we believe that this is a manageable issue since: (1) We show, in the Results and Performance section, that the creation of images is constant, O(1), in time as the problem size grows; (2) we have demonstrated in the past that in situ data visualization and analysis weakly scales; and (3) we expect that visualization and analysis operators will be accelerated to run on next-generation hardware at tens of frames per second [34].

C. Metadata Searching By leveraging the interactive exploration database, our image-based approach allows the analyst to execute metadata queries (or browse analysis objects) to produce a prioritized sequence of matching results. The metadata, produced by the in situ analysis python script includes data properties of the simulation data, such as histograms, as well as image properties.

In Figure 7, the desired query can be represented by leveraging the keywords in the camera metadata, theta == 90 phi == 45 the time metadata, time == 50 and the visualization parameter metadata, surfaceContour == temperature. These results would normally be presented in a sequential accessed, but unordered, manner with respect

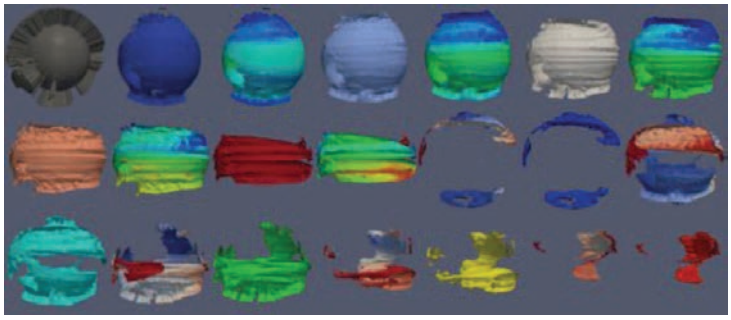


Figure 5.8: For visualization object compositing, the image is replaced by an image set consisting of an image sprite file, a composite file, and a query file. This image sprite contains all the example MPAS-O images (22 images) except the background.

to the query. If the analyst would like to sort by increasing isocontour index, then the isocontourIdx equation would present the results as desired. This is akin to the prioritized results returned from a Google search.

D. Creation of New Visualizations and Content Querying A core contribution of this work is the way in which new visualizations and queries are supported by the interactive exploration database. We utilize real-time compositing to create an experience similar to interactively exploring the simulation data itself, with significant additional capabilities only possible because of the image-based approach.

Adding visualization and analysis object compositing to our image-based approach framework allows the analyst to reason about his/her simulation results from visualization space as opposed to the explorations offered from image space rendering and sampling [14], [35], [20]. With the addition of visualization object compositing, the interactive exploration database retains the three elements and the two modes of interaction described in Figure 5.

For compositing, instead of a single image, the imagebased approach framework creates an image sprite of the separate visualization objects to be interactively composited (see Figure 8).

The visualization objects compositing provides an approximation of exploratory interaction with a raw data set. We can automatically display multiple objects from visualization space by selecting the associated image set for the (time, objects, and camera) selection from the database and composing them together. But, we do not require the analyst to do this manually through a database query. Instead, the analyst uses an interactive tool that emulates applications like VisIt and ParaView to simulate the experience of exploring simulation data.

In Figure 9, the eyes indicate visualization and analysis objects that can be interactively turned on and off, the Background allows for changing the background object color, and the time and camera control selection and animation for these elements.

The 22 visualization objects in the image sprite of Figure 8 can be combined, as demonstrated in Figure 10, into n r=0 n! (n r)!r! = 2n = 4, 194, 304 unique images.

While, in this example, a large number of these items occlude one another, this demonstrates the magnitude of the data space spanned by this set of elements for interactive exploration. All of the new image possibilities are a result of the compositing, which utilizes the composite file that is a per pixel linked list object order encoding created by comparing the z-buffers for each object.

The interactive explorations database can be further extended through the output of

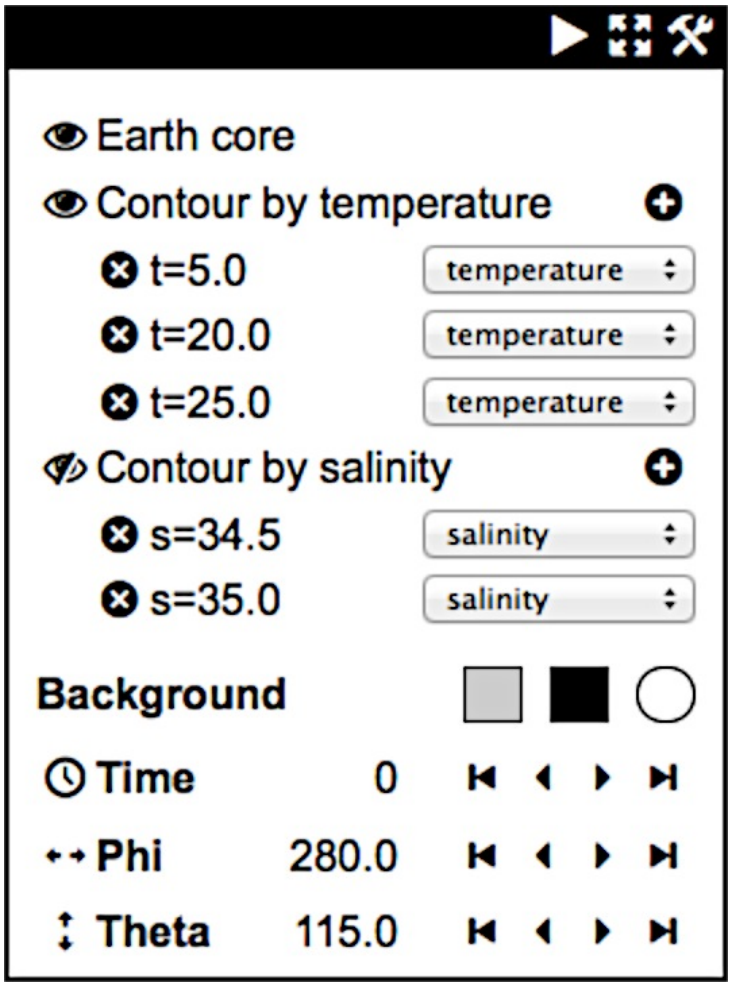



Figure 5.9: The user interface of the scientist defined visualization pipeline for visualization object compositing.

Figure 5.10: A subset of the possible images the analyst can interactively create from this one viewpoint in time.

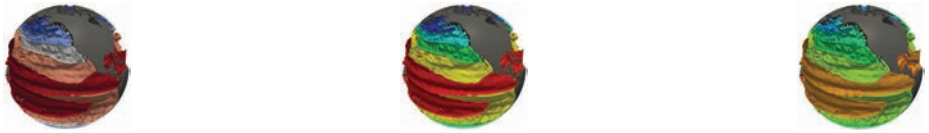


Figure 5.11: Using lighting and color mapping, render passes and compositing enable more capable visualization pipelines such as changing color scale mapping for objects.

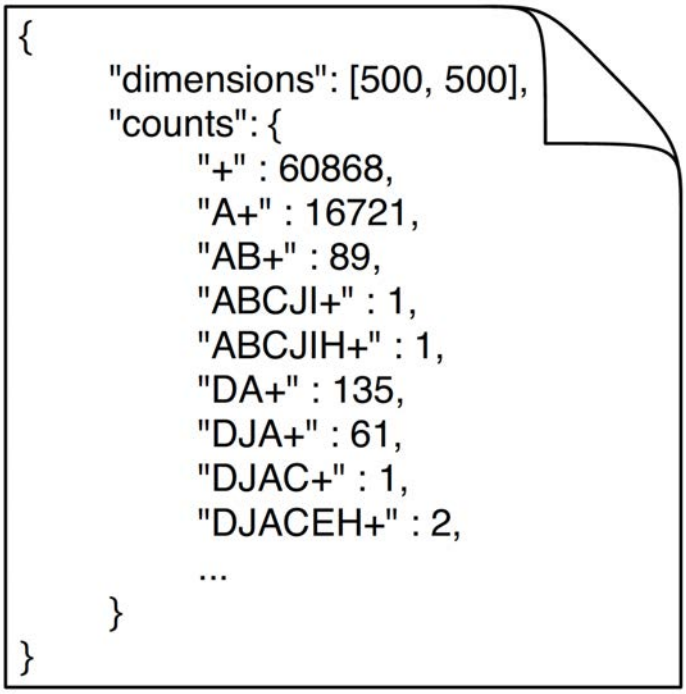



Figure 5.12: An example query file

image sets for a particular viewpoint consisting of the composite file, image sprite file, and possibly a lighting image and/or a raw floating point image, i.e., the simulation data values. For volume images, there has been recent work for creating images with changeable transfer functions [63]. For our opaque image sets, if we also save the simulation data associated with the visualization and analysis objects, then more capable visualization pipelines, such as the one presented in Figure 11, are possible using a number of rendering passes, including the lighting and color map passes.

The visualization objects compositing infrastructure makes it easier to perform queries that search on the content of the image in the database [36]. For example, a query could be formulated that matches on the quality of the view of a particular isocontour value [71, 72].

The query file contains statistics on object mapping pixel coverage. For example, in Figure 12, if we want the pixel coverage of both object A and object D in a resulting image, then we would add up the pixel counts for all object order mappings where A proceeds D, or D does not exist, to get 16812 pixels with object A. We would also add up the pixel counts for all object order mappings where D proceeds A, or A does not exist, to get 199 pixels with object D. Note that the + symbol is use to indicate the background object, ends each combination in the query file.

The query histogram, shown in Figure 13, can be used to determine the reasonableness

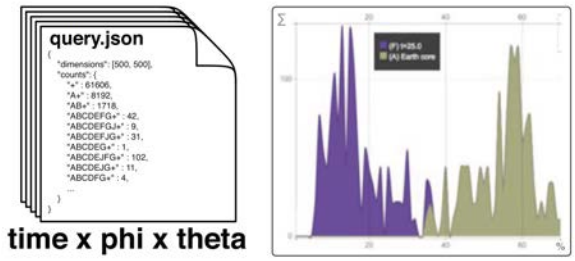



Figure 5.13: The query files are examined to produce a histogram that depicts the percentage of coverage (x-axis) by the count (y-axis) of possible resulting images with that pixel coverage for each object independently.

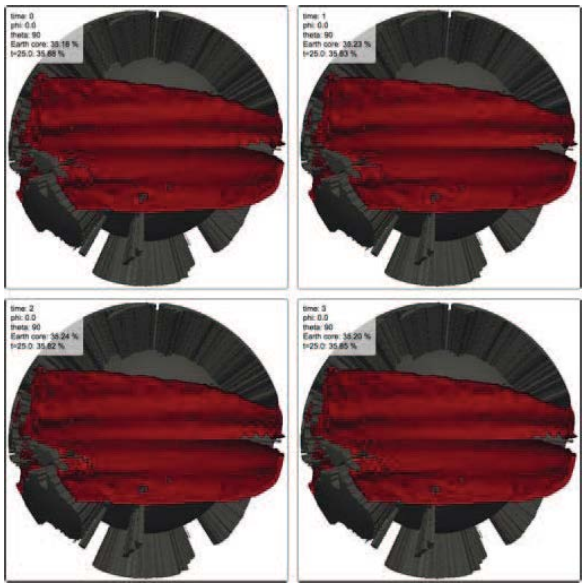


Figure 5.14: Building an example science query and sorting algorithm by leveraging the query pipeline histogram.

of proposed queries. In Figure 14, we want to perform a science-based visual query to determine the images (location on earth) with the largest isocontour representing a temperature of 25 C (warmest ocean) for the first four time steps. From the query histogram, we see that F represents the desired isocontour object and that the pixel coverage percentage achieves a maximum somewhere above 35.535.5) (time ¡ 4). These results would normally be presented in a sequential accessed, but unordered, manner with respect to the query. If we would like to sort by the maximum pixel coverage biased by the increasing time step, then the F - time equation would present the results as requested.

Both Figures 14 and 15 demonstrate that we can ask simple questions and get the expected results. These image-based queries, however, show promise of answering much more complicated questions.

A second example used to test and demonstrate our imagebased approach involves the xRage code, developed by LANL, which is a one-, two-,and three-dimensional, multi-material Eulerian hydrodynamics code for use in solving a variety of high deformation flow of materials problems. Examples present simulation results of the asteroid impact that created the Chicxulub crater in Mexicos Yucatan Peninsula [37].

In Figure 16, the scientist is looking for the best view of: the deep ground material

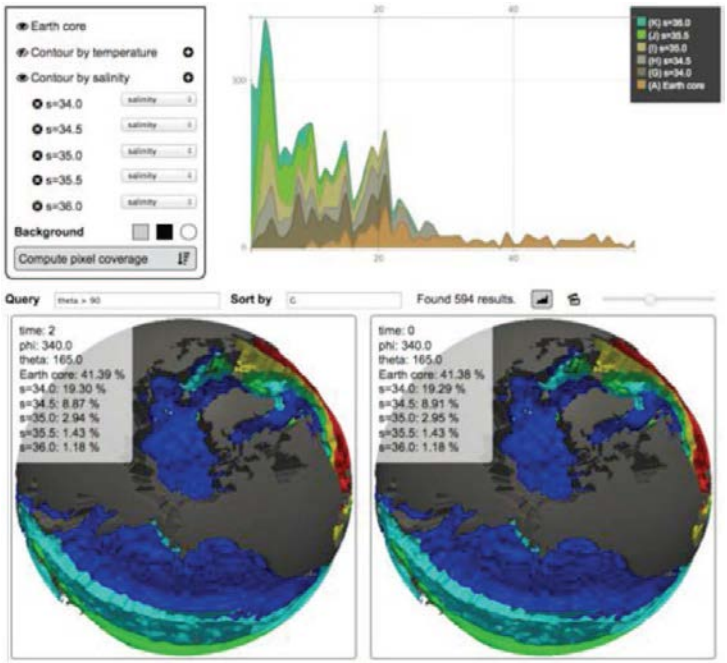



Figure 5.15: Image-based query of a simple scientific question: Where is the largest visible mass of low salinity in the northern hemisphere?

threshold (D); the isocontour of the pressure wave in the ground material (F); slice colored by the velocity magnitude (H); and the threshold of asteroid material rooster tail (E). In this case, we can sort the entire data set by the equation E+F+H. The method uses visible pixel count as a metric for a hit, and then returns a possible matches priority-sorted list. In this case, the query does return logical results showing the front and back views perpendicular to the slice at later time steps. The query results can be toggled between query statistics and the resulting images.

IV. RESULTS AND PERFORMANCE The traditional GUI-based post-processing work flow suffers extremely variable wait times based on algorithm differences in loading, interactive pipeline selection, computing, and rendering. The problem quickly becomes intractable with large-scale data sets that require parallel resources. Interactive post-processing gives way to work flows requiring the scientist to write batch scripts and execute a second independent HPC work flow. In practice, this has an enormous impact on what visualizations and analysis methods are explored.

For our image-based approach, the time to display any image is approximately the same for any other image because the image generation time has been amortized in the single simulation work flow. The constant retrieval time (or the time to fetch an image from the database) of our image-based approach enables interactive exploration, and embedding the image capture in situ removes the additional HPC work flow. The remaining obstacle, for the scientist, of writing the in situ analysis script is greatly simplified by using our framework.

The cost of producing imagery for the interactive exploration database is roughly two times the cost of producing an equal number of in situ images (see Figure 17). Twice the cost is an astonishingly small price given that the interactive exploration database imagery, for example, generates 10 plus 1 background images in an image sprite to create $211 = 2048$ unique images through post-processing, compared to simply the 10 unique images.

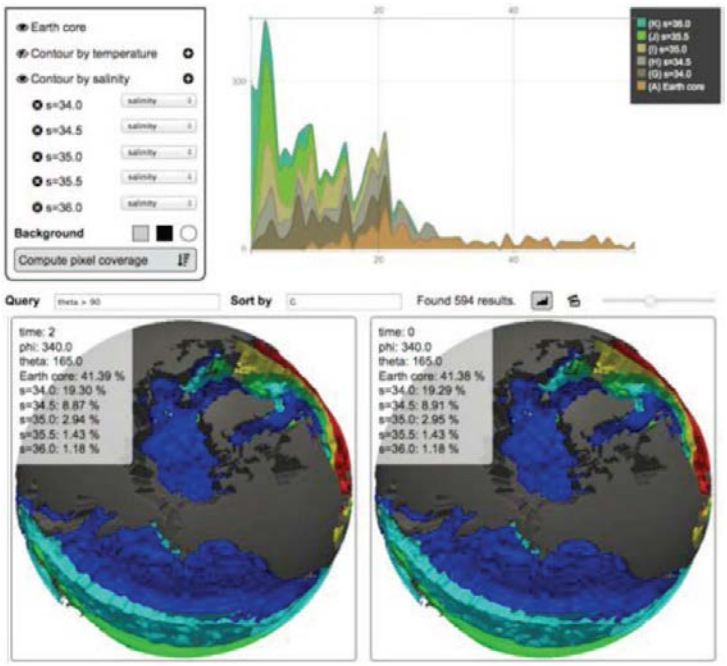



Figure 5.16: Queries based on the image content can be used to search for qualitative results like best view. The top three images show great views of the four items simultaneously. Later down in the search results are the bottom three images that obscure almost everything except the deep ground threshold.

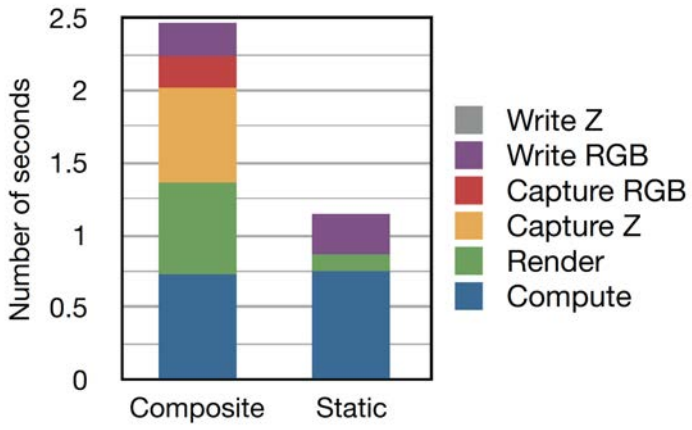



Figure 5.17: The cost to produce one viewpoint of imagery for the interactive exploration database versus the production of an equal number of in situ images.

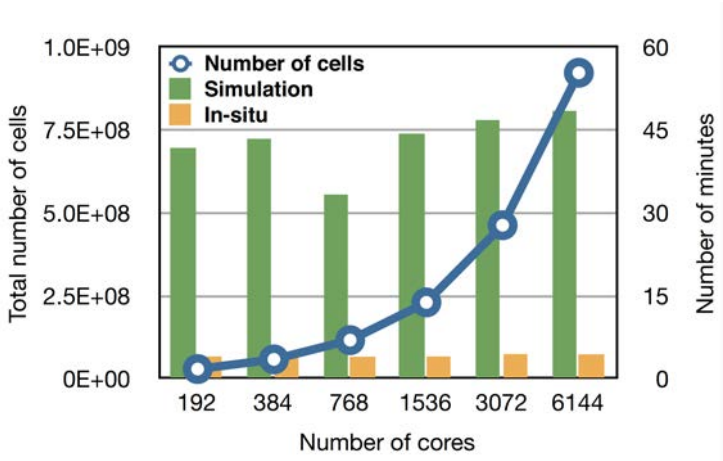



Figure 5.18: The weak scaling results of compositing imagery produced every ten time steps up to 6144 cores along with the growth in problem size (number of cells).

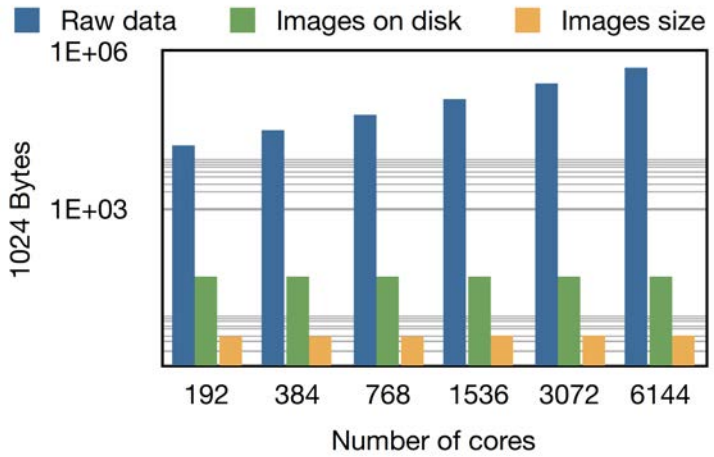



Figure 5.19: Disk usage reduction comparing full xRage data files versus disk space occupied on the Panasas disk is 52MB with large block size versus the actual size of the images is approximately 3.5MB.

One goal of in situ visualization and analysis is to reduce the time it takes a scientist to gain insight into the problem being simulated. From past studies, we see that ParaView Catalyst performs and scales well [38]. In a detailed study [39] led by the visualization group at Sandia National Laboratory, the in situ analysis showed weak scaling up to 64k cores on a variety of simulation codes. A second study comparing in transit and in situ analysis work flows [40], led by the same group, demonstrates the overall computational time (simulation + in situ analysis) scales for Sandias CTH simulation code for various problem sizes and process counts. It rivals in transit approaches as the simulation size grows.

As the problem size increases and the number of processes increase, the benefits of using in situ analysis become more apparent [65].

For our performance results, we analyze the xRage code simulation results of the asteroid impact, mentioned previously. These simulations were the subject of a detailed study [41], performed by the LANL Data Science at Scale group, on in situ analysis image production. Their weak scaling study fixed the maximum number of cells at roughly 150K per core for AMR xRage runs. The study demonstrates that, as the problem sizes continue to grow, the image production of simple, single visualization and analysis objects remains constant.

Our weak scaling study examined the same xRage simulations from [41] using our approach to produce 10 different isocontour objects plus a background object at an image size of 500x500. Figure 18 shows the results of our weak scaling study (under normal cluster operation load) that demonstrates that the production of interactive exploration imagery remains constant.

Figure 19 demonstrates that significant data reduction can occur at relatively small core counts. It also demonstrates an interesting issue related to the size of the imagery as it appears on disk. Typically, extreme scale storage systems are tuned for large files. The image-based approach might benefit from a significantly different tuning of these file systems.

## 5.4 Core References

James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press

John M Patchett, James P Ahrens, Boonthanome Nouanesengsy, Patricia K Fasel, Patrick W Oleary, Christopher Meyer Sewell, Jonathan L Woodring, Christopher J Mitchell, Li-Ta Lo, Kary L Myers, et al. Lanl csse l2: Case study of in situ data analysis in asc integrated codes. *Technical report LA-UR-13-26599*, 2013

J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012

# Chapter 6

# Analysis Driven Refinement

**Abstract**

Prioritization of data is necessary for managing large-scale scientific data, as the scale of the data implies that there are only enough resources available to process a limited subset of the data. For example, data prioritization is used during *in situ* triage to scale with bandwidth bottlenecks, and used during focus+context visualization to save time during analysis by guiding the user to important information. In this chapter, *ADR* visualization is presented, a generalized analysis framework for ranking large-scale data using Analysis-Driven Refinement (ADR), which is inspired by Adaptive Mesh Refinement (AMR). A large-scale data set is partitioned in space, time, and variable, using user-defined importance measurements for prioritization. This process creates a *prioritization tree* over the data set. Using this tree, selection methods can generate sparse data products for analysis, such as focus+context visualizations or sparse data sets.

## 6.1 Related Work and State Of the Art

Large-scale data implies that there is insufficient computational or human resources to fully process all of the data. If all of the data will not be processed, decisions must be made as to which data will be processed, which data will be looked at first, and which data will be culled or ignored. For example, in time-evolving, large-scale simulations, a common way to save I/O bandwidth is to store time step data at periodic intervals. This is an explicit culling of the time series, deciding *a priori* which parts of the data are important enough to use in later analysis. Likewise, a focus+context visualization, like automatic camera guidance, is able to focus the user's attention, saving them time. The viewpoint is positioned toward more important data, while less important data might never be looked at.

The common thread in both of these examples of data analysis work flows is that data elements are ranked with respect to each other, and some data is identified as more important and thus chosen for resource access while others are not. Analysis-Driven Refinement (*ADR*) visualization is a generalized framework for prioritization of large-scale data for applications in analysis in resource-constrained environments. Our method prioritizes a data set by creating a *prioritization tree*, which is a relative ranking of the data by recursive partitioning, as seen in Figure 6.1. The first key concept is that ADR creates an analysis-driven Adaptive Mesh Refinement-like (AMR) grid, using user-defined importance measurements for data partitioning. This analysis grid spans space, time, and variables, and is independent of the source data set's original grid.

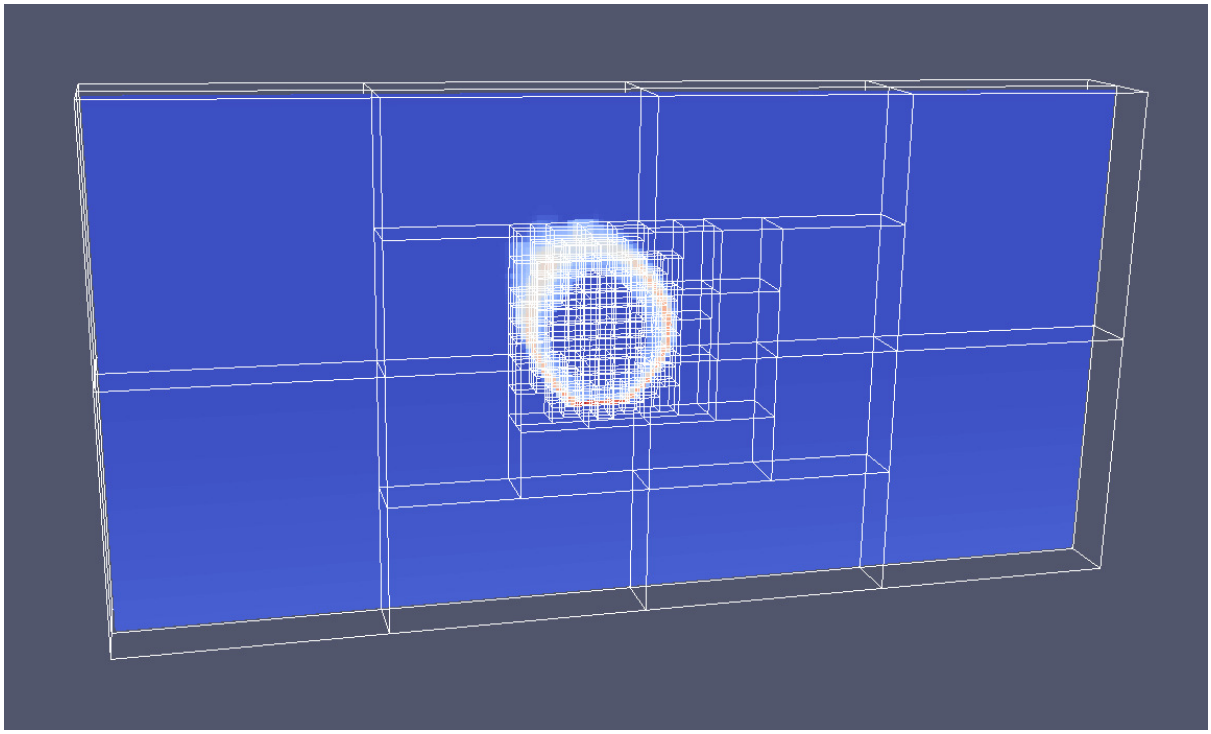


Figure 6.1: ADR, Analysis-Driven Refinement, using the maximum value of a partition as the importance measurement for a 3D asteroid simulation. A partition is refined if the maximum value meets an importance criteria. Data priority, or importance, is the refinement depth of the data in the priority tree.

The second key concept in ADR is the ability for end-users to define custom measurements for partitioning the tree. User-specified measurements define what is important in a data set and determines whether recursive partitioning should continue. Examples of measurements include minmax, standard deviation, entropy, or value range to determine if data is important enough to be partitioned further. Importance measurements can also be domain specific, such as counting the number of eddies in an ocean simulation partition, or a halo census in a cosmology data partition.

The third key concept in ADR is that the resulting prioritization aids selection algorithms in generating sparse data products for explicit data budgets, such as *in situ* data triage and focus+context visualizations. This prioritization system specifically improves upon previous streaming architecture research [74], by describing a generalized framework for prioritizing data. Thus, our contribution is a framework for prioritizing large-scale scientific data by: 1) an AMR-like partitioning over a data set, independent of its grid, which 2) utilizes user-defined measurements during partitioning, and 3) is applicable to many different triage, analysis, and focus+context visualizations that have data budget constraints.

In ADR visualization, the priority tree creates a high-dimensional refinement grid, which is very similar in appearance to an Adaptive Mesh Refinement (AMR) grid. AMR is a technique for improving the runtime of scientific simulations by changing the mesh cell size and computation requirements for parts of the computational domain. AMR has been shown to be essential in a wide variety of scientific fields, including fluid dynamics [75], astrophysics [76] [77], cosmology [78] [79], computer graphics [80] and many others. Berger *et al.* [81] [82] introduced the approach of block-structured AMR, which

covers the computation domain with a hierarchical data structure of Cartesian grids and subgrids. Another popular approach for AMR is using finite-element models on unstructured meshes, an approached used by Lohner [83] for computational fluid dynamics. For scientific simulations which employ AMR, one possibility is to use the AMR grid generated by simulation for visualization and analysis purposes. One problem with that approach, for ADR, is the AMR grid represents the finest resolution required over all variables.

ADR narrows down subsets of data based on importance measurements, and the result is similar to feature detection and extraction. Silver and Zabusky [84] use a flood fill algorithm to extract features from a 3D volume. Post *et al.* [85] use a similar technique for feature extraction and then represents those features as icons. More recently, Tzeng and Ma [86] use machine learning techniques on the problem of feature extraction in order to reduce the need for manual intervention. The system learns to extract features with certain properties through limited user interactions. Ji and Shen [87] use earth mover's distance to find the globally best match for tracking features over time. A survey of feature extraction and tracking techniques for flow fields is given by Post *et al.* [88].

Data selection and triage are important techniques for large-scale data, which can drastically reduce the amount of data written to disk or transmitted over a network. Indeed, one primary goal of ADR is to provide a flexible prioritization framework for data triage. Tong *et al.* [89] developed a method to select salient time steps of a time-varying data set using dynamic time warping. Their method, though, finds a globally optimal solution and thus requires all time steps to be available at once. Because of this, their method is not suitable for *in situ* uses. Woodring *et al.* [90] utilize statistical sampling techniques to create a level-of-detail representation of a cosmology simulation. Modeling a time-varying data set as a 3D array with Time Activity Curves (TAC) is the approach used by Fang *et al.* [91] to locate user specified regions and time steps of interest. Biswas *et al.* [92] used mutual information to find groups of variables in a multivariate data set which had high information overlap.

Similar to our work, Wang *et al.* [93] [94] partitioned volume data in image space based on entropy to guide level-of-detail selection. Our framework is more general, as it is applied in data space, which allows many other types of measurements, and selection algorithms to be used. Another work by Wang *et al.* [95] calculated an importance curve for each block of a time-varying data set by using conditional entropy, and clustered the data using these importance curves.

Another goal of ADR visualization is focus+context visualization, to save scientist and analyst time. Viewpoint selection has been studied extensively for several different types of data. In 2005, Bordoloi and Shen [96] generated informative views for volume rendering by using a viewpoint goodness measure, which included using entropy and taking into account the transfer function. Viewpoints for vector fields were evaluated by Lee *et al.* [97] by calculating the entropy of a vector field, and using a maximal entropy projection framebuffer to obtain a view dependent measure. ADR can utilize several different measurements for automatic camera placement. Muehler *et al.* [98] produced viewpoints of anatomical structures from medical images. Several parameters, including occlusion and viewpoint stability were used in their method. Vazquez *et al.* [99] calculated viewpoint entropy by measuring the distribution of projected mesh polygons to find good cameras views for image-based rendering.

## 6.2 Approach

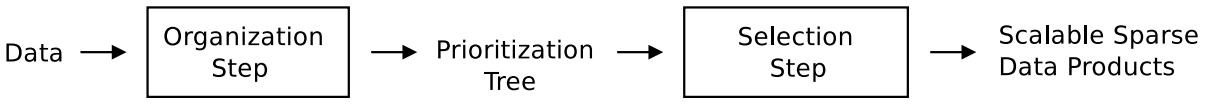



Figure 6.2: An overview of the ADR visualization framework.

Figure 6.2 displays an overview of the ADR visualization framework. The first step, and primary focus of this chapter, is the *organization* step, where input data is ranked by partitioning them into a prioritization tree. An example of a spatial partitioning can be seen in Figure 6.1. A large-scale data set is recursively refined, or split, into partitions. For each partition, an importance measurement is calculated, and the measurement is tested against *importance criteria*. If the measurement passes the importance criteria, the partition is split and each resulting child partition is recursed on. An example would be refining a data set based on maximum temperature as the importance measurement, and setting the importance criteria as a user-defined temperature threshold. For each partition, the maximum temperature is calculated, and then the partition is refined if the maximum is greater than the user threshold. The relative importance between the resulting data partitions is determined by the depth at which the partitions occur in the tree. Construction of the prioritization tree is described in Section 6.2.2.

The second step in ADR visualization, the *selection* step, uses the resulting prioritization information to generate data products, such as sparse data sets or focus+context visualizations. The selection step is discussed in Section 6.3.1. Examples of the selection step include using the prioritization tree to determine automatic camera placement, or choosing only the most important data partitions to write to disk.

### 6.2.1 Data Model

The primary focus for ADR visualization is to prioritize data generated by large-scale scientific simulations that are run on supercomputers. Therefore, the input data set to ADR visualization is assumed to be a time-varying, multivariate data set with spatial information. In particular, ADR is designed to be run *in situ*, so that data may be prioritized for triage operations that occur while the data is still in supercomputer memory. For sake of clarity, the abstract data model will be described in several different ways, which are shown graphically in Figure 6.3. The most familiar data model for a time-series simulation (with a set of discrete time steps, $t$) is spatial blocks over time ($t_i$ in $t$). The set of spatial data points, per time step $t_i$, have position ($x, y, z$ coordinates) and field values (a set of variables, $v$, per point).

Alternatively, spatio-temporal data set can be described as a table or matrix, with $c$ columns and $r$ rows. The $c$ columns describe the $x$, $y$, $z$, $t$, positions of all the data in space-time, along with their $v$ values per point (row), for a total of $|v| + 4$ columns. The total number of rows (points) $r$ is equal to $|t| * n$, where $n$ is the average number of data points per discrete time step in $t$. Finally, the data in the table can also be described by a set of $|t| * n$ points in $|v| + 4$ dimensional space, where the coordinates of a point $p$ is $(x_p, y_p, z_p, t_p, v_p^0 ... v_p^{|v|-1})$.

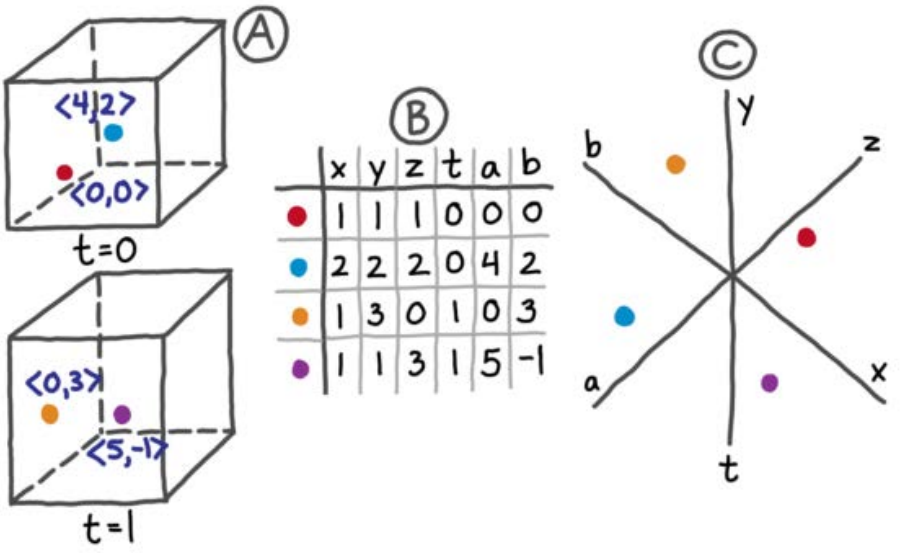



Figure 6.3: Different views for representing the same time-varying data set, where $a, b \in v$ are a set of variables at a position in space, $x$, $y$, and $z$, over time $t$: (A) "traditional" spatio-temporal view, (B) table (matrix) view, and (C) high-dimensional projection using star coordinates.

### 6.2.2 Prioritization Tree

A data set is recursively split into subpartitions, and Figure 6.4 shows an illustration of this applied to the three data models, which will result in a hierarchical tree representation. The core algorithm is a high-dimensional partitioning, similar to a kd-tree, where high-dimensional data points are recursively partitioned into hypercubes, inspired by stratification and latin hypercubes [90]. Our particular axis partitioning order is time groups ($t$ dimension) first, followed by variable groups ($v$ dimensions), and ending with space ($x$, $y$, and $z$ dimensions) groups. Divisions (partitions) are decided by user-defined, analysis-driven importance criteria, which decide whether partitioning should continue. This is similar to how a traditional spatial kd-tree will continue to spatially subdivide data until it reaches stopping criteria.

The partitioning algorithm is different from a simulation AMR-octree or a visualization kd-tree in two primary ways: 1) analysis-driven criteria to control partition refinement is used, and 2) the time and variable "dimensions" are partitioned, in addition to the spatial dimensions. Considering a recursive partitioning of a spatio-temporal data set, the root of the tree represents a cluster containing all of the data in a data set. Nodes further down the tree are data points contained within a space-time-variable partition. This new grid is created independently of the source data grid, which is based on refinement using the importance measurements. Different example measurements are discussed in Section 6.2.6. In general, our ADR construction forms a high-dimensional tree of data points, similar to kd-trees for k-NN search.

### 6.2.3 Top-Down Partitioning

In constructing the prioritization tree, two methods have been developed, a top-down and bottom-up approach. The top-down approach is easier to explain, while the bottom-up approach is more practical from an implementation point-of-view. For now, assume that

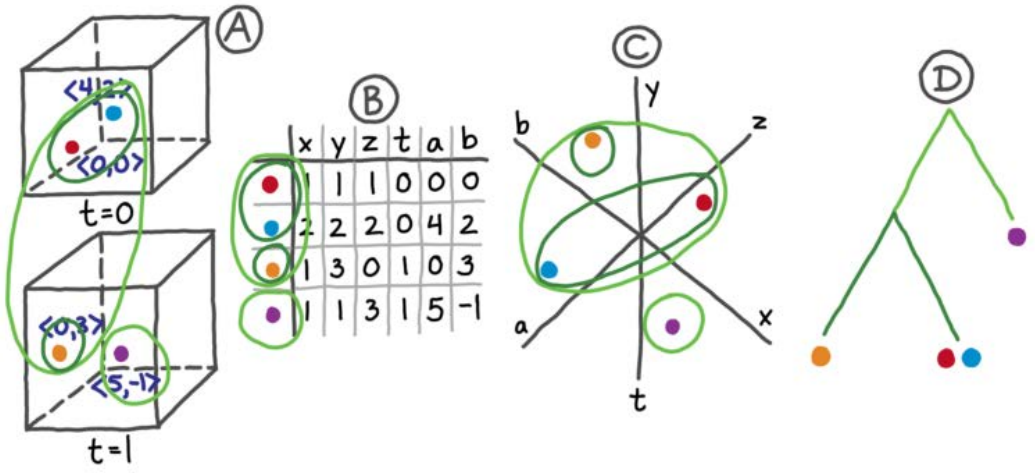
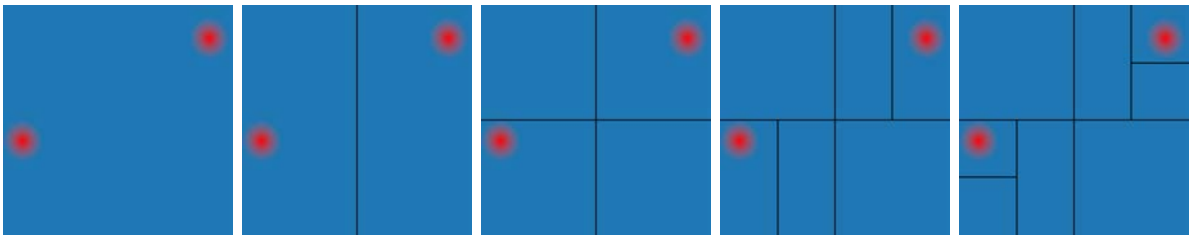



Figure 6.4: One hierarchical partitioning of the data set, shown in (A), (B), and (C), across the three data models. (D), the priority tree, is equivalent to this partitioning.



(a) (b) (c) (d) (e)

Figure 6.5: An illustration of the top-down algorithm for the spatial partitioning of a single time-variable partition. Low values are blue, and high values are red. The importance measurement is the maximum value of the partition. The importance criteria is whether the maximum value is close to the global maximum, within a user-defined percentage.

all of the data from a time-series, spatial, multivariate data set is available. In a top-down fashion, the priority tree is created by applying *time importance measures* to the data set, and partitioning it by time values, if the data passes the criteria measurements. Variables, and then space, are partitioned in a similar manner, using *variable importance measures* and *space importance measures*. Partitioning stops if the importance measurement does not pass the importance criteria. Figure 6.6 describes the steps involved in the top-down approach, while Figure 6.5 shows a concrete example of the spatial partitioning process.

The top-down algorithm will recursively partition a data set by the time dimension, until recursion cannot continue (i.e., either the smallest time partition size has been reached or the time importance criteria has failed). Then, it will partition the data set by variables, and then space, until partitioning cannot continue in those dimensions, either. The importance measurements and criteria, used in steps 4 and 5, need to be supplied by the user. They define how the data set is partitioned according to analysis-driven criteria, such that several time, variable, and space importance measurements and importance criteria can be used. For now, the algorithm is restricted to axis aligned splitting, such that hypercube partitions in $|v| + 4$ dimensional space are formed (i.e., ranges of values on a dimension are split and created). Note that any general type of

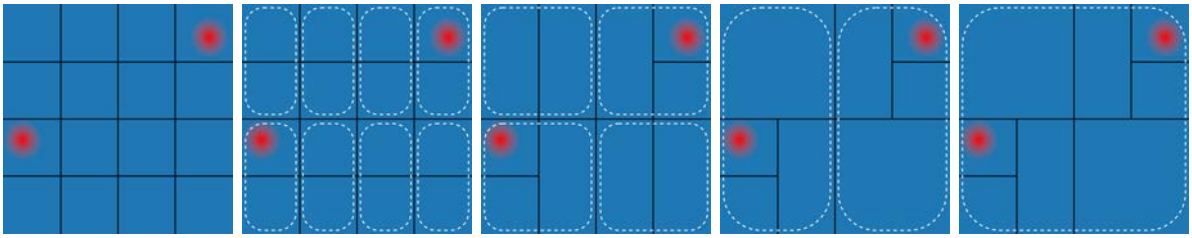
```
1: given: a data set of one partition
2: constraint: apply partitioning in sets of dimensions: time, variable, and space order
3: if partition is larger than smallest permissible for a dimension then
4:    calculate the importance measurement of the current partition
5:    if the measurement passes the importance criteria then
6:       split the partition into child partitions
7:       recurse on step 3 with each child partition
8:    end if
9: end if
```

Figure 6.6: The top-down approach for partitioning

split is acceptable in the ADR framework.

The top-down approach is easy to understand, but it has multiple drawbacks when trying to implement it in parallel. Typically for a large-scale simulation, the data for a particular discrete time step will be distributed over several processes. Therefore, to calculate the importance measurements *in situ*, importance values or the data itself need to be gathered from all of the processes in a partition. For example, calculating the importance value for a time partition requires gathering all data from all of the processes that the time steps are distributed over. Secondly in the top-down approach, the data set will be read and communicated multiple times (in the worst case, as many times as the maximum depth of the priority tree), as the importance measure has to be recalculated after every splitting step. Due to these drawbacks, an alternative, bottom-up construction was developed.

### 6.2.4 Bottom-Up Merging



(a) (b) (c) (d) (e)

Figure 6.7: An illustration of the bottom-up merging approach for the smallest spatial partitions of a single time-variable partition that is analogous to Figure 6.5. White dotted borders show parent partitions being created from merging child partitions. Child partitions only remain in the tree if the parent meets the importance criteria.

An alternative approach is to build the prioritization tree from the bottom-up through merging partitions together, which amounts to a result equivalent to the top-down approach. One requirement for this bottom-up algorithm is that all importance measurements must be order-independent reducible (i.e., associative). This is not a strict constraint for the top-down construction. Figure 6.8 describes the steps for the bottom-up

approach and Figure 6.7 shows an example of creating the spatial partitioning of the tree.

```
1: given: a data set split into time-space-variable partitions of the smallest permissible
2: for each partition, calculate importance measurements
3: constraint: apply merging in sets of dimensions: space, variable, and time order
4: while the tree is not complete do
5:    create a parent partition, merging valid child partitions
6:    reduce the child importance to create parent importance
7:    if parent importance meets the importance criteria then
8:       keep the children of the parent in the tree
9:    else
10:      prune the children, the parent is a now a leaf
11:   end if
12: end while
```

Figure 6.8: The bottom-up approach for partitioning

In this method, rather than recursively splitting, all of the data was pre-split into the smallest partitions permissible, and gather them into a priority tree. This is quite similar to a hierarchical, bottom-up clustering approach, where a data set is split into the smallest elements and then continually merged, until the tree is complete. To make our algorithm equivalent to the top-down splitting approach, the merging of two children is evaluated for a result that in a parent that would have split in the top down case (step 7). If not, the children are pruned from the tree.

The computational advantage, primarily for parallelism, is that the points in a data set are only read once to generate subsequent importance values. The importance value of a parent partition is calculated from the children through associative reduction of importance. For example, if the importance measurement is maximum temperature, then the measurement for a parent partition can be found by finding the maximum among the all importance measurements of children partitions. A top-down, splitting approach will have to constantly rebalance and communicate data importance values to processors that share data within an ADR partition, which significantly slows down the computation. Also, the bottom-up parallel communication happens in pairs, while the communication pattern in the top-down parallel approach will have scatter-to-many. As mentioned earlier, this does require that the importance measurements are associative for bottom-up merging, but this is rarely a serious constraint, particularly for our test cases.

### 6.2.5 Incremental Time Handling

Strictly speaking, time partitioning cannot happen as described in prior sections for the majority of time-varying simulations. This is because generally only one discrete time step is available *in situ*, due to time-evolving simulations and supercomputers having limited memory. Therefore, evaluating the importance of a discrete time step needs to be done in an out-of-core, best-effort, and/or heuristic manner. The splitting/merging step for the time dimension has to be done with this expectation in mind, such that the data from an entire time step may never be saved, due to the selection (triage) step.

Approaches taken use streaming, out-of-core, statistical calculations to determine time partitions. For example, time-series data models (not described in this chapter) using the current time step and past time steps. This is used to predict if the creation of a new time partition is needed, or if the current time step belongs to a previous partition. Another problem is that future time-series data (as well as past data, which may not be saved) is also not known at a particular instance in time. Therefore, a best-effort approach to predict if the current time step will be important, relative to future data, using projections of past statistical time-series data is used. Time partitioning is difficult, and likely an unsolvable problem in the context of time-series simulations. Despite this, there are many different strategies for time partitioning that could be developed in the future.

### 6.2.6 Importance Measurements and Importance Criteria

The prioritization tree construction relies on importance measurements and importance criteria to determine if a partition is significant enough to subdivide. This framework is flexible enough to allow many types of importance measurements to perform arbitrary data refinement. For computational efficiency, spatial importance measurements, in particular, ought to be associative to be able to apply our bottom-up parallel implementation. Importance measurements are categorized into three types, based on the level of the tree currently being evaluated: time axis, variable axes, or space axes.

One special case is that an explicit stopping criteria should be used in any ADR implementation: stop refinement if the priority tree data structure is growing too large. The priority tree is an additional data structure that will temporarily increase the memory footprint, but this is a trade off to intelligently lower the cost of what a simulation actually saves to disk. Note that the priority tree is a temporary data structure, because after it is created, it is immediately used to triage and select data, and then discarded afterwards.

The temporary memory cost can be controlled by a variety of factors: maximum recursion depth, size of the priority tree to memory allocation ratio, maximum number of leaves, maximum subtree size, etc. For most use cases, the inner structure of the priority tree is not actually needed, only the leaves. Thus, the tree data structure size can be reduced through depth-first, tail recursion in the top-down implementation. In the bottom-up construction, bounding hypercubes are merged, which means if left uncontrolled memory footprint in the worst case, will roughly double in size. That is the same memory overhead footprint as in the top-down method in a worst-case, tail-call optimized implementation that only keeps leaf hypercube bounds.

## 6.3 Results

#### Time Importance Examples

**Modulo Time Index:** This is the typical time selection algorithm employed by most scientific simulations, where time steps are saved at periodic intervals. The importance measurement for this calculates the time index, and the criteria is whether the index is modulo $m$. The drawback for this is that it does not inspect the time-series data, and just marks each $m$th time step as important.

**Entropy Change:** The entire entropy of all of the data in a time step is calculated. The importance criteria is whether the difference between the entropy of the current time

step and a previous important time step exceeds an absolute delta in bits. This allows us to detect large-scale changes in the distribution of values in a time-evolving simulation and use those time steps as partition markers, as shown in Figure 6.9.

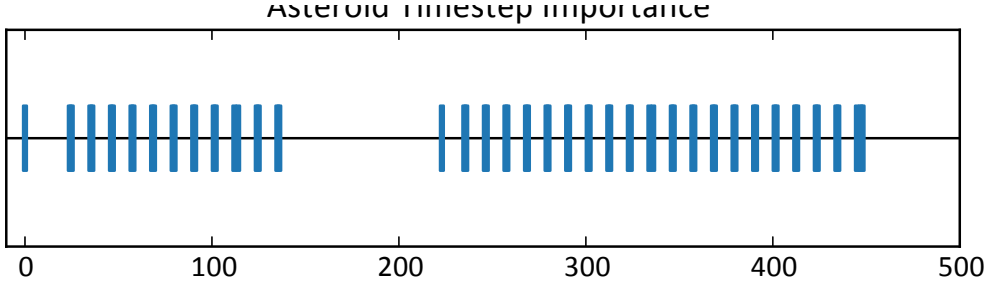



Figure 6.9: Incremental time partitioning for an asteroid impact simulation by measuring change in entropy. Out of the 500 time steps, they were grouped into 65 time partitions.

### Variable Importance Examples

**Kolmogorov-Smirnov:** The histograms for a variable are generated and the last histogram that met the importance criteria is kept. The importance is the Kolmogorov-Smirnov (K-S) distance between the two histograms. The criteria is met if the distance exceeds a threshold. In this case, the current histogram for the variable is significantly different from last histogram of the variable. Figure 6.10 shows examples of variables marked as important over time by using the K-S distance.

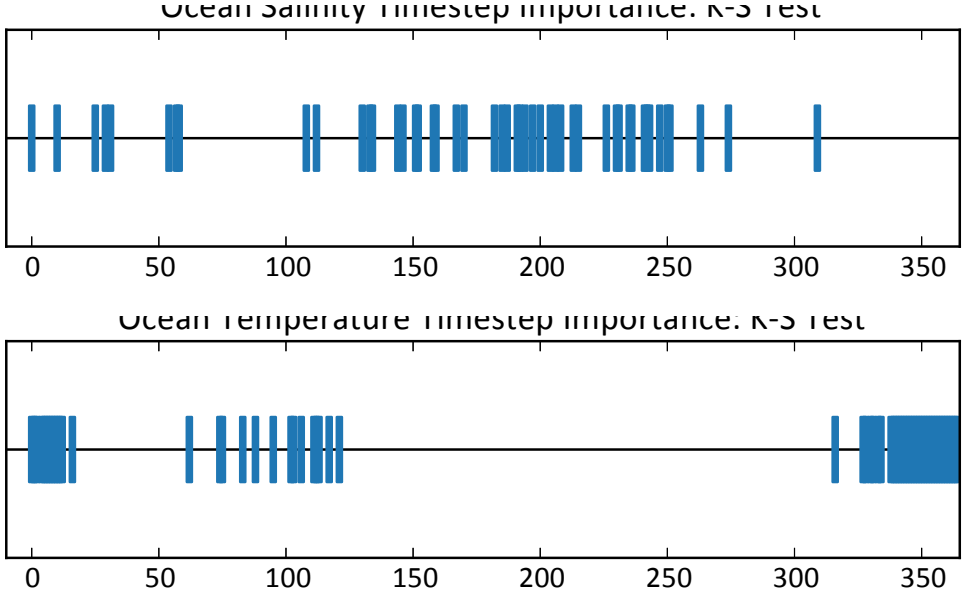



Figure 6.10: An ocean climate simulation, showing variable refinement over time, using the Kolmogorov-Smirnov distance. Out of 365 time steps, 48 salinity and 61 temperature variable partitions passed the importance criteria, and were refined further in space.

**Top N Mutual Information:** Using the work introduced by Biswas *et al.* [92], variables can be ranked relative to each other based on clustering them by the amount of mutual information contained among the different variables. Using the mutual information graph in their method, rank is assigned to each of the variables depending on the

total amount of information overlap in a cluster. The importance criteria picks the top ranked variable from each cluster, which is the most informative variable.

### Spatial Importance Examples

**Shannon Entropy:** Information entropy measures the apparent randomness or predictability of a set of data. Figure 6.11 shows an example of using entropy as a spatial importance measurement, with partitioning occurring when the entropy is relatively high.

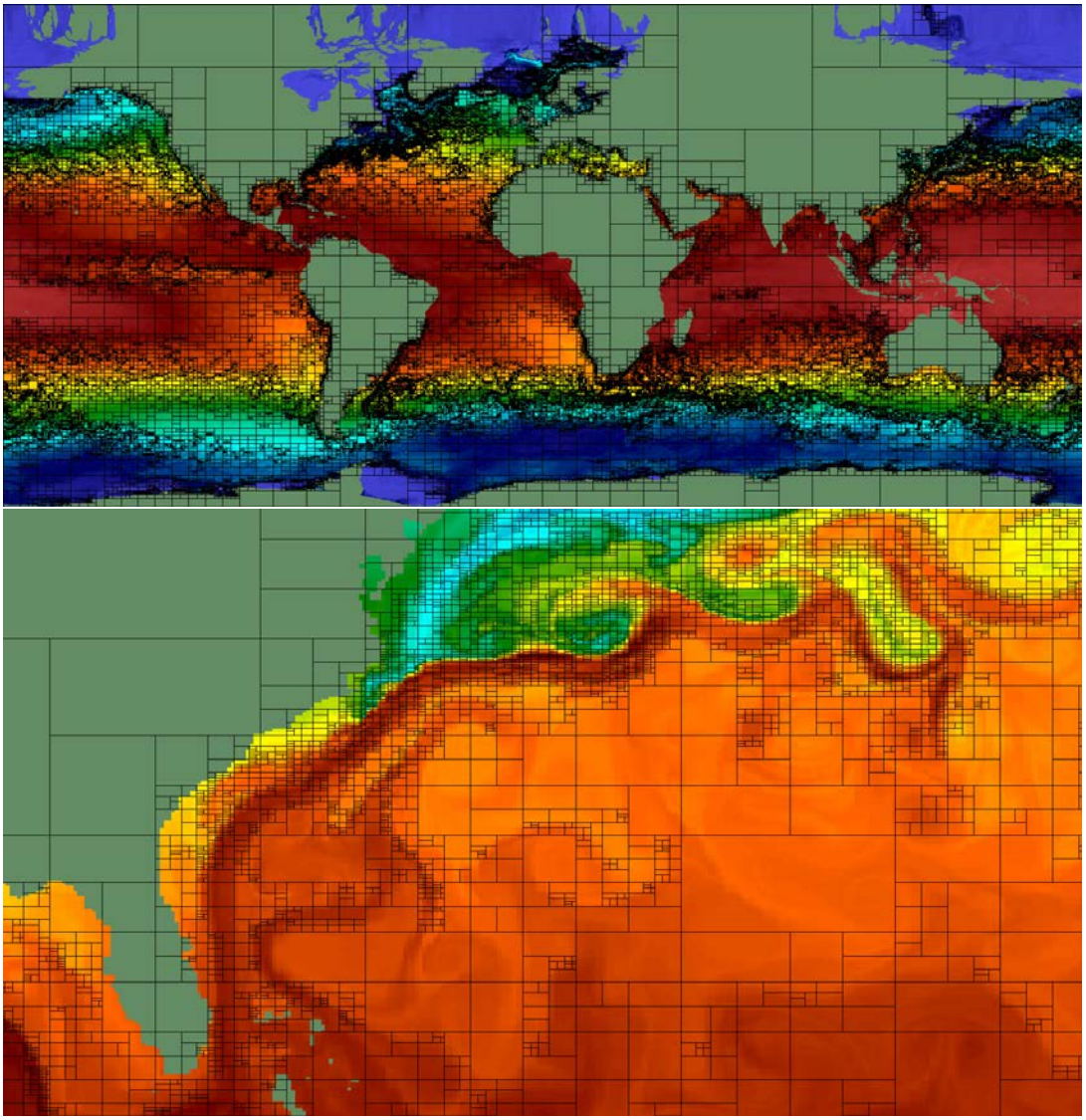


Figure 6.11: Spatial partitioning using entropy as the importance measurement. The importance criteria is whether entropy is relatively high. The top figure shows the refinement of the entire spatial domain of an ocean simulation, while the bottom figure shows a zoomed in region of the same refinement.

**Value Range:** Prioritizing user-specific values allows the data to be ranked based on values of interest. Figure 6.12 shows an example of spatial partitioning of data using maximum value and user-selected values of interest.

**Feature Census:** Any domain-specific feature measurement coupled with an importance criteria can be used in priority tree construction. For example, the Okubo-Weiss

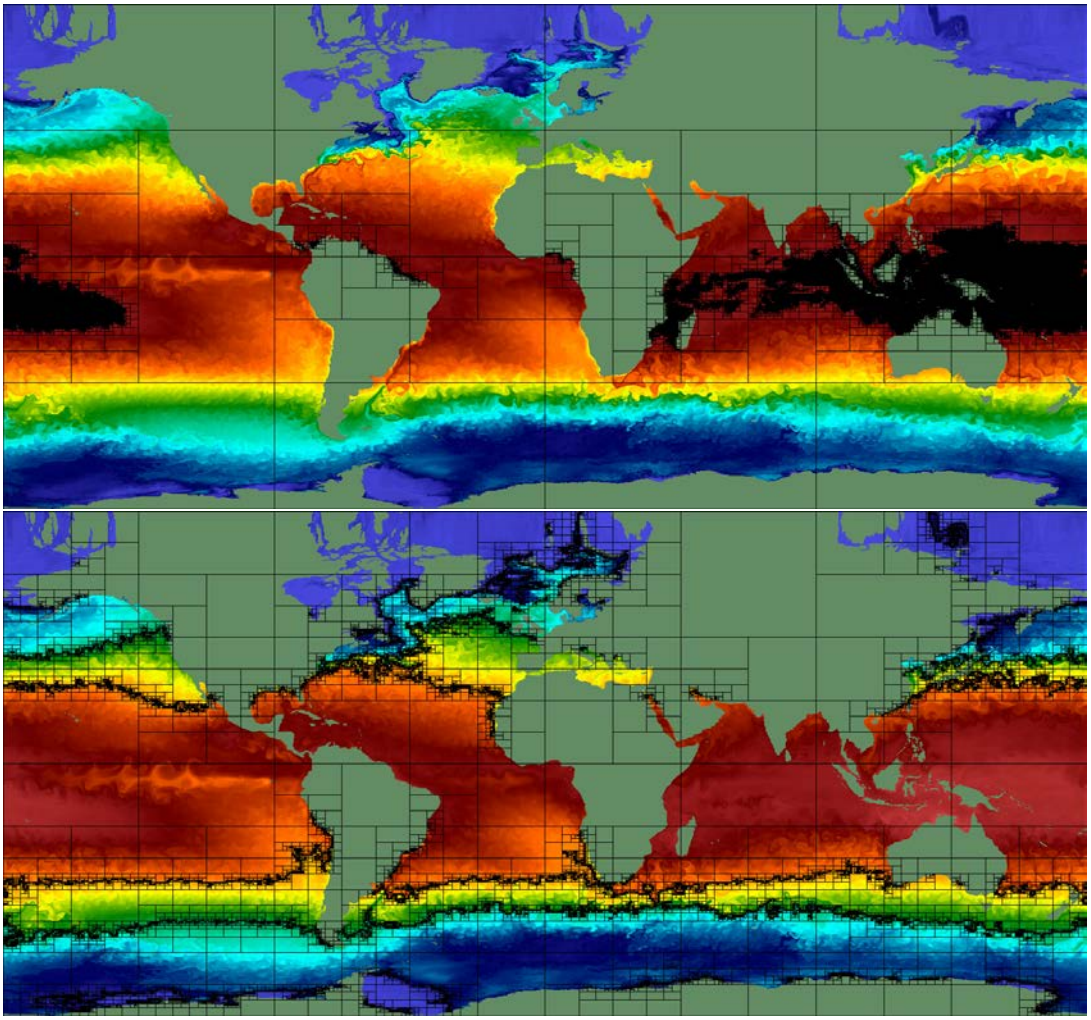


Figure 6.12: Spatial partitioning utilizing different temperature importance measurements for the ocean simulation data set. The top figure uses the global maximum value, while the bottom figure uses three user-selected values of interest. A partition passes the importance criteria if the values in the partition are within a threshold of the target importance values. Notice that the partitioning is different.

(or Lambda-2) parameter, for eddy detection in ocean simulations [100], can be used to prioritize data based on the eddy census meeting a threshold. Similarly, other features, such as the number of halos in dark matter cosmology simulations or curvature surface count in seismic data, can be used to prioritize a data set.

### 6.3.1 Data Triage and Filtering: The Selection Step

Once the organization step has completed and the prioritization tree has been built, the selection step is performed. Selection algorithms walk through the prioritization tree to generate sparse data products. This is where the triage takes place, as the data has been ranked, and it is up to the selection step algorithm to determine which data to keep, based on the priority.

Depth in the priority tree describes the relative importance of data, i.e., the data within a partition deep in the prioritization tree has passed the importance criteria more often than data which is present at a shallower part of the tree. Therefore, the most

important data, as determined the by the priority tree, is the data in the leaf partitions.

The basic tree-walking algorithm for data selection is to *walk the leaves of the priority tree with depth as the priority value of the partition.* The selection step method defines what data to save and what data products to generate from the leaves. Below, several data product generation examples are provided with descriptions of how the data is selected and stored from the leaves of the priority tree.

**Raw Data Storage:** Typical scientific simulations will store "vis dumps," or raw data every few time steps for post-processing visualization and analysis (see *Modulo Time Index* in Section 6.3). With ADR, data-introspective control can be had over raw data storage by choosing to save time steps and/or variables which have passed the analysis-driven criteria. For example, Figures 6.9 and 6.10 show time steps and variables that have passed entropy and Kolmogorov-Smirnov importance criteria, respectively. In these cases, raw data storage to disk can be triggered, rather than in periodic intervals, to store only one time step or variable per partition.

**Camera Selection:** In focus+context visualization, important data points are highlighted for the user to save time or highlight data points that may be obscured. In ADR, one way to define focus areas is to choose the spatial partitions corresponding to the lowest leaves in the tree. To visually focus on these partitions, camera bounds are created around clusters of the lowest leaves in the tree, as seen in Figure 6.13. In this example, higher leaves are pruned from the tree and the remaining leaves are spatially clustered to form groups for camera positioning.

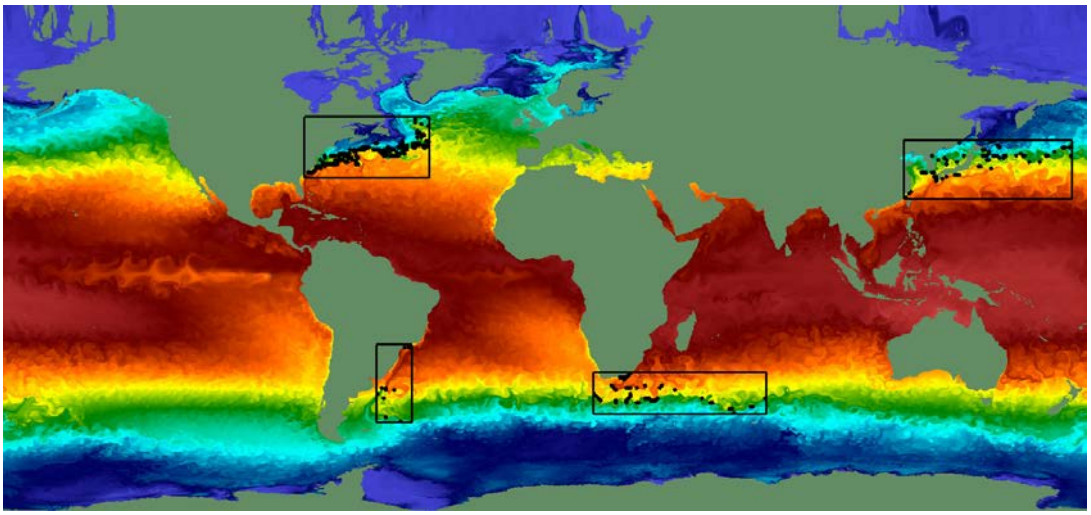


Figure 6.13: An example of producing camera placements using ADR. Tree partitions whose level are less than a threshold are filtered out, while the remaining partitions are clustered using K-means. The bounding boxes of the resulting spatial clusters form camera bounds, with four clusters shown.

**Partition Database:** The ADR priority tree algorithm essentially builds an index for the data, as the data is sorted by their position in the tree (depth, space, time, and variable). Each leaf has a bounding hypercube that defines the unique range of the data in the partition, and the priority tree is a search tree for that data. Using this indexing, data can be directly stored in a database [101,102]. The search keys for those data points are the partition bounds from the priority tree.

Table 6.1 shows the size of a database through different storage schemes for ocean climate data in an SQLite database. The three schemes shown are: 1) store all of the

Disk Usage for Database Schemes

| Max Depth | Leaves | Full Data | Compress | Sample |
|---|---|---|---|---|
| 6 | 58 | 1300 MB | 583 MB | 6.2 MB |
| 10 | 423 | 1300 MB | 582 MB | 17 MB |
| 14 | 1486 | 1300 MB | 582 MB | 46 MB |
| 18 | 2753 | 1300 MB | 612 MB | 159 MB |

Table 6.1: Disk usage for ADR partitioned ocean climate data that is stored in an SQLite database. The data in each partition is stored in the database, indexed by the depth in the tree and bounds of the partition (space, time, and variable). The Full Data column is the size of the database if all the data in a partition is stored. The Compress column is the size if the partition data is compressed with *ip*, and the Sample column is the size if a random sample is taken per partition.

data in a partition (just use the priority tree indexing), 2) compress the data in each partition, and 3) randomly sample data from the partitions, equivalent to stratified random sampling seen in [90].

**Streaming Visualization:** Ahrens et al. [74] noted that many different priority schemes could be used for prioritization of streaming data. ADR visualization, by providing a framework for defining importance and priority, is a generalization of the prioritization methods used in that work. In their work, they primarily used spatial closeness to the viewing position for the selection of streaming data. The same camera position prioritization can be provided, through selection and ordering of partitions. The leaves of the priority tree are sorted by the distance from a viewing position, and serve them in that order. This assumes the data has been partitioned into regions of interest. An alternative way of doing this is to directly incorporate distance as the priority measurement, and serve partitions based on depth ranking.

### 6.3.2 Computational Performance

I have written a parallelized prototype of ADR visualization, and tested it with two different data sets, shown in various figures throughout the chapter. Since the intent is to work *in situ*, scaling studies on the priority tree construction are performed, to show that it is efficient and can be run in conjunction with a simulation. One data set is ocean climate simulation data from the Parallel Ocean Program (POP), a tenth of a degree, high-resolution eddy resolving simulation. The native simulation grid is a structured resolution of 3600 x 2400 x 42 single floating-point data, which is roughly 1.4GB per variable. Four variables were used in our studies, with 365 simulated time steps.

In the implementation hybrid top-down and bottom-up strategy is used, where partitioning of time and variables is performed top-down, to "trim" the priority tree. Then, the bottom-up merging approach is applied to create the spatial partitions. This hybrid method works well with how parallel simulations are set up to run, which increase incrementally over time and and are distributed in space over processes. In this case, the data is scanned three times, once per time partitioning (globally once, but incrementally over time), once per variable partitioning, and once for spatial partitioning.

In Figure 6.14, strong scaling of the top-down and bottom-up priority tree construction for spatial partitioning is shown. The bottom-up construction runs very quickly, approximately in 1 to 2 seconds, even at low processor count, due to only needing to

calculate measurements once. The bottom-up reduction allows us to save computational time, as the measurement is aggregated (reduced) from leaves in the priority tree. On the other hand, the top-down construction is slow in comparison, approximately 100 seconds, due to several drawbacks: 1) it has to repartition the data at every level of the tree 2) it has to communicate more often, performing much more data movement between processors, and 3) the importance measurement is recalculated, at every level of the tree.

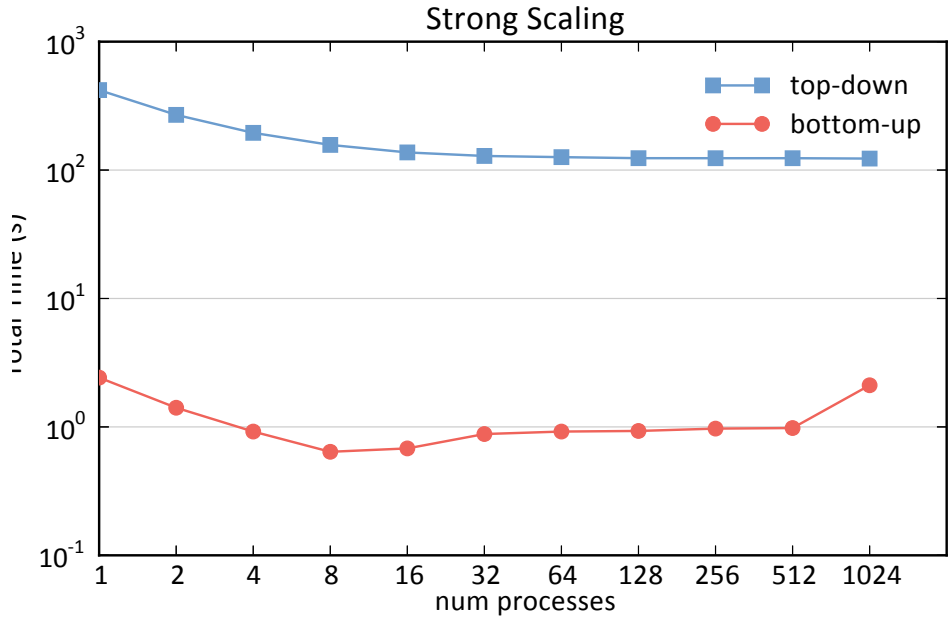



Figure 6.14: Strong scaling of the spatial partitioning with one time step of the ocean climate simulation. Both top-down and bottom-up methods are shown. The importance measurement and criteria are information entropy.

In Figure 6.15, the weak scaling of constructing the priority tree using top-down and bottom-up methods is shown. Here, the effects of communication and repartitioning is shown between multiple processors in the top-down construction. In the top-down construction, as the number of processors double, the amount of data which needs to be moved between processors also doubles. This is because the data set needs to be repartitioned between processors at every level of the tree. On the other hand, the bottom-up construction has a small amount of data communicated at each step, and only between pairs, due to pair-wise tree reduction. Top-down in the worst case will move $nlogn$ data per time step while the bottom-up will only move $plogp$ data per time step, where $p \ll n$, $p$ is the number of processors, and $n$ is the number of data points per time step.

## 6.4 Core References

Boonthanome Nouanesengsy, Jonathan Woodring, John Patchett, Kary Myers, and James Ahrens. ADR visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 43–50. IEEE, 2014

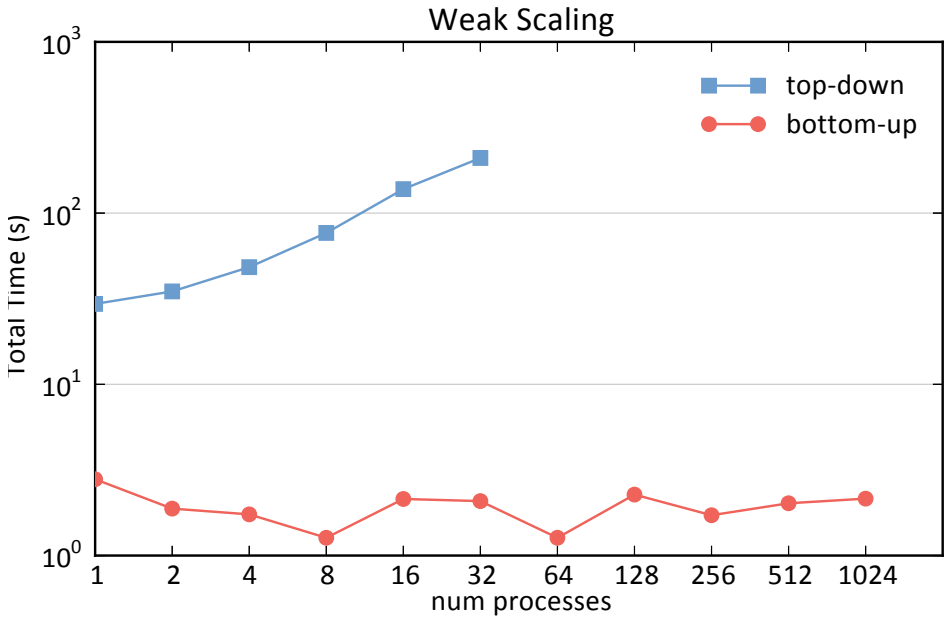



Figure 6.15: Weak scaling of the spatial partitioning with one time step of the ocean climate simulation. Both top-down and bottom-up methods are shown. The importance measurement used is information entropy. Note that top-down weak scaling cannot be run after a certain processor count due to lack of memory.

Jonathan Woodring, Mark Petersen, Andre Schmei$\beta$er, John Patchett, James Ahrens, and Hans Hagen. In situ eddy analysis in a high-resolution ocean climate model. *IEEE transactions on visualization and computer graphics*, 22(1):857–866, 2016

# Chapter 7

# Conclusion

Delivering supercomputing to the ultrascale requires the supercomputing community to solve a variety of problems. This work has focused on the visualization and data analysis aspects that need to be functional for supercomputing to work for the domain scientists that will use it to solve problems and understand a breadth of phemomena. I developed a general purpose ghost cell generator algorithm that works on unstructured grids that have already been partitioned according to the needs of the simulation. This enables in situ processes to leave the data where it resides without needing an expensive in time and space repartitioning. Second, I developed a new model for communicating with diverse teams of scientists who are designing simulation runs and making decisions about their various potential outputs. In addition, generalized production capabilities were developed at the Los Alamos National Laboratory to support the visualization and data analysis simulation products decided upon. I also studied the implications of simulation output retrieval storage patterns and developed a new analysis pipeline framework that enables improved mapping of the visualization tools readers to the storage pattern on disk, finding upto a 400x improvement in the costly read operation of persistently stored, large simulation data. With in situ visualization and data analysis the possibility of producing large counts of data products is enabled, I, and my team, developed a seminal approach to simulation output management that enables simulation scientists an interactive post hoc analysis with only saved imagery and other small products. This ongoing work enables extremely large quantities of small data products and provides search and viewing capabilities for the domain scientist. Finally, my work towards enabling unltrascale supercomputing includes automatic detection of areas of user-defined interest with the Analysis Driven Refinement project, this is a foundation for allowing a simulation to autonomously make decisions about data to preserve and data to ignore during runtime. More specifically:

In Chapter 2, A parallel multi-level ghost cell generator algorithm and performance results on two client data sets was presented. The need for ghost cells was described, as well as a specific use case which required multiple layers of ghost cells. The multi-level ghost cell generator is available in ParaView version 5.2.0 and beyond.

In Chapter 3, We presented a model with an example, including the context of a real-world use. The ideas can be leveraged by individuals and groups in designing simulation runs to ensure accessibility and flexibility needs are met. This is important to manage, not only for compute and storage resources, but importantly the scientist time. This leads us to a full delivery of in situ capability to end users. In addition, n in situ capability was successfully contributed over a one year period at LANL. The delivery was successful because of buy in from management, code development teams, production desktop, the

production supercomputing, training, and users. At least one LANL simulation code is regularly built with an in situ capability and a small subset of users are now demanding the code be built and linked with ParaView Catalyst. This work has made the catalyst libraries available for building simulation codes. It is still being made to ensure that various compiler and MPI versions are supported. ParaView Catalyst has been driven by LANL users to be improved. In particular the promise of the Cinema specifications have driven further development in the PareView Catalyst support for generating cinema databases in situ. As the technology of catalyst becomes more mature, the simple use case of writing VTK files for post processing is a big win because the simulation code teams do not have to maintain writers that catalyst provides.

In Chapter 4, I dealt with the fact that when decomposing a problem into parallel tasks, the read pattern that results from the decomposition is often overlooked. It is critical to understand this effect, since the file access pattern, combined with the format of the stored data, plays a significant role in I/O read performance. In this paper, we have introduced a model which can estimate the I/O read time for a file, given the partitioning of the file. Using this model, coupled with the flexibility of the spatiotemporal pipeline, we were able to generate read patterns which obtained far greater I/O performance versus spatial parallelism. Several timing tests showed that the optimized file access patterns resulted in a factor of more than 400 speedup. The spatio-temporal pipeline is implemented in ParaView, which is also bundled alongside UV-CDAT [1].

In Chapter 5, I developed a novel framework for an image-based approach to extreme scale data analysis, coupling visualization and analysis outputs with an image database query method to enable interactive exploration and metadata browsing. As implemented in this paper, the goals of the system are to 1) preserve important elements of the simulations, 2) to significantly reduce the data needed to preserve these elements, and 3) to offer as much flexibility as possible for post-processing exploration. We have demonstrated the framework using an open-source tool and shown how a scientist can easily define a useful set of operations that will preserve important elements of the simulation. Our results demonstrate significant data reduction, especially when considering the size of data space that can be interactively explored in a post-processing work flow. The performance section demonstrates that the in situ production of the simulation outputs weakly scale and require constant time. Finally, we have shown the flexibility of the approach, which uses compositing to enable interactive visualization. As extreme scale computing continues to grow, we expect these methods can be tailored to effectively utilize compute resources to increase the value and effectiveness of interactive, explorable results that can be produced through in situ methods.

In Chapter 6, I have described our prioritization framework for large-scale data triage and focus+context visualization. It provides an analysis imposed structure on top of scientific data, independent of the source grid. The prioritization tree can then be used for selection of data in resource constrained situations. Embedded movies of time selection and camera selection on an exploding asteroid simulation can be seen in Figures 16, 17, 18, and 19. The primary future work required for this research is that the precise amount of data resulting from the prioritization can not be known a priori. The amount of triage is controlled by importance criteria, but it does not directly control the amount of data that will

This work combined supports the ability of domain scientist to extract useful information from their simulations and run them at ever increasing scales.

# Bibliography


[1] D. Williams, C. Doutriaux, J. Patchett, S. Williams, G. Shipman, R. Miller, C. Steed, H. Krishnan, C. Silva, A. Chaudhary, P. Bremer, D. Pugmire, W. Bethel, H. Childs, M. Prabhat, B. Geveci, A. Bauer, A. Pletzer, J. Poco, T. Ellqvist, E. Santos, G. Potter, B. Smith, T. Maxwell, D. Kindig, and D. Koop. The ultra-scale visualization climate data analysis tools (UV-CDAT): Data analysis and visualization for geoscience data. *Computer*, PP(99):1–1, 2013.

[2] Stephen Hamilton, Randal Burns, Charles Meneveau, Perry Johnson, Peter Lindstrom, John Patchett, and Alexander S Szalay. Extreme event analysis in next generation simulation architectures. In *International Supercomputing Conference*, pages 277–293. Springer, 2017.

[3] John Patchett, Boonthanome Nouanesengsy, Joachim Pouderoux, James Ahrens, and Hans Hagen. Parallel multi-level ghost cell generation for distributed unstructured grids. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on*. IEEE, 2017.

[4] Carson Brownlee, John Patchett, Li-Ta Lo, David DeMarle, Christopher Mitchell, James Ahrens, and Charles Hansen. A study of ray tracing large-scale scientific data in parallel visualization applications. In *Proceedings of the Eurographics Workshop on Parallel Graphics and Visualization, EGPGV*, volume 12, pages 51–60, 2012.

[5] John M Patchett, Boonthanome Nouanesengsy, G Gisler, J Ahrens, and H Hagen. In situ and post processing workflows for asteroid ablation studies. In *Eurographics Conference on Visualization (EuroVis)*, 2017.

[6] Jonathan Woodring, Mark Petersen, Andre Schmei$\beta$er, John Patchett, James Ahrens, and Hans Hagen. In situ eddy analysis in a high-resolution ocean climate model. *IEEE transactions on visualization and computer graphics*, 22(1):857–866, 2016.

[7] John Patchett, Boonthanome Nouanesengsy, Patricia Fasel, and James Ahrens. 2016 CSSE L3 milestone: Deliver in situ to XTD end users. Technical report, Los Alamos National Laboratory, 2016. LA-UR-16-26987.

[8] Francesca Samsel, John M Patchett, David Honegger Rogers, and Karen Tsai. Employing color theory to visualize volume-rendered multivariate ensembles of asteroid impact simulations. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1126–1134. ACM, 2017.

[9] Boonthanome Nouanesengsy, John Patchett, James Ahrens, Andrew Bauer, Aashish Chaudhary, Ross Miller, Berk Geveci, Galen M Shipman, and Dean N

Williams. A model for optimizing file access patterns using spatio-temporal parallelism. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 4. ACM, 2013.

[10] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press.

[11] John Patchett, James Ahrens, Boonthanome Nouanesengsy, Patricia Fasel, Patrick O'leary, Chris Sewell, Jon Woodring, Christopher Mitchell, Li-Ta Lo, Kary Myers, Joanne Wendelberger, Curt Canada, Marcus Daniels, Hilary Abhold, and Gabe Rockefeller. Case study of in situ data analysis in ASC integrated codes, 2013. LA-UR-13-26599.

[12] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[13] Boonthanome Nouanesengsy, Jonathan Woodring, John Patchett, Kary Myers, and James Ahrens. ADR visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 43–50. IEEE, 2014.

[14] John M Patchett, Boonthanome Nouanesengsy, James Paul Ahrens, Michael Kenneth Lang, David Honegger Rogers, Jennifer Kathleen Green, Francesca Samsel, Giovanni Antonio Cone, and Hans-Jurgen Hagen. Delivery of in situ capability to end users. *Visualization in Practice*, 2017.

[15] John Patchett, Francesca Samsel, Karen Tsai, Galen Gisler, David Rogers, Greg Abram, and Terece Turton. Visualization and analysis of threats from asteroid ocean impacts. 2016. Winner, Best Scientific Visualization & Data Analytics Showcase; LA-UR-16-26258.

[16] Utkarsh Ayachit, Berk Geveci, Kenneth Moreland, John Patchett, and Jim Ahrens. The ParaView visualization application. *High Performance VisualizationEnabling Extreme-Scale Scientific Insight*, pages 383–400, 2012.

[17] James Ahrens, Li-Ta Lo, Boonthanome Nouanesengsy, John Patchett, and Allen McPherson. Petascale visualization: Approaches and initial results. In *Ultrascale Visualization, 2008. UltraVis 2008. Workshop on*, pages 24–28. IEEE, 2008. LA-UR-10-02237.

[18] Jonathan Woodring, Susan Mniszewski, Christopher Brislawn, David DeMarle, and James Ahrens. Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 31–38. IEEE, 2011.

[19] Sean Williams, Mark Petersen, Matthew Hecht, Mathew Maltrud, John Patchett, J Ahrens, and Bernd Hamann. Interface exchange as an indicator for eddy heat transport. In *Computer Graphics Forum*, volume 31, pages 1125–1134. Wiley Online Library, 2012.

[20] William J Schroeder and Kenneth M Martin. The visualization toolkit-30. 1996.

[21] James Ahrens, Kristi Brislawn, Ken Martin, Berk Geveci, C Charles Law, and Michael Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer graphics and Applications*, 21(4):34–41, 2001.

[22] Kenneth Moreland, Lisa Avila, and Lee Ann Fisk. Parallel unstructured volume rendering in ParaView. *Proceedings of SPIE Visualization and Data Analysis*, 6495:6495 – 12, January 2007.

[23] Michael Gittings, Robert Weaver, Michael Clover, Thomas Betlach, Nelson Byrne, Robert Coker, Edward Dendy, Robert Hueckstaedt, Kim New, W Rob Oakes, Dale Ranta, and Ryan Stefan. The RAGE radiation-hydrodynamic code. *Computational Science & Discovery*, 1(1):015005, 2008.

[24] John Patchett and Galen Gisler. Deep water impact ensemble data set. Technical report, 2017. LA-UR-17-21595.

[25] James Ahrens, Berk Geveci, Charles Law, CD Hansen, and CR Johnson. *ParaView: An End-User Tool for Large-Data Visualization*, chapter 36, pages 717–731. Elsevier, 2005.

[26] Hank Childs. VisIt: An end-user tool for visualizing and analyzing very large data. December 2013. LBNL Paper LBNL-6320E.

[27] Amy Henderson Squillacote and James Ahrens. *The paraview guide*, volume 366. Kitware, 2007.

[28] James Ahrens, John Patchett, Li-Ta Lo, David DeMarle, Carson Brownlee, and Christopher Mitchell. A report documenting the completion of the Los Alamos National Laboratory portion of the ASC level II milestone, visualization on the supercomputing platform, ASC level II milestone meeting, 2010. LA-UR-11-00494.

[29] Staff. *VisIt User's Manual*. Lawrence Livermore National Laboratory, Livermore, CA, 1.5 edition, 10 2005. UCRL-SM-220449.

[30] Martin Isenburg, Peter Lindstrom, and Hank Childs. Parallel and streaming generation of ghost data for structured grids. *IEEE Computer Graphics and Applications*, 30(3):32–44, May 2010.

[31] Gunther H Weber, Hank Childs, and Jeremy S Meredith. Efficient parallel extraction of crack-free isosurfaces from adaptive mesh refinement (amr) data. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 31–38. IEEE, 2012.

[32] NASA. Newly released map data shows frequency of small asteroid impacts, provides clues on larger asteroid population, 2016.

[33] P Jenniskens, MH Shaddad, D Numan, S Elsir, AM Kudoda, ME Zolensky, L Le, GA Robinson, JM Friedrich, D Rumble, et al. The impact and recovery of asteroid 2008 tc3. *Nature*, 458(7237):485–488, 2009.

[34] Olga P Popova, Peter Jenniskens, Vacheslav Emelyanenko, Anna Kartashova, Eugeny Biryukov, Sergey Khaibrakhmanov, Valery Shuvalov, Yurij Rybnov, Alexandr Dudorov, Victor I Grokhovsky, et al. Chelyabinsk airburst, damage assessment, meteorite recovery, and characterization. *Science*, 342(6162):1069–1073, 2013.

[35] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C Bauer, Pat Marion, B Gevecik, Michel Rasquin, and Kenneth E Jansen. The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011.

[36] Hank Childs, Eric Brugger, Kathleen Bonnell, Jeremy Meredith, Mark Miller, Brad Whitlock, and Nelson Max. A contract based system for large data visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 191–198. IEEE, 2005.

[37] NASA. Asteroid-generated tsunami (AGT) and associated risk assessment. online, August 2016. https://tsunami-workshop.arc.nasa.gov/workshop2016/.

[38] John Patchett, Galen Gisler, Boonthanome Nouanesengsy, David H. Rogers, Greg Abram, Francesca Samsel, Karen Tsai, and Terece Turton. Visualization and analysis of threats from asteroid ocean impacts. Available at https://youtu.be/yeXcgnj8AG0 LA-UR-16-28934.

[39] John T Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems*, 22(3):303–312, 2006.

[40] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434. IEEE Press, 2014.

[41] Sean Williams, Mark Petersen, Peer-Timo Bremer, Matthew Hecht, Valerio Pascucci, James Ahrens, Mario Hlawitschka, and Bernd Hamann. Adaptive extraction and quantification of geophysical vortices. *IEEE transactions on visualization and computer graphics*, 17(12):2088–2095, 2011.

[42] Analyzing and visualizing cosmological simulations with ParaView, author=Woodring, Jonathan and Heitmann, Katrin and Ahrens, James and Fasel, Patricia and Hsu, Chung-Hsing and Habib, Salman and Pope, Adrian, journal=The Astrophysical Journal Supplement Series, volume=195, number=1, pages=11, year=2011, publisher=IOP Publishing.

[43] Sheng Di and Franck Cappello. Fast error-bounded lossy hpc data compression with sz. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 730–739. IEEE, 2016.

[44] Allison H Baker, Dorit M Hammerling, Sheri A Mickelson, Haiying Xu, Martin B Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, et al. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381, 2016.

[45] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–11. IEEE, 2012.

[46] Kenneth Moreland, Christopher Sewell, William Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, et al. Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, 2016.

[47] Russell L Ackoff. From data to wisdom. *Journal of applied systems analysis*, 16(1):3–9, 1989.

[48] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. ParaView catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 25–29. ACM, 2015.

[49] Kitware, Inc. *The Visualization Toolkit User's Guide*, January 2003.

[50] David Camp, Hank Childs, Amit Chourasia, Christoph Garth, and Kenneth I Joy. Evaluating the benefits of an extended memory hierarchy for parallel streamline algorithms. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 57–64. IEEE, 2011.

[51] Michael L Norman and Allan Snavely. Accelerating data-intensive science with gordon and dash. In *Proceedings of the 2010 TeraGrid Conference*, page 14. ACM, 2010.

[52] Christopher Michell, James Ahrens, and Jun Wang. VisIO: Enabling interactive visualization of ultra-scale, time series data via high-bandwidth distributed i/o systems. pages 1–12. IEEE International Parallel and Distributed Processing Symposium, May 2011.

[53] Venkatram Vishwanath, Mark Hereld, and Michael E Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14. IEEE, 2011.

[54] Brad Whitlock, Jean M Favre, and Jeremy S Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 101–109. Eurographics Association, 2011.

[55] Matthew Woitaszek, John M. Dennis, and Taleena R. Sines. Parallel high-resolution climate data analysis using swift. In *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, MTAGS '11, pages 5–14, New York, NY, USA, 2011. ACM.

[56] Hongfeng Yu and Kwan-Liu Ma. A study of i/o methods for parallel visualization of large-scale data. *Parallel Computing*, 31(2):167 – 183, 2005. Parallel Graphics and Visualization.

[57] Hongfeng Yu, Kwan-Liu Ma, and Joel Welling. A parallel visualization pipeline for terascale earthquake simulations. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 49–, Washington, DC, USA, 2004. IEEE Computer Society.

[58] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Gunther H Weber, E Wes Bethel, et al. Extreme scaling of production visualization software on diverse architectures. *Computer Graphics and Applications, IEEE*, 30(3):22–31, 2010.

[59] Prabhat, Oliver Rbel, Surendra Byna, Kesheng Wu, Fuyu Li, Michael Wehner, and Wes Bethel. Teca: A parallel toolkit for extreme climate analysis. *Procedia Computer Science*, 9(0):866 – 876, 2012. Proceedings of the International Conference on Computational Science, 2012.

[60] Wesley Kendall, Jian Huang, Tom Peterka, Robert Latham, and Robert Ross. Toward a general i/o layer for parallel-visualization applications. *Computer Graphics and Applications, IEEE*, 31(6):6–10, 2011.

[61] Tom Peterka, Robert Ross, Attila Gyulassy, Valerio Pascucci, Wesley Kendall, Han-Wei Shen, Teng-Yok Lee, and Abon Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 105–112. IEEE, 2011.

[62] John Biddiscombe, Berk Geveci, Ken Martin, Kenneth Moreland, and David Thompson. Time dependent processing in a parallel pipeline architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1376–1383, November 2007.

[63] Anna Tikhonova, Carlos D Correa, and K-L Ma. An exploratory technique for coherent visualization of time-varying volume data. In *Computer Graphics Forum*, volume 29, pages 783–792. Wiley Online Library, 2010.

[64] 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*, page 91. ACM, 2006.

[65] Kenneth Moreland, Nathan Fabian, Pat Marion, and Berk Geveci. Visualization on supercomputing platform level ii asc milestone (3537-1b) results from sandia.

[66] Hongfeng Yu, Chaoli Wang, Ray W Grout, Jacqueline H Chen, and Kwan-Liu Ma. In situ visualization for large-scale combustion simulations. *IEEE computer graphics and applications*, 30(3):45–57, 2010.

[67] Kwan-Liu Ma. In situ visualization at extreme scale: Challenges and opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.

[68] Eric J Luke and Charles D Hansen. Semotus visum: a flexible remote visualization framework. In *Proceedings of the conference on Visualization'02*, pages 61–68. IEEE Computer Society, 2002.

[69] Harry Shum and Sing B Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing 2000*, pages 2–13. International Society for Optics and Photonics, 2000.

[70] Shigeo Takahashi, Issei Fujishiro, Yuriko Takeshima, and Tomoyuki Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 495–502. IEEE, 2005.

[71] Udeepta D Bordoloi and H-W Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494. IEEE, 2005.

[72] Guangfeng Ji and Han-Wei Shen. Dynamic view selection for time-varying volumes. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[73] John M Patchett, James P Ahrens, Boonthanome Nouanesengsy, Patricia K Fasel, Patrick W Oleary, Christopher Meyer Sewell, Jonathan L Woodring, Christopher J Mitchell, Li-Ta Lo, Kary L Myers, et al. Lanl csse l2: Case study of in situ data analysis in asc integrated codes. *Technical report LA-UR-13-26599*, 2013.

[74] J. P. Ahrens, N. Desai, P. S. McCormick, K. Martin, and J. Woodring. A modular extensible visualization system architecture for culled prioritized data streaming. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6495 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, January 2007.

[75] Stphane Popinet. Gerris: A tree-based adaptive solver for the incompressible euler equations in complex geometries. *J. Comp. Phys*, 190:572–600, 2003.

[76] B Fryxell, K Olson, P Ricker, FX Timmes, M Zingale, DQ Lamb, P MacNeice, R Rosner, JW Truran, and H Tufo. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131(1):273, 2000.

[77] A Mignone, G Bodo, S Massaglia, Titos Matsakos, O Tesileanu, C Zanni, and A Ferrari. Pluto: A numerical code for computational astrophysics. *The Astrophysical Journal Supplement Series*, 170(1):228, 2007.

[78] Matthias Steinmetz. Grapesph: cosmological smoothed particle hydrodynamics simulations with the special-purpose hardware grape. *Monthly Notices of the Royal Astronomical Society*, 278(4):1005–1017, 1996.

[79] Romain Teyssier. Cosmological hydrodynamics with adaptive mesh refinement: a new high resolution code called ramses. *Astron.Astrophys.*, 385:337–364, 2002.

[80] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 457–462, New York, NY, USA, 2004. ACM.

[81] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989.

[82] M. J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53:484–512, March 1984.

[83] R. Löhner. An adaptive finite element scheme for transient problems in cfd. *Comput. Methods Appl. Mech. Eng.*, 61(3):323–338, April 1987.

[84] Deborah Silver and Norman J Zabusky. Quantifying visualizations for reduced modeling in nonlinear science: Extracting structures from data sets. *Journal of visual communication and image representation*, 4(1):46–61, 1993.

[85] F.J. Post, T. van Walsum, F.H. Post, and D. Silver. Iconic techniques for feature visualization. In *Visualization, 1995. Visualization '95. Proceedings., IEEE Conference on*, pages 288–295, 464, Oct 1995.

[86] Fan-Yin Tzeng and Kwan-Liu Ma. Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 6. IEEE Computer Society, 2005.

[87] Guangfeng Ji and Han-Wei Shen. Feature tracking using earth movers distance and global optimization. In *Pacific Graphics*. Citeseer, 2006.

[88] Frits H Post, Benjamin Vrolijk, Helwig Hauser, Robert S Laramee, and Helmut Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792. Wiley Online Library, 2003.

[89] Xin Tong, Teng-Yok Lee, and Han-Wei Shen. Salient time steps selection from large scale time-varying data sets with dynamic time warping. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 49–56, Oct 2012.

[90] Jonathan Woodring, James Ahrens, Jeannette Figg, Joanne Wendelberger, and Katrin Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. EuroVis '11, May 2011.

[91] Zhe Fang, Torsten Moeller, Ghassan Hamarneh, and Anna Celler. Visualization and exploration of time-varying medical image data sets. In *Proceedings of Graphics Interface 2007*, pages 281–288. ACM, 2007.

[92] Ayan Biswas, Soumya Dutta, Han-Wei Shen, and Jonathan Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, December 2013.

[93] Chaoli Wang, Antonio Garcia, and Han-Wei Shen. Interactive level-of-detail selection using image-based quality metric for large volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):122–134, 2007.

[94] Chaoli Wang and Han-Wei Shen. Lod map - a visual interface for navigating multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1029–1036, 2006.

[95] Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Importance-driven time-varying data visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1547–1554, Nov 2008.

[96] U.D. Bordoloi and Han-Wei Shen. View selection for volume rendering. In *Visualization, 2005. VIS 05. IEEE*, pages 487–494, Oct 2005.

[97] Teng-Yok Lee, O. Mishchenko, Han-Wei Shen, and R. Crawfis. View point evaluation and streamline filtering for flow visualization. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 83–90, March 2011.

[98] Konrad Muehler, Mathias Neugebauer, Christian Tietjen, and Bernhard Preim. Viewpoint Selection for Intervention Planning . pages 267–274, Norrköping, Sweden, 2007. Eurographics Association.

[99] Pere-Pau Vzquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum*, 22(4):689–700, 2003.

[100] Sean Williams, Matthew Hecht, Mark Petersen, Richard Strelitz, Mathew Maltrud, James P. Ahrens, Mario Hlawitschka, and Bernd Hamann. Visualization and analysis of eddies in a global ocean simulation. *Comput. Graph. Forum*, 30(3):991–1000, 2011.

[101] R. J. Brunner, I. Csabai, A. Szalay, A. J. Connolly, G. P. Szokoly, and K. Ramaiyer. The Science Archive for the Sloan Digital Sky Survey. In G. H. Jacoby and J. Barnes, editors, *Astronomical Data Analysis Software and Systems V*, volume 101 of *Astronomical Society of the Pacific Conference Series*, page 493, 1996.

[102] Jason Graham, Edward Givelberg, and Kalin Kanov. Run-time creation of the turbulent channel flow database by an hpc simulation using mpi-db. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 151–156, New York, NY, USA, 2013. ACM.

[103] James Ahrens, John Patchett, Andrew Bauer, Sébastien Jourdain, David H Rogers, Mark Petersen, Benjamin Boeckel, Patrick OLeary, Patricia Fasel, and Francesca Samsel. In situ mpas-ocean image-based visualization. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Visualization & Data Analytics Showcase*, 2014.

[104] James P. Ahrens, Jonathan Woodring, David E. DeMarle, John Patchett, and Mathew Maltrud. Interactive remote large-scale data visualization via prioritized multi-resolution streaming. In *Proceedings of the 2009 Workshop on Ultrascale Visualization*, UltraVis '09, pages 1–10, New York, NY, USA, 2009. ACM.

[105] Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett. Slic: Scheduled linear image compositing for parallel volume rendering. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization*

*and Graphics*, PVG '03, pages 6–, Washington, DC, USA, 2003. IEEE Computer Society.

# Appendix A

# CV

## John M. Patchett

Los Alamos National Laboratory
Computer, Computational, and Statistical Sciences Division
PO Box 1663, Mail Stop B287
Los Alamos, NM 87544
patchett@lanl.gov

### Education and Training

| | | | |
|---|---|---|---|
| 2017 | Doctoral Candidate | Computer Science | TU Kaiserslautern |
| 2011 | MSc | Computer Science | University of New Mexico |
| 2003 | BA | Anthropology | University of New Mexico |

### Research and Professional Experience

| | | |
|---|---|---|
| Los Alamos National Laboratory | Staff Scientist | Oct 2005 – present |
| Advanced Computing Laboratory | Systems Administrator | Oct 1999 – Oct 2005 |
| Protection Technology Los Alamos | Systems Administrator | Oct 1997 – Oct 1999 |
| The PC Place | Assistant Sales Manager | Jun 1995 – Sep 1997 |

### Selected Publications

- John M Patchett, Boonthanome Nouanesengsy, G Gisler, J Ahrens, and H Hagen. In situ and post processing workflows for asteroid ablation studies. In *Eurographics Conference on Visualization (EuroVis)*, 2017

- John Patchett, Boonthanome Nouanesengsy, Joachim Pouderoux, James Ahrens, and Hans Hagen. Parallel multi-level ghost cell generation for distributed unstructured grids. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on*. IEEE, 2017

- John M Patchett, Boonthanome Nouanesengsy, James Paul Ahrens, Michael Kenneth Lang, David Honegger Rogers, Jennifer Kathleen Green, Francesca Samsel, Giovanni Antonio Cone, and Hans-Jurgen Hagen. Delivery of in situ capability to end users. *Visualization in Practice*, 2017

- Francesca Samsel, John M Patchett, David Honegger Rogers, and Karen Tsai. Employing color theory to visualize volume-rendered multivariate ensembles of asteroid impact simulations. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1126–1134. ACM, 2017

- Stephen Hamilton, Randal Burns, Charles Meneveau, Perry Johnson, Peter Lindstrom, John Patchett, and Alexander S Szalay. Extreme event analysis in next generation simulation architectures. In *International Supercomputing Conference*, pages 277–293. Springer, 2017

- Jonathan Woodring, Mark Petersen, Andre Schmei$\beta$er, John Patchett, James Ahrens, and Hans Hagen. In situ eddy analysis in a high-resolution ocean climate model. *IEEE transactions on visualization and computer graphics*, 22(1):857–866, 2016

- John Patchett, Boonthanome Nouanesengsy, Patricia Fasel, and James Ahrens. 2016 CSSE L3 milestone: Deliver in situ to XTD end users. Technical report, Los Alamos National Laboratory, 2016. LA-UR-16-26987

- John Patchett, Francesca Samsel, Karen Tsai, Galen Gisler, David Rogers, Greg Abram, and Terece Turton. Visualization and analysis of threats from asteroid ocean impacts. 2016. Winner, Best Scientific Visualization & Data Analytics Showcase; LA-UR-16-26258

- James Ahrens, John Patchett, Andrew Bauer, Sébastien Jourdain, David H Rogers, Mark Petersen, Benjamin Boeckel, Patrick OLeary, Patricia Fasel, and Francesca Samsel. In situ mpas-ocean image-based visualization. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Visualization & Data Analytics Showcase*, 2014

- James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press

- Boonthanome Nouanesengsy, Jonathan Woodring, John Patchett, Kary Myers, and James Ahrens. ADR visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 43–50. IEEE, 2014

- John Patchett, James Ahrens, Boonthanome Nouanesengsy, Patricia Fasel, Patrick O'leary, Chris Sewell, Jon Woodring, Christopher Mitchell, Li-Ta Lo, Kary Myers, Joanne Wendelberger, Curt Canada, Marcus Daniels, Hilary Abhold, and Gabe Rockefeller. Case study of in situ data analysis in ASC integrated codes, 2013. LA-UR-13-26599

- D. Williams, C. Doutriaux, J. Patchett, S. Williams, G. Shipman, R. Miller, C. Steed, H. Krishnan, C. Silva, A. Chaudhary, P. Bremer, D. Pugmire, W. Bethel, H. Childs, M. Prabhat, B. Geveci, A. Bauer, A. Pletzer, J. Poco, T. Ellqvist, E. Santos, G. Potter, B. Smith, T. Maxwell, D. Kindig, and D. Koop. The ultra-scale visualization climate data analysis tools (UV-CDAT): Data analysis and visualization for geoscience data. *Computer*, PP(99):1–1, 2013

- Boonthanome Nouanesengsy, John Patchett, James Ahrens, Andrew Bauer, Aashish Chaudhary, Ross Miller, Berk Geveci, Galen M Shipman, and Dean N Williams. A model for optimizing file access patterns using spatio-temporal parallelism. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 4. ACM, 2013

- J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012

- Carson Brownlee, John Patchett, Li-Ta Lo, David DeMarle, Christopher Mitchell, James Ahrens, and Charles Hansen. A study of ray tracing large-scale scientific data in parallel visualization applications. In *Proceedings of the Eurographics Workshop on Parallel Graphics and Visualization, EGPGV*, volume 12, pages 51–60, 2012

- James P. Ahrens, Jonathan Woodring, David E. DeMarle, John Patchett, and Mathew Maltrud. Interactive remote large-scale data visualization via prioritized multi-resolution streaming. In *Proceedings of the 2009 Workshop on Ultrascale Visualization*, UltraVis '09, pages 1–10, New York, NY, USA, 2009. ACM

- Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett. Slic: Scheduled linear image compositing for parallel volume rendering. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '03, pages 6–, Washington, DC, USA, 2003. IEEE Computer Society