EuroVis 2016
R. Maciejewski, T. Ropinski, and A. Vilanova
(Guest Editors)

Volume 35 (2016), Number 3
STAR – State of The Art Report

In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms

A. C. Bauer[1], H. Abbasi[2], J. Ahrens[3], H. Childs[4], B. Geveci[1], S. Klasky[2], K. Moreland[5], P. O'Leary[1], V. Vishwanath[6], B. Whitlock[7] and E. W. Bethel[8]

[1]Kitware Inc., USA
[2]Oak Ridge National Laboratory, USA
[3]Los Alamos National Laboratory, USA
[4]University of Oregon, USA
[5]Sandia National Laboratories, USA
[6]Argonne National Laboratory, USA
[7]Intelligent Light, USA
[8]Lawrence Berkeley National Laboratory, USA

Abstract

The considerable interest in the high performance computing (HPC) community regarding analyzing and visualization data without first writing to disk, i.e., in situ processing, is due to several factors. First is an I/O cost savings, where data is analyzed/visualized while being generated, without first storing to a filesystem. Second is the potential for increased accuracy, where fine temporal sampling of transient analysis might expose some complex behavior missed in coarse temporal sampling. Third is the ability to use all available resources, CPU's and accelerators, in the computation of analysis products. This STAR paper brings together researchers, developers and practitioners using in situ methods in extreme-scale HPC with the goal to present existing methods, infrastructures, and a range of computational science and engineering applications using in situ analysis and visualization.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

The traditional use model for analysis and visualization has been to write data first to persisent storage, then later read it back into memory for the purpose of analysis or visualization. This *post hoc* usage reflects that visualization or analysis is performed "after the fact." An alternative approach, for which we use the umbrella term *in situ*, is one where visualization or analysis processing happens without first writing data to persistent storage. While the concept of *in situ* processing has existed for several decades, the motivations for its use have varied over time in response to changing needs of the scientific community and to the changing balance of computational architectures.

In the present, there is a great deal of active work and interest in *in situ* methods, infrastructures, and applications. This interest is particularly strong in high performance computing (HPC) where machines are built with multiple processors to achieve much greater computational capacity than is possible with single systems. The largest HPC systems today aggregate the power of many thousands of processors to achieve over 10^{16} floating point operations per second (FLOPS). This STAR report brings into focus several different dimensions of this vibrant area into one place with an emphasis on the many significant recent advances of *in situ* for HPC applications. It is timely, due to the rapidly changing landscape of computational architecture that accompanies the evolution from petascale- to exascale-regime computing, along with the attendant challenges facing computational, computer, and domain scientists. Because of breadth and diversity in the *in situ* space, as well as a significant amount of historical work, this STAR report covers a significant amount of material in several different dimensions. However, the scope of this STAR report is limited to HPC applications and does not include "big data" applications or HPC architectures designed for big data.

We begin with a discussion of the motivations for *in situ* methods and approaches (§ 2). In some cases, these motivations include the desire to perform computational steering, which can help in decreasing time-to-solution for some challenging computational problems. In other cases, pursuit of *in situ* methods is a practical necessity due to the architectural balance present in modern HPC platforms that favors FLOPS over I/O.

Owing to the diversity of approaches and considerations, we de-

vote an entire subsection (§ 3) to terminology in this space. Because there are many different ways of performing *in situ* processing, and because this style of processing has been labelled with different terms over the decades, there is a rich and diverse vocabulary of terms that are in use to describe different aspects of this domain.

Next, we provide a review of previous and current work in this space (§ 4), some of which goes back over two decades. In some cases, early work is targeting computational steering, while more recent works focus on infrastructures and on methods that help overcome some of the fundamental limitations of *in situ* approaches, namely the limited ability for exploratory visualization analysis as a post process.

Bringing the focus more into the present, we next go into some depth for four different *in situ* infrastructures (§ 5), with an eye towards helping give a sense of how a developer would integrate one of these infrastructures with a simulation code along with a view of the types of *in situ* methods and operations each supports.

We devote an entire section (§ 6) to show the use of *in situ* methods and infrastructure applied to specific contemporary science problems in climate modeling, structural engineering, and magnetic fusion energy modeling.

2. Motivation for *In Situ* Methods and Infrastructure

On the surface, one might be inclined to believe that it is somehow simpler and less complex to simply have a simulation write data to persistent storage and then later do *post hoc* analysis or visualization. After all, why would one want to potentially increase the complexity of an already complex simulation code with the addition of new code that does additional analysis or visualization processing? That line of reasoning is predicated upon two assumptions, though, that don't always hold true. The first is the view that the simulation is a write-only "black box" that will run, from start to finish, free from any sort of intervention or external and dynamic input, which is a capability statement. The second is that it is actually possible to write some, or all, data to persistent storage for subsequent analysis, which is a capacity statement. A third issue is the very real fact of economics: moving data around is expensive in terms of energy, and disks and I/O fabric are expensive, as is having separate computational infrastructure for performing visualization and analysis.

As we discuss later (§ 4), a significant body of earlier work in the *in situ* space focused on computational steering and interaction, which takes aim at the capability argument. In some cases, finding a good set of initial conditions for a simulation, so that it can converge to a meaningful result, is not something that can be done analytically. Having means to interact with a simulation, to gain quick feedback on how a change in an input parameter will affect the simulation, or perform on-the-fly debugging, is a significant capability that can, in the long run, result in more efficient use of computational resources. In other cases, some computational problems that are known to be *NP-hard*, can be guided towards a solution with a human-in-the-loop, who can cull portions of the potential solution space believed to be non-optimal. Finally, in many cases, it is desirable to periodically check on the progress of a running simulation, and perhaps induce early termination if it is not going well. In all these cases, it might be possible to accomplish these objectives using a *post hoc* processing path, where simulation data is written first to persistent storage, then read from storage for visualization or analysis processing. This approach would be possible only if it is possible write all the data to persistent storage. The *post hoc* approach here is, by definition, a one-way flow of information, and would preclude somehow providing information back into the simulation.

The real challenge, though, and the one that is the primary focus of most in the community today, is the fact that it is increasingly difficult to write simulation data to persistent storage. There is a very clear reason for this challenge, and very real impacts on our ability to perform scientific investigation. The reason stems from the fact there simply is insufficient I/O capacity on modern HPC systems, and that the problem will be getting worse. Therefore, *in situ* methods and approaches are of increasing interest and importance to the HPC computational science community.

An ongoing trend in high performance computing is an exponential increase of the computational throughput of the machine, but a comparatively much smaller increase in the bandwidth to the disk storage system [Ahe12, BDE13, Ahr15, Mor16]. This has lead to a very large disparity between the computation bandwidth and storage bandwidth even today. For example, as demonstrated in Figure 1 on the Titan supercomputer at the Oak Ridge Leadership Class Facility, there are five orders of magnitude difference between the aggregate computational bandwidth and the peak bandwidth to the parallel file system. The next generation supercomputer there, Summit, will have more than 5 times the amount of computation as Titan, but there will be *no improvement to the storage system*.

As the disparity between computation bandwidth and storage bandwidth grows, simulations are capable of writing out only smaller proportions of the data that are generated. Attempting to write too much information to disk storage could stall the simulation, possibly enough to prevent the simulation from making progress. Given a large enough disparity between computation bandwidth and storage bandwidth, a simulation may be incapable of writing enough data to properly analyze the results.

The *in situ* approach circumvents this bottleneck by removing the necessity of first storing data to persistent storage before processing. By running the visualization or analysis on the data as they are generated and while the simulation is still running, it is possible to have a much more thorough analysis by processing data that would otherwise be discarded.

All types of visualization and analysis can benefit from access to full spatio-temporal resolution data. However, it is particularly valuable for certain classes of operations, such as flow analysis and feature tracking, whose output fidelity increases with access to higher resolution input data.

Ultimately, *in situ* visualization can be used to maximize the information in data written to disk. Most visualization processes have a tendency to reduce the amount of data required to represent information. *In situ* implementations can also help provide more general data reduction capabilities such as compressing data and making better determinations of which regions should be captured at what times.

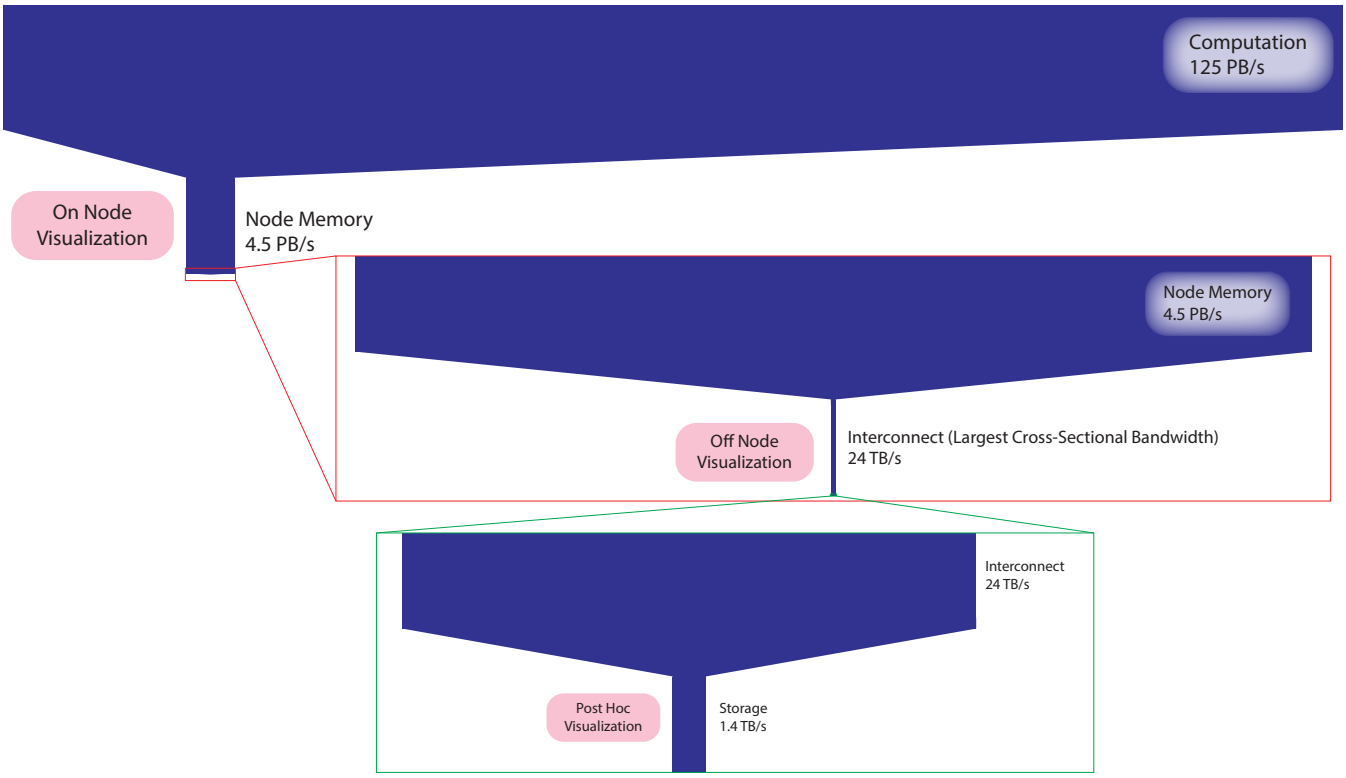

Figure 1: *A plot of the relative bandwidth of system components in the Titan supercomputer at the Oak Ridge Leadership Class Facility. The widths of the blue boxes are proportional to the bandwidth of the associated component. Multiple scales are shown to demonstrate the 5 orders of magnitude difference between the computational bandwidth and the storage bandwidth. Image source: Moreland.*

The *in situ* approach has the potential to reduce the overall cost of performing computational science. HPC systems dedicated to visualization can comprise a significant proportion of the overall facility [Chi07], so integrating visualization with simulation and running directly on the computational resources could make the most of facilities dollars. Making use of the same computational platform for both simulation and visualization/analysis activities reflects an economy of scale, where a center does not need separate platforms and additional support staff [BvRS*11].

In addition to these classic motivations for *in situ* visualization, the primary reason for the recent increase in interest comes from preparation for computational science on exascale computers. *In situ* visualization is considered a critical technology for achieving scientific discovery at exascale [ASM*11, Ahe12, Mor12, CGS*13, BDE13, Ahr15].

3. *In Situ* Methods

The term "*in situ* visualization" has evolved into an umbrella term to cover a variety of methods for processing. Recently, a group of approximately fifty visualization scientists convened to formalize the terminology for describing different *in situ* methods [ins16], also known as the "*In Situ* Terminology Project." This group currently is characterizing *in situ* methods using six axes: integration type, proximity, access, synchronization, operation controls, and output type. This section describes these axes, drawing from ideas and discussion from the participants of the *In Situ* Terminology Project.

3.1. Integration Type

Several different methods are used to integrate visualization capabilities into running simulations. There are many examples of simulation developers creating and embedding their own visualization routines as part of the simulation system. Such implementations tend to be lightweight but unsuitable for reuse elsewhere. For more universal reuse of *in situ* visualization capabilities, there exist general-purpose libraries intended to be used by simulations to incorporate visualization routines. These libraries allow visualization capabilities designed by one group to be directly integrated into a simulation of another group.

There are also indirect methods to integrate *in situ* visualization with a simulation. One such approach is to use a shared protocol to indirectly connect the two components. This approach can be realized through a middleware framework, such as ADIOS [LZKS09] or GLEAN [VHMP11] (both discussed further in § 5), where one or both of the components could be using the simulation data for purposes in addition to visualization. Another indirect integration

method is function interposition where functions already used in the simulation are replaced by functions that do *in situ* visualization processing. For example, the simulation's function to write data to disk can be replaced, unbeknownst to the simulation code, with an alternate function that intercepts the data for visualization purposes.

3.2. Proximity

The proximity between visualization routines and the simulation code can greatly affect performance. Enumerating all possibilities for proximity is difficult, especially in the face of emerging architectures and deep memory hierarchies. The closest proximity for *in situ* routines is to share the same cores as the simulation, but even this basic configuration is complicated when considering how data is moved through the cache. The furthest proximity for *in situ* would be to send data to faraway nodes, possibly even to distinct machines (and possibly even to another continent). This model, where data is moved between nodes, is sometimes referred to as in transit processing. Points along the close-to-far proximity spectrum include architectural features such as burst buffers, local file systems, dedicated connections (e.g., PCI between CPU and GPU, NVLink between GPUs), etc. Finally, it is important to note that visualization routines may run in multiple locations. A common example would be to run data triage routines on the same nodes as the simulation and also to run additional visualization routines on distinct nodes (that access data via a transport operation).

3.3. Access

An important description of an *in situ* system is its access to simulation data. With direct access, the visualization routine runs in the same logical memory space as the simulation code. In this case, the visualization routine typically gains access to data via pointers to simulation memory. With indirect access, the visualization routine runs in a distinct logical memory space separate from the simulation code. In this case, the visualization routine typically gains access to data via a communication mechanism that copies data from the logical memory space of the simulation.

Access is often conflated with proximity, because direct access occurs most often with on-node proximity, and indirect access occurs most often with off-node proximity. However, the remaining options are possible, although not common. Indirect access and on-node proximity occurs when visualization routines are run on the same nodes as the simulation, but using distinct memory resources (likely as a separate program running alongside the simulation). While this approach incurs extra overhead for accessing data, it enables clear separation between simulation and visualization, which can lead to simpler implementations. Direct access and off-node proximity can occur in PGAS-type settings. While this approach is feasible in some settings, it would likely be impractical. For example, if the simulation and visualization are located on distinct computing resources connected with a slow network, then algorithms that do random access within a data set are likely to perform slowly due to latency. That said, algorithms that can hide latency could still possibly perform well even in this setting.

3.4. Synchronization

Synchronization is about the relationship of "when" the visualization routines and the simulation code operate with respect to each other. With synchronous *in situ*, computing resources are devoted exclusively to the visualization routine or the simulation. In this model, the simulation and visualization routines trade off control of the computing resources, with only one executing at a time. With asynchronous *in situ*, the visualization routine occurs concurrently with the simulation. In this model, the simulation and visualization routines can execute at the same time. This sharing may occur by partitioning compute nodes between simulation and visualization, by sharing resources within a node for both activities, or by other models where the allocation of resources vary over time.

As mentioned in the discussion of Proximity, visualization routines may be occurring in multiples locations within a single *in situ* system. In this case, each routine may have its own synchronization. Revisiting the example from the previous section of an architecture that does data triage in close proximity and visualization routines from distant proximity, it would be common for the former to run synchronously (i.e., the simulation passes execution control to the triage routine, which passes execution control back to the simulation when finished) and the latter to run asynchronously (i.e., execute on data extracts after they arrive from the triage step).

3.5. Operation Controls

Operation controls describe whether the end user can modify which visualization operations can be performed during execution. One type of operation controls allows the end user to modify the visualization operations being performed while the simulation is executing. This is often referred to as "interactive" usage. Interactive controls often have further distinctions regarding whether the simulation data can be modified (i.e., "steering") or not. Another type of operation controls requires that the set of visualization operations to be performed be fixed before the simulation begins, i.e., they cannot be changed by the user during execution. This is often referred to as "batch" usage.

3.6. Output Type

Output type describes what the *in situ* visualization routines generate. While the output of the execution does not affect the design of the system per se, many participants of the *In Situ* Terminology Project felt that it was an important descriptor of the system.

Explorable outputs are outputs that are useful for *post hoc* exploration, while non-explorable outputs are outputs from visualization routines that are not useful for *post hoc* exploration. These two options are best seen as extremes of a spectrum. If the output of the simulation is static images, for example renderings of isosurfaces, then that would typically be described as non-explorable. That said, animating these images over time may enable *post hoc* exploration, so even this simple example is fuzzy. Further, the Cinema system [AJO*14b], which extracts images *in situ* for multiple visualizations, viewpoints, and time slices, and then enables *post hoc* exploration by providing an environment where users can explore data in a traditional manner (for example by animating images

from different viewpoints to fly around a data set), is an example of an approach which produces images and yet is clearly explorable. Other important examples of explorable extracts are those that compress fields, for example using wavelets, and those that extract key portions or aspects of the data (for example subsetting or topology).

4. *In Situ* History and Survey

While interest and research are rising presently for *in situ* methods, primarily due to the architectural motivations spelled out earlier (§ 2), there is a long and rich history of prior work in this space that dates back several decades.

The idea of generating images, or performing analysis, without first writing data to persistent storage, is as old as the field of computer graphics itself. In perhaps what may be the earliest known and documented example, Zajac, 1964 [Zaj64] computes the orbital path of two bodies and generates movie frames on-the-fly through a direct-to-film process. The NCAR Graphics Library [NCA], originating in the 1960s, may be some of the earliest production-quality "*in situ* infrastructure and methods", and continues to be developed and used by a world-wide community today. It consists of set of subroutine-callable methods for generating images/plots of scientific data. The NCAR Graphics Library has been widely utilized for both *in situ* and *post hoc* use cases.

Since the earliest of days in the history of *in situ*, there have been many diverse accomplishments that have explored many different dimensions of the space. The survey of the salient work will be organized along the following lines in ensuing subsections. First, we present work from the 1990s and early 2000s (§ 4.1), which focus primarily on coupling simulation codes with external infrastructures for *in situ* analysis and visualization, as well as highly specialized *in situ* applications and methods. Second, we examine the more recent large body of work (§ 4.2) that explores ways of overcoming the limitations of *in situ* such as: non-explorable versus explorable *in situ* data products; coarse versus fine temporal sampling; feature detection/tracking; guided simulation processing; and intelligent reduction of data saved to persistent storage for subsequent processing. In some cases, a given work has a contribution to more than one of these categories. Finally, we describe the significant amount of work focusing on topics related to *in situ* infrastructures (§ 4.3).

4.1. Early Work: 1990s-early 2000s

In a survey of methods and infrastructure of the *in situ* space, Heiland and Baker, 1998 [HB98], referred to this type of technology as "co-processing systems". That report focused on systems/methods that support interactive computation, or computational monitoring and steering. All the systems they surveyed have some visual data exploration and analysis dimension. Those systems served as the basis for a large body of work in the 1990s. A year later, Mulder, et al., 1999 [MvWvL99] performed a similar survey of computational steering environments that included several additional systems. They distinguished three computational steering use cases: model exploration, algorithm experimentation and performance optimization. They categorized each system based on these steering use cases, user interface and architecture.

Two particular examples cited in the Heiland and Baker report are pV3 and AVS. Haimes, 1995 [Hai95] couples a CFD code with the pV3 distributed visualization toolkit [Hai94], and describes the usefulness of being able to visually inspect calculations as they evolve on a MPP-class system. pV3 itself supports distributed memory operation, and uses an application interface design pattern that is very similar to contemporary *in situ* infrastructures (c.f., § 5.3). Bethel, et al., 1994 [BJH94] and Jacobsen, et al., 1995 [JBDGH95] couple a multi-phase flow code used for subsurface modeling with the AVS system [UFK*89] for the purpose of rapid configuration of initial conditions, namely the location of injection and production wells in a reservoir simulation. This implementation makes use of the AVS co-processing API to connect the visualization infrastructure to a simulation code.

Meanwhile, there were other independent research projects during this period that did not use any of these infrastructures, and instead focused on the core concept of producing images *in situ* without first writing data to storage. For example, Ma, 1995 [Ma95] describes a method for doing volume visualization concurrent with a parallel CFD solver on an Intel Paragon.

Along a similar vein, Globus, 1995 [Glo95] proposed a visualization software model where data extracts could be generated in the simulation run to reduce the size of unsteady CFD output. These extracts could then be further processed later in order to generate graphics. The model consisted of a database for organizing the various *in situ* and post-processed visualization products in order to enable rapid exploration of the results *post hoc*. In the analysis, he compared several extract outputs to the full data size and compute times for extraction compared to rendering for the results.

4.1.1. Integrated Computational Environments

As machines and software technology evolved, at times the distinction between *in situ* framework and computational framework have blurred. During this period, integrated computation or problem solving environments were widely explored. Related to *in situ* visualization, Tu et al. [TYRG*06] propose integrating the entire simulation workflow, including mesher, partitioner, solver, and visualization, in a single execution for the sake of efficiency. They implement this idea in a finite element simulation system for earthquake modeling named Hercules. SCIRun [Ins15,PJB97] is a problem solving environment developed at the University of Utah. It is designed for interactive construction and *in situ* steering of simulations.

The Cactus Code framework [GAL*03] consists of infrastructure for building codes (the "flesh") and then add-on components (the "thorns") that provide specific types of functionality. Cactus provides *in situ* visualization and analysis capability through this thorn mechanism. For example, one can view a Cactus-generated *in situ* visualization of simulation results while it is running by pointing a browser at a URL that is specific and unique to that simulation run.

Finally, specialized applications emerged that made use of custom code that focused on a specific activity. Bethel et al., 2000 [BTl*00] made use of a distributed architecture that used a combination of image-based and parallel rendering techniques to address the challenges of large-data visualization, and use this

methodology, which included *in situ* coupling with a binary black hole merger simulation built in the Cactus framework, to win the SC Network Bandwidth Challenge three years in a row [BS05]. That project used what we are referring to in this paper as a multi-stage in transit processing model.

4.2. Focused Work

In this subsection, we examine the more recent, large body of research and development that explores ways of overcoming the perceived limitations of *in situ* such as: non-explorable versus explorable *in situ* data products; coarse versus fine temporal sampling; feature detection/tracking; guided simulation processing; and intelligent reduction of data saved to persistent storage.

4.2.1. Explorable Extracts

With *in situ* visualization, the basic idea is that you produce images showing the results of some visualization application. But what if you want to change some of the visualization parameters, like viewpoint position, isocontouring level, or so forth?

One of the common concerns about *in situ* methods is that they produce results that are not "explorable." In other words, with traditional *post hoc* approaches, a visualization application would allow a user full, unconstrained navigation through the parameter space of potentially dozens of different visualization or analysis operations. That property is something that makes those *post hoc* applications so useful in terms of scientific discovery. To overcome that limitation, there has been a great deal of work in the past decade that focuses on different dimensions of the problem.

Chen et al., 2008 [CYB08] present the idea of computing a collection of imagery from a visualization application in a way that represents a discrete sampling of parameter settings in a visualization application, e.g., viewpoint, isocontour level, time step, and so forth, and then using a remote lightweight client to allow a user to explore this collection of pre-rendered images. Ye et al., 2013 [YMM13] focused on facilitating flow-field visualization in an *in situ* setting, where *post hoc* interactions focus on changing viewpoint, doing block cutaways, or changing the lighting or color transfer function. Finally, Ahrens et al., 2014 [AJO*14b] explore extracting many images and creating seamless animations using the Cinema system.

Others have looked at exploring enhanced images that can be used as input to create new renderings. Tikhonova et al., 2010 [TCM10] focused on isosurfaces, storing layers of images that could be explored, and referred to the approach as visualization by proxy. She later demonstrated this technique in an *in situ* setting [TYC*11]. Fernandes et al., 2015 [FBF*15] focused on creating volume renderings, by capturing regions of interest via volumetric depth images (VDI) to enable later exploration.

Other works have focused on extracts derived from topology. Duque et al., 2005 [DL05] extract isosurfaces into their XDB format for later exploration in their FieldView visualization tool. Biedert et al., 2015 [BG15] utilize contour trees to create compact image-based representations which enable explorative analysis and visualization, including analysis based on specific subsets of the contour tree's segmentations. Similarly, Ye et al., 2015 [YWM*15] extract features into depth maps to enable *post hoc* feature extraction and tracking.

Lastly, Agranovsky et al., 2014 [ACG*14] considered *in situ* reduction and *post hoc* exploration in the context of flow visualization. Rather than using the traditional Eulerian frame of reference, i.e., interpolating particle trajectories from vector fields, they considered a paradigm where Lagrangian basis particles were extracted *in situ* and new trajectories could be interpolated from the basis trajectories *post hoc*. This paradigm was shown to be faster, more accurate, and use less storage than the traditional approach, because their Lagrangian particles were able to benefit from the increased temporal resolution afforded by *in situ* processing.

4.2.2. Fine Spatiotemporal Sampling

In typical *post hoc* workflows, simulations typically perform data saves at relatively infrequent temporal intervals. This situation arises from the disparity between the ability to compute and save data. *In situ* methods offer the potential to perform visualization or analysis processing at a much higher temporal resolution than would otherwise be possible in a *post hoc* scenario.

Ellsworth, et al., 2006 [EGH*06] describe a concurrent visualization pipeline used in a production environment for use with a weather forecasting code, where shared memory buffers hold simulation output for use by visualization tools. Their concurrent processing pipeline enabled this team to achieve and 864-fold increase of temporal resolution in animations, aiding scientists in gaining new insights. Coupling custom *in situ* methods with a combustion simulation code running on a large parallel machine, Yu, et al., 2010 [YWG*10] enable scientists to see phenomena that are high frequency in nature, or that occur with temporal sparsity.

In a related vein, computational scientists may be forced to run their codes at reduced spatial resolution due to the large cost associated with performing full spatial resolution I/O. Rübel et al., 2016, [RLV*16], customize production *in situ* methods and infrastructure for guiding *in situ* domain-science focused query- and feature-based analytics to facilitate deeper understanding of phenomena in an ion accelerator simulation. This approach enables computational scientists to overcome the high cost of I/O and run their codes at higher model resolution than would otherwise be possible.

4.2.3. Feature Detection/Tracking

In Situ methods can be used to adequately detect and track and ultimately output features in a running simulation in order to significantly reduce data output. To this end, Bennett et al., 2012 [BAB*12] combine *in situ* and in transit processing using the ADIOS framework to compute merge trees for combustion simulations. A topological segmentation [BWT*11] accelerates computing statistics concerning burning regions and defines a reduced data representation that still supports analysis for multiple fuel consumption thresholds.

Additionally, Morozov and Weber, 2013 [MW13] describe an algorithm to compute a distributed representation of merge trees,

which describe the connectivity in scalar fields (connected components below a given thresholds), and write halo catalogs to storage for subsequent analysis. Here, halos are features of interest in cosmological simulations, and the size of halo catalogs is tiny in comparison to the size of simulation output.

4.2.4. Guided Simulation Processing

Simulation codes often have some form of intrinsic analysis that will help guide processing. For example, AMR-based codes [Ber] will often refine the spatial domain in regimes of "interesting physics." Using this information, simulations can be steered by leveraging *in situ* instrumentations.

Crivelli et al. 2004 [CKH*04] couple custom in transit methods and infrastructure with a parallel code that computes minimal-energy protein structure configurations with external visual data exploration and analysis tools that provide visual feedback on the solution progress, as well as to steer the simulation towards an optimal solution by pruning unrealistic or unpromising combinations of permutations. Here, the use of an in transit methodology helps to accelerate convergence of a computation for a problem that is $NP-$hard.

Biddiscombe et al. 2012 [BSO*12] developed ICARUS, a simplified *in situ* interface for codes that output results to HDF5. By changing the HDF5 function calls to ICARUS function calls with the same API they were able to send the simulation data to a ParaView server without further code modification. A ParaView plugin allowed for interactive simulation monitoring and steering. Rivi et al. 2012 [RCMS12] reviewed instrumenting two simulation codes with ICARUS and another with Libsim and presented some parallel performance analysis of ParaView and VisIt.

4.2.5. Intelligently Reduced-size Data Products

Some works consider a paradigm where *in situ* triage is used to prioritize data where only the most important aspects are saved. While the data reduction goal is also achieved through feature detection/tracking use cases, these works explore other methods for the intelligent reduction of data output.

First, Lehmann et al., 2014 [LJ14] consider both multi-resolution and temporal compression of data using a technique that can generate adaptively refined meshes. Nouanesengsy et al., 2014 [NWP*14] introduced a scheme around Analysis-Driven Refinement (ADR), also inspired by the principles behind adaptive mesh refinement. Fernandes et al., 2014 [FFSE14] extend volumetric depth images (VDI) for compression of simulation results for *post hoc* visualization. Their method takes advantage of space-time coherence of time-dependent simulation data to obtain *in situ* data reduction. Rübel et al., 2016 [RLV*16] compute and save histograms, statistics, and multivariate joint distribution functions from both 2D and 3D simulations to enable intermodel comparison and rapid *post hoc* exploration. Finally, Li et al., 2015 [LGP*15] considered the efficacy of wavelet compression, which prioritizes based on wavelet coefficients, with an eye toward *in situ*.

4.3. *In Situ* Infrastructure Projects

While later in section § 5 we provide details of four contemporary *in situ* infrastructures, this section provides a brief summary of other contemporary *in situ* infrastructure projects. For each of these projects, we give a short description and list their functionality based on the categorization of *in situ* methods presented earlier (§ 3). The categorization combined with the software availability are contained in Table 1.

There exist a large number of simulation codes that have *in situ* capabilities. A majority of these instrumentations have been focused on solving issues in their specific workflows and not developed with the goal for use with other codes. For that reason we omit their coverage here. In addition, we do not include libraries like VTK or NCAR Graphics in this section because they are general purpose tools that would require significant development in order to customize them for *in situ* use with a simulation code.

Cactus [GAL*03, cac] is a computational framework that has *in situ* capabilities. While Cactus is a framework, its design includes the ability to support legacy codes (`C`, `C++`, `Fortran 77` and `Fortran 90`) which allows it to be used as a way to instrument an existing simulation code with Cactus's *in situ* infrastructures. Cactus provides thorns, to output data to screen or output files. The remote screen output is done through a web-browser. The *in situ* operations include isosurfacing, downsampling and subsetting to reduce the data. Remote visualization is done through sending polygons for isosurfacing and 2D slides for the rest. The output file formats include images, ASCII and HDF files. In addition, steering of pre-defined parameters can be done through a web-browser as well.

CUMULVS [Koh], short for Collaborative User Migration, User Library for Visualization and Steering, contains infrastructure for visually monitoring and steering a simulation. The steering mechanism allows for multiple connected users to coordinate the simulation run progress through locking. Users can dynamically connect and disconnect from a CUMULVS instrumented simulation code run. Text and 2D viewers are included with CUMULVS for viewing outputs but no 3D viewer is included. CUMULVS has a `Fortran` and `C` interface and can operate on PVM or MPI interprocess communication libraries. It handles topologically structured grids and particle decompositions. CUMULVS does not support output of data extracts or images. The last released version of CUMULVS was version 1.3.0 alpha2 and is from 2003.

Damaris/Viz [DSP*13, dam] is an *in situ* framework based on the Damaris [Ker] middleware. It provides a simple way of connecting a simulation to *in situ* analysis and visualization tasks via an external, XML-based data description. Damaris/Viz can be used both synchronously and asynchronously. For asynchronous usage, the simulation's computing resources can be partitioned such that a subset of cores in a multicore node or a subset of the nodes of a job's allocation can be used to run *in situ* analysis and visualization tasks. While *in situ* tasks take the form of user-provided plugins written in `C++` or `Python`, Damaris/Viz also natively supports a connection to VisIt through the Libsim interface.

EPIC [DHH*15] is a toolkit for generating *in situ* surface extracts and then optionally computing a Proper Orthogonal Decom-

position reduced order model for *post hoc* analysis and visualization. It uses a master-slave MPI task hierarchy to manage the data from the other analysis and user application tasks. The simulation is required to use the EPIC defined MPI communicator.

EPSN [eps] is a library for computational steering and on-line visualization. The *in situ* visualization tool works in parallel and can disconnect and reconnect to the running simulation. A light-weight GUI is included for examining results *in situ*. It allows viewing and modifying the steered parameters and uses VTK and IceT [Ice] for viewing. A front-end API is also included for connecting with a high-level visualization tool. EPSN version 1.0.0 came out in 2009 and the last code change is from 2011.

***Free*processing** [FPS*14, fre] is a tool designed to reduce the barriers for *in situ* integration. Its fundamental motivation is that integrating *in situ* visualization routines into simulation codes is labor-intensive, which in turn reduces the number of codes that adopt the practice. Its approach is to use library interposition, including co-opting IO routines to insert *in situ* visualization into the simulation's workflow. The authors then furthered their goal of simplifying integration in separate work [FK15, vis] by observing a simulation to identify when visualization should occur as well as characterize memory organization for the purpose of creating interactive visualizations. This removed *Free*processing's underlying assumptions about the simulation code and its usage of I/O.

Nessie [LOKR11, LOK12], short for NEtwork Scalable Service InterfacE, is a framework for developing parallel, application-specific data services for HPC systems. These services include checkpointing, interactive visualization, network traffic analysis, and in transit analysis. Nessie uses a client-server architecture with asynchronous methods with heterogenous system support. Nessie is now part of Trilinos [Tri].

pV3 [Hai94, Hai], short for parallel Visual3, was developed at MIT and handles 3D unstructured grids. It has variants that can use PVM or MPI for inter-process communication. pV3 is designed as a library that can be linked to for *in situ* visualization and analysis and/or for post-processing results performing operations such as isosurfaces, slices and probing. It has a GUI client that connects to a parallel server through PVM. pV3's design allows the GUI to connect and disconnect from a running simulation to do *in situ* analysis and visualization and also allows computational steering. Additionally, extract output is also supported. The latest release of pV3 is revision 2.05 and as of 2000 is no longer under development.

QIso [ZABP15] is an *in situ* library for parallel generating of images of isosurfaces. It works with topologically regular grids and has been shown to scale to 92,160 MPI ranks.

SCIRun [Ins15, PJB97] is a problem solving environment developed at the University of Utah. It is designed for interactive construction, debugging, steering, and visualization of simulations. SCIRun's design is modular with the goal of having other modules added to compute the desired physics. Several physics modules are already included and *in situ* use is most easily done through building on top of the existing framework as opposed to use as a library. SCIRun is publicly available and is still under active development with a stable version 4.7 release in September 2014 and an alpha release of version 5 available.

Strawman [LBC*15] is a thin infrastructure designed to explore the *in situ* analysis and visualization needs of simulation code development teams planning for multi-physics calculations on exascale architectures. Another goal of Strawman is to serve as a community proxy for research into *in situ* techniques. It uses Conduit's [con] data model for describing uniform, rectilinear, and unstructured grids, EAVL for the analysis and visualization pipeline and IceT for the parallel compositing of the individual images. It can perform a zero-copy use of the simulation data structures if the formats match. Otherwise, a full copy of the data is required.

yt [TSO*11] is a Python-based scripted visualization system designed specifically for AMR data from astrophysics simulation although it can be applied to data from other science problems as well. yt is customarily used in file-based post-processing, but it also contains a library interface that allows it to be integrated with simulation codes. yt is publicly available and is still under active development with a stable release of version 3.2.3 in February 2016.

5. In Depth Analysis of Four *In Situ* Infrastructures

In situ analysis and visualization is still relatively new. Until recently, the field has been dominated by ad hoc, proof-of-concept prototypes initially concentrating on the monitoring and steering use case for simulations, and later transitioning to *in situ* focused work (as detailed in sections § 4.1 – § 4.3). However, several production quality infrastructures have emerged. In this section, we present three of the most widely deployed *in situ* infrastructures, ParaView Catalyst, Libsim, and ADIOS, to give a full idea of design considerations and general functionality available. To balance the methodologies presented in this section, we provide a deep dive into a promising research infrastructure, GLEAN.

5.1. ParaView Catalyst

ParaView Catalyst is an *in situ*, in transit, and hybrid workflow library [BGS15, FMT*11], with a malleable application programming interface (API), that orchestrates the delicate alliance between simulation and analysis or/and visualization processes. It exposes the renown capabilities of VTK [SML04] and ParaView [Aya15]. The analysis and visualization methods can be implemented in `C++` or `Python` and can run *in situ*, in transit, or a hybrid of the two methods. `Python` scripts can be crafted from scratch or using the ParaView GUI to interactively setup prototypes and export as Catalyst scripts.

The largest scale run to date used 256K MPI processes on Argonne National Laboratory's BlueGene/Q Mira machine [RSC*14]. The scaling studies utilized Parallel Hierarchic Adaptive Stabilized Transient Analysis (PHASTA), a highly scalable CFD code, developed by Kenneth Jansen at UC Boulder, for simulating active flow control on complex wing design (see Figure 2).

5.1.1. Methods

Catalyst supports ***In Situ*** workflows as it was designed to run synchronously with the simulation, where analysis methods and visualization pipelines are executed along side the simulation run, in the same address space.

Tool	Synchronization	Operation Controls	Integration Type	Availability
Cactus	Synchronous	Interactive	Framework	Public
CUMULVS	Synchronous	Interactive	Direct Integration	Public
Damaris/Viz	Both	All	Shared Protocol	Public
EPIC	Asynchronous	Batch	Direct Integration	Private
EPSN	Both	Interactive	Direct Integration	Public
*Free*processing	Synchronous	Batch	Interposition	Public
Nessie	Asynchronous	Batch	Shared Protocol	Public
pV3	Asynchronous	Interactive	Direct Integration	Public
QIso	Synchronous	Batch	Direct Integration	Private
SCIRun	Synchronous	Interactive	Framework	Public
Strawman	Synchronous	Batch	Direct Integration	Public
yt	Synchronous	Batch	Direct Integration	Public
ADIOS*	Both	Both	Shared Protocol, Direct Integration	Public
Catalyst*	Synchronous	All	Direct Integration	Public
GLEAN*	Both	Batch	Shared Protocol, Direct Integration, Interposition	Public
Libsim*	Synchronous	All	Direct Integration	Public

Table 1: *Categorization of in situ infrastructures. Note that ADIOS, Catalyst, GLEAN and Libsim are discussed in detail in § 5.*

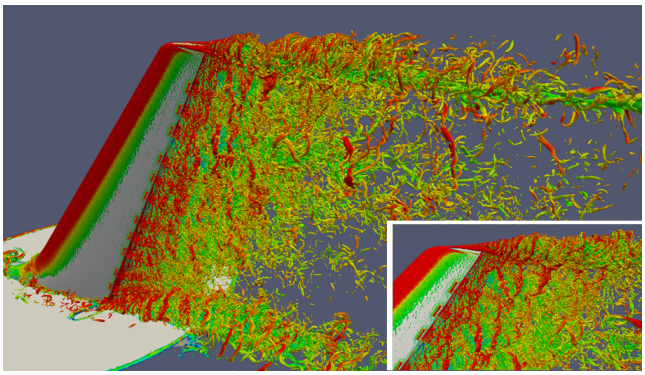

Figure 2: *Catalyst was utilized to contour two separate PHASTA quantities that are used for fluid flow computation and analysis, wall distance and Q-criterion, and generated Q-criterion contours colored by velocity magnitude. Inset shows a zoomed view of the wing tip. Image source: Rasquin & Jansen, UC Boulder*

In Transit workflows can be implemented using two sub-groups of a global MPI communicator: one for simulation processes and one for analysis and visualization processes. However, the data movement from the simulation processes is not automatic, and requires the writing of an additional communication routine during instrumentation.

Catalyst enables **Hybrid** workflow using VTK's I/O capabilities or by leveraging additional middleware such as Nessie [OMFR14]. For example, analysis methods and visualization pipelines could send intermediate results to burst buffers, and ParaView or another application would pull data from the burst buffers for interaction and/or further analysis.

In addition, Catalyst can connect to a separately running ParaView *Live* session for exploring results as they are being produced. The *Live* method can facilitate a **Monitoring/Steering** workflow. This, in turn, enables subtly unique **Analysis and Visualization Steering** workflows where the analysis methods and visualization pipelines are modified interactively through user feedback.

Finally, synchronous and asynchronous communication patterns are generally aligned with specific Catalyst workflows. *Live* supports both, and communications can be changed, as described above with hybrid workflows, utilizing third-party software.

5.1.2. Basic *In Situ* Integration

Instrumenting a simulation with Catalyst is as simple as implementing three routines: `Initialize`, `CoProcess`, and `Finalize`, but mapping the simulation's data structures to the VTK data model may require significant effort. However, by leveraging the flexible API, the impact on the simulation codebase can be minimized.

Catalyst needs to be initialized before the first invocation of the `CoProcess` call using `Initialize`. This initialization step includes setting up the *in situ* output and *Live* connection option. `CoProcess` is typically called for each time step as the simulation progresses mapping simulation data structures and performing the desired analysis or/and visualization. `Finalize` is called to clean up Catalyst state including releasing any allocated memory.

Once implemented, end-users have access to the flexible analysis and visualization capabilities associated with the ParaView application *in situ*.

5.1.3. Products

Catalyst scripts provide access to a wide range of analysis methods and/or visualization pipelines. These scripts may produce images, statistical quantities, plots, derived information (such as polygonal data geometry), and reduced or full data results. Recent work added support for exporting explorable images or Cinema [AJO*14b] databases. These image databases can be post-processed using light weight applications. Other efforts are adding support to serialize analysis results using high performance I/O libraries like ADIOS [LZKS09].

5.1.4. Editions

For systems with limited memory, one common concern is the size of code that Catalyst (through ParaView and its dependencies) brings in. Eliminating components that are not used during the analysis and/or visualization saves on the system memory used. Reduced-size versions of the Catalyst library are called Catalyst *editions*. Using a `Python` script included in the ParaView or VTK source code, combined with JSON files describing the classes to include, customized Catalyst editions can be produced. Several editions are pre-configured in the ParaView source code. These pre-configured editions have different levels of common analysis and visualization functionalities included. The memory impact on the simulation was closely examined in [FMM*14] for several Catalyst editions, build configurations and per node run concurrency. Table 2 gives the increase in executable size for several of the pre-configured editions using static Catalyst builds.

Name	Size
Flyweight	2.1 MB
Base	16 MB
Base-Python	23 MB
Essentials	18 MB
Extras	23 MB
Essentials+Extras+Rendering	32 MB
Essentials+Extras+Rendering-Python	44 MB
Full ParaView	98 MB

Table 2: *Size impact of common Catalyst editions on executable.*

5.2. Libsim

Libsim [WFM11, CMY*12] is an *in situ* library that enables data analysis and visualization to be performed on simulation data using the full complement of tools available in the VisIt software. Libsim is fairly low level and can be adapted to a variety of applications from batch-style simulations to interactive programs. Libsim's more recent niche has been the generation of engineering extract databases, or smaller feature based subsets of the simulated data that provide enough information for useful post-processing of the data, albeit on a much reduced version of the data. Libsim has been used to produce extract databases in FieldView XDB [WFL16] format using simulations such as OVERFLOW2, CREATE-AV Kestrel, and AVF-LESLIE (see Figure 3) at overheads of between 2-3 percent of the overall solver runtime, depending on the solver and requested extracts. The runs for AVF-LESLIE at 62K compute cores are currently the high water mark for Libsim, though VisIt's compute server, which forms the basis of the Libsim runtime library has run up to 98K cores.

5.2.1. Methods

Libsim was conceived as a mechanism to allow users interactively running the VisIt graphical user interface to connect to running simulations so that the simulation's data could be explored and monitored, offering a powerful debugging capability to simulation code developers. The VisIt GUI can send commands to the simulation to tell it which data to obtain, process *in situ*, and return to the VisIt viewer for display. This enables users to interactively apply the full set of VisIt features to *in situ* data analysis and to also perform some simulation steering via custom simulation user interfaces. All Libsim data processing is done in the simulation's address space, using pointers to data stored in the simulation's data structures. The need for interactive visualization of the running simulation has been overshadowed by a need to automatically create data products as the simulation runs, for offline analysis when the job completes. Libsim has adapted to this use case by providing a simpler-to-use batch mode that adds new functions for programmatically setting up the VisIt plots that form the foundation of the generated image and extract data products.

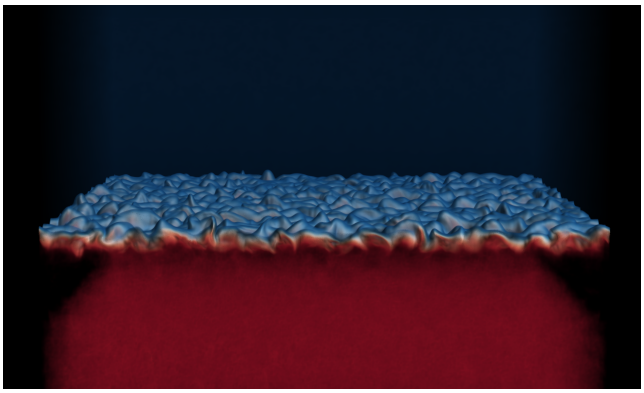

Figure 3: *Visualization of flame front from AVF-LESLIE. Image source: Whitlock.*

5.2.2. Basic *In Situ* Integration

Libsim itself consists of two pieces: a control library and a runtime library. Solvers normally link only to the control library, which is a lightweight library that dynamically loads the Libsim runtime library. The separation of the front end and runtime libraries means that typical simulations instrumented with Libsim do not grow appreciably or use additional memory until *in situ* operations are requested. To run at the highest levels of concurrency, it is possible to statically link the Libsim libraries and their dependencies into the executable, eliminating the added cost of loading shared libraries.

The typical batch-mode solver makes a few Libsim calls to set up the environment needed to locate and load the Libsim runtime library. Once the runtime has been loaded, Libsim functions for setting up plots, saving images, or saving data extracts can be added to the code. Data for these VisIt operations are obtained via an adaptor layer, which consists of `C/C++`, `Fortran` or `Python` functions written by the simulation developer. The adaptor functions are invoked by the Libsim runtime library on demand when certain data are needed and their role is to interface the simulation's data structures with Libsim. Typically, adaptor functions will create Libsim objects by making calls to the Libsim library and then store pointers to simulation data arrays in those Libsim objects, which are ultimately returned to VisIt. VisIt uses the data stored in the Libsim objects to create VTK objects whose data points directly to the simulation data, enabling most data to be passed in a zero-copy fashion.

5.2.3. Products

Libsim builds on VisIt's notion of plots, which are visual representations of data. Libsim provides functions for creating plots directly or setting up more complex visualizations with multiple plots via VisIt session files, which are XML files that describe all of the attributes of a VisIt visualization. Once plots have been created, Libsim provides functions for saving sets of image files or exporting geometric plot data to visualization output formats such as VTK, or FieldView XDB, among others.

5.3. ADIOS

ADIOS [LLT*14, LKS*08] is a high-performance I/O middleware library that combines fast synchronous I/O to storage, with advanced capabilities for *in situ* and in transit processing. ADIOS uses a high-level description of data to guide its data processing pipeline. The data is described either through an external XML file, or through a comprehensive descriptive API. The structural and semantic information thus produced is maintained alongside the stored data. ADIOS provides data access APIs that can utilize *a priori* knowledge of data structure to read the data, as well as APIs that enable dynamic discovery of the data's structural and semantic information. This enables a class of generalized data processing components for analysis and visualization workflows. Applications have, in the past, utilized platform agnostic interfaces such as `Fortran` native I/O, MPI-IO, etc, to address their needs; but there is a performance penalty due to the over generalization of the interface. ADIOS obviates this trade-off by introducing a platform agnostic API for the application, and abstracting the architecture specific I/O techniques to separate methods, thus providing both a low maintenance interface and a high performance mechanism for I/O. ADIOS has seen significant success in addressing the I/O needs for many leadership class applications at the DOE Leadership Class Facilities. For example, the XGC fusion simulation code that runs at full scale on Titan uses ADIOS for data output and leverages the *in situ* capabilities of ADIOS for online analysis and visualization.

5.3.1. Methods

ADIOS is designed as a componentized application library, where one of multiple available data transports can be selected at runtime. An important benefit of this approach is that once applications utilize ADIOS for I/O, *in situ* and in transit workflows can be instantiated without requiring any further modifications to the application code. Transports such as DataSpaces [DPK12, DZJ*14], FlexPath [DCE*13, ZCD*11] and ICEE [CWW*13] all follow the same principle - data is buffered in memory at the application node, additional *in situ* processing can be applied to this buffer, and the processed data is moved to auxiliary nodes for an in transit workflow.

ADIOS supports execution of data processing actions *in situ* [ZZC*13] (data is buffered and processed in local memory), in transit [ZAD*10] (data is buffered and processed in remote memory), through post processing (data is accessed from storage) and a combination of all three [BAB*12]. Wide area networks are similarly supported through a specialized transport.

5.3.2. Basic Integration

Adding *in situ* and in transit operators to applications that already utilize ADIOS for I/O is a trivial operation - instead of using a to-disk transport, the user can modify a single file to utilize a memory/network transport. The basic ADIOS interface comprises of two distinct parts for data output. First, the application provides a description of the data, including the extents of arrays, the decomposition of data within the parallel cohort and the specific transport to be utilized. This information can further embed a visualization schema to enable general visualization operations without customization for a particular application or use case. Second, a POSIX-write like API is provided for actual data output from the application. In a coupled scenario, the data is made available to the consumer at the end of the close operation.

The actual *in situ* operators also utilize the ADIOS read API for access to data. The ADIOS read API was designed to enable differentiated access to data through programmatic querying and selection functionality. This enables visualization and analytic components utilizing ADIOS for data access to easily convert to *in situ* operation.

The read API also provides introspection support for the data structure, allowing applications to discover variables at runtime. The read API differs from standard reading APIs by providing an explicit mechanism for scheduling and batching read operations. Individual operations can be chained to form a complete end-to-end workflow, with each operator reading data with ADIOS, processing data using internal logic, and outputting data with ADIOS.

5.4. GLEAN

GLEAN [VHMP11] is a flexible and extensible framework that takes application, analysis, and system characteristics into account to facilitate simulation-time data analysis and I/O acceleration. The GLEAN infrastructure hides significant details from the end user, while at the same time providing a flexible interface to the fastest path for their data and analysis needs and, in the end, scientific insight. It provides an infrastructure for accelerating I/O, interfacing to running simulations for in transit analysis, and/or an interface for *in situ* analysis with zero or minimal modifications to the existing application code base.

5.4.1. Methods

GLEAN supports **In Transit** workflows wherein the application data is moved from the simulation nodes to a set of dedicated nodes to be processed. Next, to analyze the data on the staging nodes, there are two possible ways: a) Use of GLEAN as a library by existing packages, such as ParaView, or, b) executed as an application wherein one can embed custom analysis and visualization kernels to process the data.

GLEAN also supports the ***In Situ*** workflow modality wherein it is embedded as part of the simulation and shares the same address space as the simulation, shares the resources, and is executed when desired by the simulation.

5.4.2. Basic Integration

Figure 4 provides an overview of the GLEAN infrastructure and compares the traditional mechanism used for I/O with GLEAN. The simulation running on the compute nodes may invoke GLEAN directly or transparently through a standard I/O library such as Parallel-netCDF [LLC*03] and HDF5 [HDF]. The data is moved out either directly to storage or to dedicated analysis/staging nodes. Using GLEAN, one can apply custom analyses to the data on the compute resource or on the staging nodes. This can help reduce the amount of data written out to storage. On the staging nodes, GLEAN uses MPI-IO or higher level I/O libraries to write the data out asynchronously to storage.

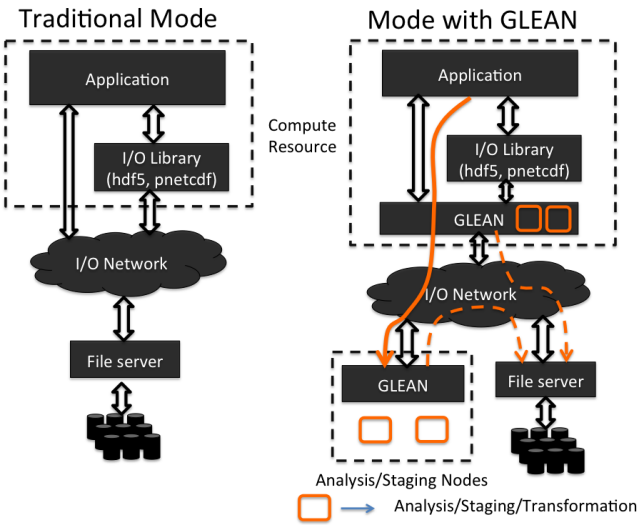

Figure 4: *Relationships between GLEAN and principal components of an HPC application. Image source: Vishwanath et al.,2014 [VBHP14]*

GLEAN is implemented in C++ leveraging MPI and pthreads, and provides interfaces for `Fortran` and C-based parallel applications. It offers a flexible and extensible API that can be customized to meet the needs of the application. It has currently scaled to 768K cores of the Mira IBM Blue Gene/Q supercomputer.

6. *In Situ* Applications

From earlier sections (§ 2 and § 5), *in situ* analysis and visualization has been actively used for more than a quarter of a century to solve science and engineering problems, and hundreds of advanced modeling and simulation codes have been instrumented with *in situ* infrastructures. In the following subsections, we leverage the reader's, now, in depth understanding of the *in situ* infrastructures highlighted previously in section § 5, and present several *in situ* use cases demonstrating the impacts of *in situ* analysis and visualization on critical science and engineering workflows instrumented with either ParaView Catalyst, VisIt Libsim, ADIOS, or GLEAN. In addition, we present Cinema explorable features application results that have excited the *in situ* analysis and visualization research community as well as domain scientists and engineers.

6.1. Using ParaView Catalyst to Analyze Nuclear Reactor Physical Phenomena

The Consortium for Advanced Simulation of Light Water Reactors (CASL) is the first United States Energy Innovation Hub established in 2010 by the Department of Energy. CASL connects fundamental research and technology development through an integrated partnership of government, academia, and industry that extends across the nuclear energy enterprise. CASL was established to provide leading edge modeling and simulation (M&S) capability to improve the performance of currently operating light water reactors (LWR).

To this end, Hydra–TH was developed, for CASL, to create a computational capability that enables the simulation of the thermalhydraulics processes inside a nuclear reactor at unprecedented fidelity. These simulations can use tens of thousands of compute cores on the largest supercomputers in the world and enable the detailed resolution of turbulent flow fields and their interaction with the reactor fuel assembly.

```
catalyst
  script hydrasinglepin.py
  node vel
  elem div
  node lambda
  node pressure
  node helicity
  elem density
end
```

Figure 5: *Example input deck option for Hydra–TH input deck. Image source: Bauer.*

The Hydra–TH developers worked to ensure that the Catalyst output mirrors what is available in the normal ExodusII full data dump output. Catalyst *in situ* analysis and visualization is simply requested within the Hydra–TH input file as depicted in Figure 5. In this example input deck snippet, the nuclear scientist or engineer identifies the *in situ* analysis and visualization script, `hydrasinglepin.py`, along with several derived quantities to be provided to Catalyst for processing. These quantities correspond to Hydra–TH's built-in *in situ* capabilities for computing derived quantities of interest on the fly. With this design, the nuclear scientist or engineer can request specific field information to be output similar to the input deck format for writing out dump files. The Catalyst output can be node, element, and/or sideset quantities. The `Python` script is then used to perform the individual analysis and visualization pipelines.

To increase the performance of current LWRs, CASL and the nuclear energy enterprise are utilizing Hydra–TH and Catalyst *in situ* analysis and visualization to investigate two performance problems [OCJ*15]: grid-to-rod fretting (GTRF) and lower plenum flow anomaly (LPFA).

The GTRF problem in pressurized water reactors (PWR) is a flow-induced vibration problem that results in wear and eventually failure of the rods in nuclear fuel assemblies. Currently, it has not

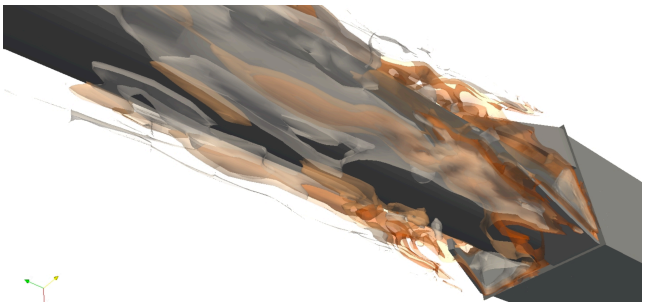

Figure 6: *In situ visualization of the GTRF problem: A single nuclear reactor rod, one of three spacers along the rod, and the mixing vanes (all in grey) with pressure isosurfaces in orange to cream colormap. Image source: O'Leary.*

been possible to completely characterize the flow-induced fluid-structure interaction (FSI) problem for the GTRF problem. Indeed, given the incompressible nature of the coolant, the relatively high Reynolds number, and the flexible character of the fuel rods and spacers, the FSI problem at the reactor core scale is daunting.

Hydra–TH is run using a Smagorinsky subgrid-scale model turbulence with Smagorinsky model constant of 0.18 for a representative single rod with three spacers. Hydra–TH is used to compute the time-accurate and fully three-dimensional flow field for the single rod, while Catalyst is used for *in situ* data reduction (a clipped 1/3 meter of the 9 meters simulated) to focus the analysis on the flow past one spacer. The reduced data sets are post-processed using ParaView (see Figure 6).

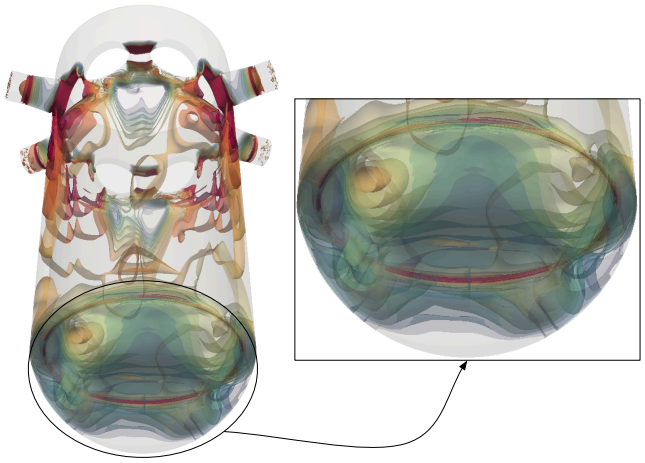

Figure 7: *In situ visualization of the LPFA problem: Simulating reactor flow in the lower-plenum during pump startup sequence with Hydra–TH. We see pressure contours in the lower plenum at startup that indicate transient behavior possibly associated with the lower-plenum flow anomaly. Image source: O'Leary.*

The LPFA problem is a known reactor flow anomaly. Hydra–TH simulations are used to develop a better understanding of the sensitivity of the flow distribution to differential inlet flow.

In this case, Hydra–TH is run using a Spalart-Allmaras model turbulence on a representative reactor vessel. Hydra–TH simulates reactor flow in the lower plenum during the pump startup sequence with an objective of identifying actions to reduce (or at least control) the variation. Catalyst creates explorable artifacts utilizing ParaView Web (a web-based interface for ParaView) for ensemble comparison analysis and visualization (see Figure 7).

The CASL M&S coupled with these *in situ* analysis and visualization capabilities assist in producing safer and more productive commercial nuclear power production.

6.2. Using Libsim To Create Scalable Extracts for Rotor-craft Simulation

Simulation of rotor-craft during forward flight and hover requires many CFD solver time steps to simulate seconds of actual flight time and accurately capture the movement of rotors and associated unsteady airflow structures.

Rotor-craft engineering applications may use hybrid or overset grids consisting of tens of millions of unstructured nodes for vehicle geometries and hundreds of millions of nodes for Cartesian, adaptive grids that capture off-body airflow. While these numbers are smaller than some science domains, the high number of solver iterations involved has the potential to create large amounts of volume-based simulation output.

Typical engineering culture tends to favor keeping volume data files for post-processing, but this practice is not sustainable for large, complex runs. *In situ* analysis and visualization methods, which create image-based and statistical data products, have not been adopted quickly. There is a concern these techniques may fail to capture enough information about the simulation to answer questions that may arise later in the analysis process.

A compromise could be *in situ* extraction of geometric features with associated scalar and vector fields that can be saved and explored later. Identification, simplification, and data extraction processes are well-suited to *in situ* technologies, and enable important structures to be saved at high frequency without writing impractical amounts of volume data. These *in situ* extracts allow engineers to analyze and/or integrate quantities and perform explorable visualization.

CREATE-AV Kestrel [FLPS15] is a CFD solver used for simulation of both fixed-wing and rotor-based aircraft. Kestrel has been coupled to the US Navy's CASTLE flight simulator to use CFD to more realistically simulate a UH60 helicopter landing on the back of a moving ship (see Figure 8).

The turbulence introduced by airflow across irregular structures on the moving ship adds to instabilities that must be accounted for to enhance realism for the flight simulator. In order to visualize the effects of the ship air-wake on the landing UH60, Kestrel integrated Libsim to produce surface-based extract files that could be used for later offline analysis and visualization. The extracts in this case were saved every 5 solver time steps to FieldView XDB format, which permitted later exploration of the much smaller extract files in FieldView. The size of the extract data per saved iteration was about 5 percent of the volume data's size, while taking as little as 2-3 percent of the overall solver runtime to generate. The production of XDB extract files from Kestrel by way of Libsim has generated a great amount interest among its user-base. In response, Kestrel's

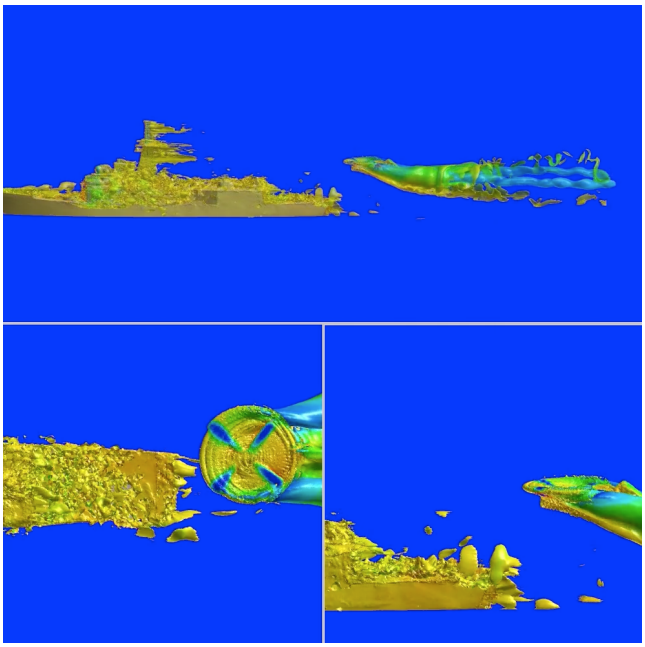

Figure 8: *Coupled ship air-wake visualization - isosurface of vorticity colored by pressure. Image source: Forsythe et al. [FLPS15].*

in situ capabilities are being expanded to include additional data products in Kestrel 7.

6.3. Using ADIOS to Visualize and Analyze a Multi Scale Fusion Edge Model

XGC is a suite of codes to study plasma edge physics in magnetic fusion devices; and is one the largest science simulations running on Leadership Computing Facilities (LCFs) in the USA. Numerical models used at the Center for Edge Physics Simulation (EPSi) and in simulation codes have been enhanced in recent years. These advances have led to greater data generation capabilities, which currently exceed the file system and disk-based storage capacities of current LCF. The volume, velocity, and variety of output data pose tremendous challenges in storing the datasets and carrying out post-simulation analysis. In order to keep pace with data volumes and velocities within current storage limits, output from an EPSi simulation needs to be processed *in situ* and in transit via complicated workflows that result in meaningful data reduction (before being written to storage) and for scientific data analysis and visualization.

The ADIOS framework has been used to develop support for such challenging workflow scenarios in EPSi. The focus is centered on providing transparent support of workflow management and execution engines to EPSi. Figure 9 shows one of the challenging EPSi workflows, called a *coupling* workflow, and how ADIOS provides an integrated environment. In the coupled execution, two different EPSi codes, XGC1 and XGCa, need to be tightly coordinated for data sharing during their concurrent executions, while data reduction-and-prolongation and post-analysis routines should be performed in a timely manner to reduce any I/O related overheads and minimize the time-to-solution in HPC environments. ADIOS has further extended the support to take into account and

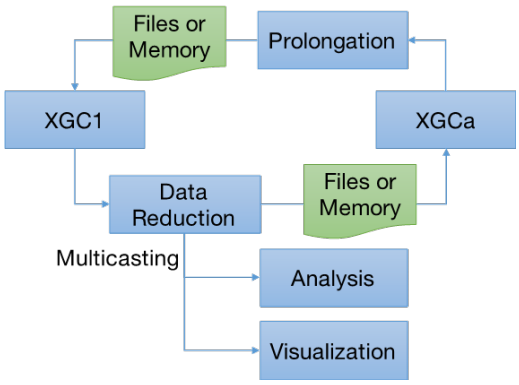

Figure 9: *EPSi XGC1-XGCa coupling workflow. Each separate code is executed as independent executables, and exchange information which is either reduced or expanded when they are coupled, and use the DataSpace method for ADIOS to exchange the data. Image source: J. Choi ORNL.*

leverage multi-level data storage hierarchies and high performance network interconnections as well as enable data reduction, compression, or indexing, in order to efficiently move data between tasks during execution.

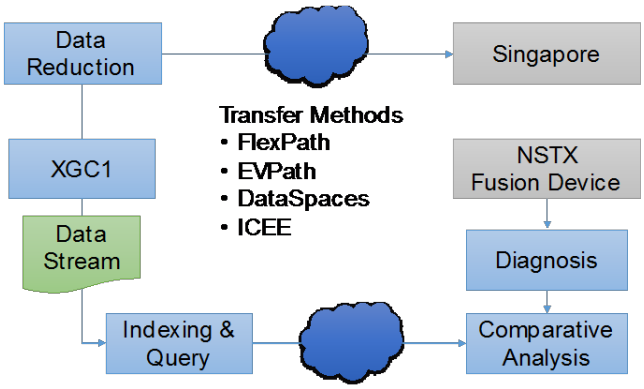

Figure 10: *Fusion comparative study workflow. The data is exchanged using the ADIOS ICEE method, and transferred from the A*STAR supercomputing center in Singapore over the WAN to Georgia Tech, where the data is processed using multiple applications with the ADIOS API. Image source: J. Choi, 2016 ORNL.*

Another challenge of data management in EPSi is in supporting workflows to perform comparative studies with remote experimental data transferred over wide-area networks in near real-time (NRT) fashion. Fusion experiments not only provide critical information to validate and refine simulations that model complex physical processes in the fusion reactor, but also drive simulations to quickly decide operational parameters for the next runs in between experiments.

Figure 10 shows an EPSi workflow used to process particle data during the simulation in order to observe if particles were following field lines. In this workflow, EPSi's XGC1 outputs data through ADIOS. ADIOS streams data over wide-area networks to a remote site, to be analyzed in a near real-time way. ADIOS's modularized transport methods (EVPath, FlexPath, and DataSpaces) enables sci-

entists to launch remote analysis in an efficient way. ADIOS's integrated *in situ* indexing and query module can be used to reduce the payload to meet the NRT requirement. Figure 11 shows the screenshot from the XGC scientist as they were trying to understand how the feature flowed around the torus.

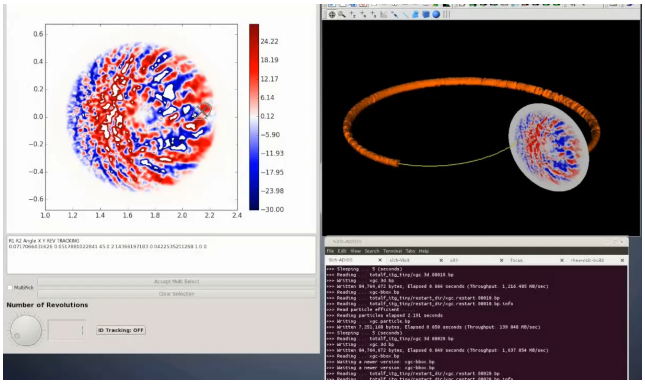

Figure 11: *On the left is an interactive image to select a feature of interest from the simulation. The data for this image is sent over the WAN by the ADIOS ICEE method. On the upper right is a 3D image of the plane along with the calculated trajectory. The simulation feature of interest is shown as a circular isosurface. The relevant data needed to visualize the feature of interest is extracted from the simulation on the supercomputer using ADIOS's query interface and then transferred to a remote visualization cluster for visualization and display. Image source: D. Pugmire et al. 2015 ORNL.*

6.4. Using GLEAN for Scalable Analysis for a FLASH simulation

FLASH multi-physics multi-scale simulation code [FOR*00] is an adaptive mesh, parallel hydrodynamics code developed to simulate high energy density physics and astrophysical thermonuclear flashes in two or three dimensions, such as Type Ia supernovae, Type I X-ray bursts, and classical novae. It solves the compressible Euler equations on a block-structured adaptive mesh. FLASH provides an Adaptive Mesh Refinement (AMR) grid using a modified version of the PARAMESH package [MOM*00] and a Uniform Grid (UG) to store Eulerian data. The Sedov explosion test problem is included in the FLASH simulation distribution. The Sedov explosion problem involves the self-similar evolution of a cylindrical or spherical blast wave from a delta-function initial pressure perturbation in an otherwise homogeneous medium [FLA].

To facilitate *in situ* analysis for FLASH with GLEAN, we designed a new dataset subclass in GLEAN to capture the data semantics of FLASH including the AMR hierarchy. Figure 12 depicts the AMR hierarchy of a FLASH simulation. For I/O, the FLASH simulation uses the pNetCDF API and format [LDL*]. To interface with FLASH in a non-intrusive way, we map relevant pNetCDF calls into appropriate GLEAN API calls. Next, when the simulation performs an I/O call, it also invokes the analytics embedded in the GLEAN framework in the I/O path. Thus, we are able to integrate with FLASH without modification to the FLASH simulation code.

Two *in situ* analyses to compute the fractal dimension and the vorticity were performed on the Sedov explosion problem. For FLASH, the fractal dimension helps illustrate the degree of turbulence in a particular time step as well as identify the variation of turbulence across sub regions in the domain and is very communication intensive. The vorticity calculation involves a more compute and memory intensive operation. The results were evaluated the efficacy of invoking the vorticity analysis *in situ* wherein it is embedded in the I/O stack with GLEAN as well as in an in transit mode wherein the data is moved to dedicated staging nodes for analysis.

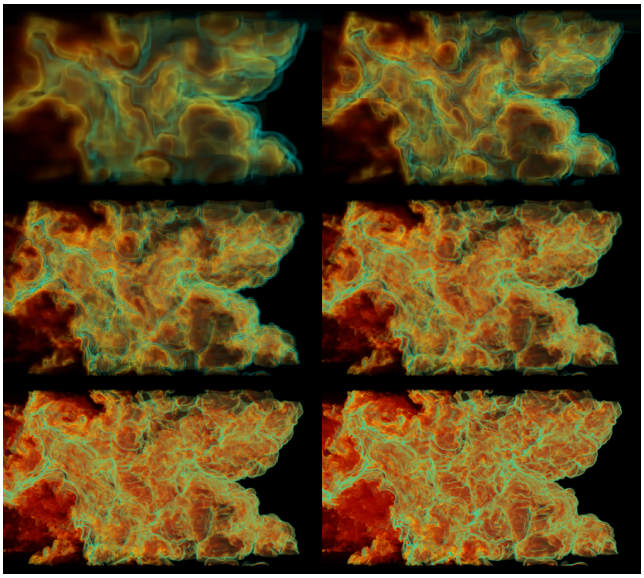

Figure 12: *Visualization of a FLASH Rayleigh-Taylor Flame simulation and the effect of additional AMR levels. The upper left image shows just the coarsest refinement level. The number of refinement levels progresses to the right and down until all six levels are shown in the lower right image. The variable being visualized is an analytical estimate of the flame front. The green-blue color highlights the flame surface, and the yellow-orange-red transition shows the fuel-ash mixture, which the flame front leaves behind.Image source: Nick Leaf et al. [LVI*13].*

A FLASH simulation could need a large memory footprint on each node depending on the science being simulated and studied. Depending on the simulation resource requirements and available analysis resources available, there exists a tradeoff between *in situ* and in transit analysis. A significant future challenge is to better orchestrate and schedule *in situ* analyses together with the simulation while taking into account the time and memory requirements of the analyses, the importance of the analyses, and the system parameters such as the computation time, I/O bandwidth, and maximum available memory to decide the optimal frequencies of the *in situ* analyses [MVM*15].

6.5. Using Cinema to Create Explorable Features from a Climate Modeling Simulation

The Accelerated Climate Modeling for Energy (ACME) project is developing and applying the most complete, leading-edge climate and Earth system models to critical, challenging and demanding climate-change research imperatives. For this community analysis

and visualization are typically done using a post-processing path through the simulator code called *analysis mode*. Typically, these analysis mode runs utilize an order of magnitude fewer MPI processes, and are executed several times with different analysis and visualization objectives. Analyzing and visualizing simulation data *in situ* in analysis mode share the same benefits that make *in situ* so attractive for simulation mode.

In general, *in situ* approaches operate on a predefined set of analyses and visualizations. Scientists need *in situ* analysis and visualization to: first, preserve important elements of the simulations, second, significantly reduce the data needed to preserve these elements, and third, offer as much flexibility as possible for post-processing exploration. Given the current views on climate change, this community has a policy to save the full simulation data results regardless of the I/O constraints. Hence, both the first and second scientific needs are satisfied by these project guidelines.

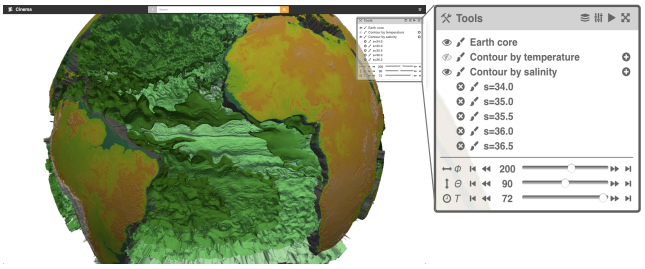

Figure 13: *(a) MPAS-Ocean simulation indicating contours that represent the locations of water masses within the ocean of salinity colored by temperature. (b) A user interface depicting the visualization pipeline for analysis and visualization objects (Earth core, temperature contours, and salinity contours) compositing. Image source: O'Leary.*

One of the ACME codes is the Model for Prediction Across Scales (MPAS) Ocean simulator [RPH*13]. The Data Sciences at Scale group at Los Alamos National Laboratory developed the highly interactive, image-based *in situ* analysis and visualization framework, Cinema [AJO*14b, AJO*14a, OAJ*15], to address the third requirement, namely interactive feature exploration. This exploration – so important to scientific discovery – is supported intuitively and effectively with Cinema.

High-resolution global ocean simulation with realistic topography, currently use 1.8 million horizontal 15 km sized grid cells with 40 vertical levels. In simulation results, contours of temperature and salinity indicate the locations of water masses within the ocean. Water masses, with names like North Atlantic Deep Water and Antarctic Bottom Water, occur within specific ranges of temperature and salinity. Visualizations of water masses allow oceanographers to view their pathways and extents, and compare them to observed climatology. Meandering ocean jets and eddies may be visible as perturbations to these Cinema visualization objects. Using these techniques helps scientists to determine if the simulated ocean currents and eddy activity compares well with observations.

These simulations are typically run on approximately ten thousand processors to achieve two simulated years per wall clock day. At this resolution, file output sizes present difficulties for traditional

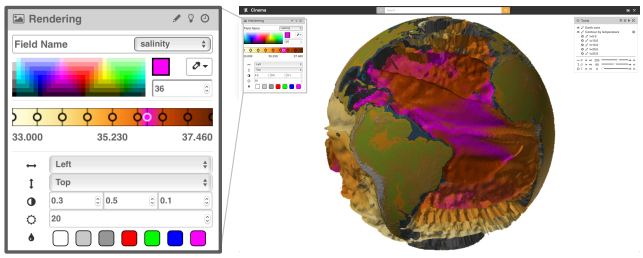

Figure 14: *(a) The dynamic rendering offers the capability to edit color lookup tables, set the light direction and color, and adjust material properties. (b) The fully interactive feature exploration, enabled by the Cinema, pulling resulting Cinema images and analysis products from the database. Image source: O'Leary.*

interactive post-processing analysis and visualization workflows. During analysis mode, ParaView Catalyst is used to output a Cinema database from the saved outputs. Then interactive visualization and analysis is supported via Cinema instead of using a traditional post-processing approach. *In situ* analysis and visualization integrated in analysis mode has been shown to adequately address the needs of the community.

7. Conclusion and Future Work

The phrase "*in situ* processing" is an umbrella term that has come to refer to a set of activities in which visualization and analysis processing happens without the need to write scientific data first to persistent storage. This concept is not new, it has been around for decades. Work in this space, which includes fundamental methods and production-quality infrastructures, has evolved to track changes in computational platforms and computing environments, to overcome the fundamantal tension of *a priori* selection of visualization or analysis algorithmic parameters with the needs for exploratory use, and to respond to the specific needs of the scientific community.

Going forward, many of the challenges facing *in situ* methods and infrastructures that have driven prior work will likely continue to persist, although these challenges will evolve as the computational landscape changes. Many of the challenges in the *in situ* space called out in earlier reports [Ma09] persist, even though the specifics have changed somewhat. For example, a number of related challenges will likely entail a concerted, collaborative effort with others, such as researchers and developers in the areas of operating systems and runtime, programming languages, system architecture, and so forth. These challenges include topics like how to share computational resources with simulations: cores, memory, and so forth. Another challenge centers around the idea of making *in situ* methods easy to use, so that a third party, like a code team, could download and use *in situ* methods and infrastructure with ease, and without the necessity of having an expert.

Beyond the future work that focuses on challenges related to changing computational platforms, usability and software engineering, future work in the *in situ* space will likely continue in several key directions. One area for future work is the notion of *in situ*

computation of extracts and derived data products that are suitable for subsequent exploratory use and for quantitative analysis.

Another is the notion of *in situ* algorithms that require persistent state information. For example, temporal analysis requires multiple timesteps of simulation data for processing. Examples of such algorithms include proper orthogonal decomposition (POD), autocorrelation and/or other temporal statistics. Retaining potentially multiple timesteps' worth of data for temporal analysis could exceed a desired memory footprint target, which could have deleterious results on a simulation. This problem may be addressed in many potential ways. From an engineering perspective, taking advantage of new architectural features like burst buffers could help to overcome some of the tension resulting from needing more memory for *in situ* analysis but not using too much as to adversely impact the simulation. Algorithmic advances could offer some benefit in the form of finding ways to perform temporal analysis but without the need for growth in memory commensurate with the size of the time window being analyzed.

One trend we have witnessed in the HPC simulation space over the years is that simulations will adapt to make the best possible use of the underlying resources. For example, they may choose to not use all cores on a node so as to increase the relative amount of memory available to those cores they do use. *In situ* methods and infrastructures will need to be able to gracefully adapt to these situations, as well as to situations where there is architectural heterogeneity as well as differing platforms for running parallel code (GPUs, MIC CPUs, etc.). These are largely engineering issues, as opposed to algorithmic ones, whereby there is interplay between resource scheduling, provisioning and monitoring. The implication is that the *in situ* infrastructures themselves need to undergo continuous evolution in sophistication with respect to interacting with the system environment.

In that light, we envision a dichotomy emerging whereby the cost of entry for a new *in situ* infrastructure may be relatively high, but there is the desire to be able to quickly develop and deploy a new *in situ* method. To that end, a move towards a generic data interface between simulation codes and *in situ* infrastructures would be of great benefit. It would help to insulate the simulations from changes in the underlying *in situ* infrastructure, and could help faciliate an ecosystem that promotes *in situ* method reusability. In other words, one might like to be able to write an *in situ* method once, and then reasonably expect it to run without modification in any number of *in situ* infrastructures. Or, conversely, a simulation code developer could code to a generic *in situ* interface and reasonably expect it to work without modification with any number of *in situ* infrastructures.

Nearly all of the contemporary examples we have presented reflect a one-sided view of the problem space: the motivation is a widening gap between FLOPS and I/O, and so the work has focused on enabling knowledge discovery in spite of that widening gap. The other view of this problem returns to the roots of the *in situ* work that has its origins in the 1990s, namely getting data back into simulations.

There are multiple science drivers for this kind of work. One is to use simulations in concert with experiments, so that experimental data from an instrument is migrated into a simulation (boundary conditions, initial conditions, material properties, etc.). Then, the simulation's computations can help to predict how to tune an experiment to maximize its value. Another is in the area of code coupling, for projects like the Accelerated Climate Modeling for Energy (ACME) project [ACM] that aim to use multiple codes for different phases of climate modeling (atmosphere, ocean, cryosphere, etc.). Meeting the needs of these types of science drivers will require major advances in both the engineering and algorithmic aspects of *in situ* technology. A third type of driver comes from the need to use analysis methods and computations to alter the trajectory of a simulation. While many simulation codes do this kind of thing already, such as block-structured AMR performing grid refinement in the presence of "interesting physics" [Ber], there is opportunity to expand this type of capability in many different directions to leverage other analysis and computational methods to aid in producing better simulations.

A significant amount of research and engineering work remains to be done to facilitate *in situ* code reusability, so that research in this space has a vector through which it can be deployed into the hands of the user community as well as to simplify research activities by eliminating or reducing reinvention of key components of *in situ* infrastructure. Fundamental work in the *in situ* space is increasingly part of a larger ecosystem that includes complex and potentially distributed workflows, scientific code teams, resource-constrained execution environments and computational platforms of rapidly increasing complexity.

Acknowledgment

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, through the grant "Scalable Analysis Methods and *In Situ* Infrastructure for Extreme Scale Knowledge Discovery," program manager Dr. Lucy Nowell.

References

[ACG*14] Agranovsky A., Camp D., Garth C., Bethel E. W., Joy K. I., Childs H.: Improved Post Hoc Flow Analysis Via Lagrangian Representations. In *Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV)* (Paris, France, Nov. 2014), pp. 67–75. 6

[ACM] Accelerated Climate Modeling for Energy. http://climatemodeling.science.energy.gov/projects/accelerated-climate-modeling-energy, last accessed April 2016. 17

[Ahe12] Ahern S.: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press/Francis–Taylor Group, 2012, ch. The Path to Exascale, pp. 331–353. ISBN 978-1439875728. 2, 3

[Ahr15] Ahrens J.: Increasing scientific data insights about exascale class simulations under power and storage constraints. *IEEE Computer Graphics and Applications 35*, 2 (March/April 2015), 8–11. DOI 10.1109/MCG.2015.35. 2, 3

[AJO*14a] Ahrens J., Jourdain S., O'Leary P., Patchett J., Rogers D. H., Fasel P., Bauer A., Petersen M., Samsel F., Boeckel B.: In Situ MPAS-Ocean Image-Based Visualization. Online - The International Conference for High Performance Computing, Networking, Storage and Analysis, Visualization & Data Analytics Showcase, June 2014. URL: http://sc14.supercomputing.org/sites/all/themes/sc14/files/archive/sci_vis/sci_vis_pages/svs105.html. 16

[AJO*14b] Ahrens J., Jourdain S., O'Leary P., Patchett J., Rogers D. H., Petersen M.: 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* (Piscataway, NJ, USA, 2014), SC '14, IEEE Press, pp. 424–434. URL: http://dx.doi.org/10.1109/SC.2014.40, doi:10.1109/SC.2014.40. 4, 6, 9, 16

[ASM*11] Ahern S., Shoshani A., Ma K.-L., Choudhary A., Critchlow T., Klasky S., Pascucci V., Ahrens J., Bethel E., Childs H., et al.: Scientific discovery at the exascale. report from the doe ascr 2011 workshop on exascale data management. *Analysis, and Visualization 2* (2011). 3

[Aya15] Ayachit U.: *The ParaView Guide*, fourth ed. Kitware, Inc., 2015. ISBN 978-1-930934-30-6. 8

[BAB*12] Bennett J. C., Abbasi H., Bremer P.-T., Grout R., Gyulassy A., Jin T., Klasky S., Kolla H., Parashar M., Pascucci V., Pebay P., Thompson D., Yu H., Zhang F., , Chen J.: Combining In-situ and In-transit Processing to Enable Extreme-scale Scientific Analysis. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC) 2012* (Nov. 2012), pp. 1–9. doi:10.1109/SC.2012.31. 6, 11

[BDE13] Report on the big data and extreme-scale computing (BDEC) workshop. http://www.exascale.org/bdec/meeting/charleston, April-May 2013. 2, 3

[Ber] Local adaptive mesh refinement for shock hydrodynamics. 7, 17

[BG15] Biedert T., Garth C.: Contour tree depth images for large data visualization. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization* (2015), Eurographics Association, pp. 77–86. 6

[BGS15] Bauer A. C., Geveci B., Schroeder W.: *The ParaView Catalyst User's Guide v2.0*. Kitware, Inc., 2015. 8

[BJH94] Bethel E. W., Jacobsen J., Holland P.: Site Remediation in a Virtual Environment. In *Visual Data Exploration and Analysis, Proceedings of SPIE 2178* (San Jose CA, USA, Jan. 1994), Moorhead R. J., Silver D. E., Uselton S. P., (Eds.), pp. 78–87. LBNL-34865. 5

[BS05] Bethel E. W., Shalf J.: Consuming Network Bandwidth with Visapult. In *The Visualization Handbook*, Hansen C., Johnson C., (Eds.). Elsevier, 2005, pp. 569–589. LBNL-52171. URL: http://vis.lbl.gov/Publications/2003/LBNL-52171-VisapultChapter.pdf. 6

[BSO*12] Biddiscombe J., Soumagne J., Oger G., Guibert D., Piccinali J.-G.: Parallel computational steering for hpc applications using hdf5 files in distributed shared memory. *IEEE Transactions on Visualization and Computer Graphics 18*, 6 (2012), 852–864. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.63. 7

[BTl*00] Bethel W., Tierney B., lee J., Gunter D., Lau S.: Using High-speed WANs and Network Data Caches to Enable Remote and Distributed Visualization. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)* (Washington, DC, USA, 2000), IEEE Computer Society. LBNL-45365. 5

[BvRS*11] Bethel E. W., van Rosendale J., Southard D., Gaither K., Childs H., Brugger E., Ahern S.: Visualization at Supercomputing Centers: The Tale of Little Big Iron and the Three Skinny Guys. *IEEE Computer Graphics and Applications 31*, 1 (Jan/Feb 2011), 90–95. LBNL-4180E. 3

[BWT*11] Bremer P.-T., Weber G. H., Tierny J., Pascucci V., Day M., Bell J.: Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics 17*, 9 (2011), 1307–1324. 6

[cac] Cactus. http://cactuscode.org/, last accessed Feb. 2016. 7

[CGS*13] Childs H., Geveci B., Schroeder W., Meredith J., Moreland K., Sewell C., Kuhlen T., Bethel E. W.: Research challenges for visualization software. *IEEE Computer 46*, 5 (May 2013), 34–42. DOI 10.1109/MC.2013.179. 3

[Chi07] Childs H.: Architectural challenges and solutions for petascale postprocessing. *Journal of Physics: Conference Series 78*, 012012 (2007). DOI 10.1088/1742-6596/78/1/012012. 3

[CKH*04] Crivelli S., Kreylos O., Hamann B., Max N., Bethel E. W.: ProteinShop: A Tool for Interactive Protein Manipulation and Steering. *Journal of Computer Aided Molecular Design (JCAMD) 18*, 4 (Apr. 2004), 271–285. LBNL-53731. 7

[CMY*12] Childs H., Ma K.-L., Yu H., Whitlock B., Meredith J., Favre J., Klasky S., Podhorszki N., Schwan K., Wolf M., Parashar M., Zhang F.: In Situ Processing. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.), Chapman & Hall, CRC Computational Science. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA, Nov. 2012, pp. 307–329. http://www.crcpress.com/product/isbn/9781439875728, LBNL-6466E. 10

[con] Conduit. https://github.com/LLNL/conduit, last accessed Feb. 2016. 8

[CWW*13] Choi J. Y., Wu K., Wu J. C., Sim A., Liu Q. G., Wolf M., Chang C., Klasky S.: Icee: Wide-area in transit data processing framework for near real-time scientific applications. In *4th SC Workshop on Petascale (Big) Data Analytics: Challenges and Opportunities in conjunction with SC13* (2013). 11

[CYB08] Chen J., Yoon I., Bethel E. W.: Interactive, Internet Delivery of Visualization via Structured, Prerendered Multiresolution Imagery. *IEEE Transactions in Visualization and Computer Graphics 14*, 2 (2008), 302–312. LBNL-62252. 6

[dam] Damaris/Viz. http://damaris.gforge.inria.fr/doku.php, last accessed Feb. 2016. 7

[DCE*13] Dayal J., Cao J., Eisenhauer G., Schwan K., Wolf M., Zheng F., Abbasi H., Klasky S., Podhorszki N., Lofstead J. F.: I/o containers: Managing the data analytics and visualization pipelines of high end codes. In *IPDPS Workshops* (2013), pp. 2015–2024. 11

[DHH*15] Duque E. P., Hiepler D. E., Haimes R., Stone C. P., Gorrell S. E., Jones M., Spencer R.: Epic–an extract plug-in components toolkit for in situ data extracts architecture. In *22nd AIAA Computational Fluid Dynamics Conference* (2015), p. 3410. 7

[DL05] Duque E. P., Legensky S. M.: Visualization of large-scale unsteady computational fluid dynamics datasets. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference* (2005), IEEE, pp. 73–73. 6

[DPK12] Docan C., Parashar M., Klasky S.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing 15*, 2 (2012), 163–181. 11

[DSP*13] Dorier M., Sisneros R., Peterka T., Antoniu G., Semeraro D.: Damaris/viz: a nonintrusive, adaptable and user-friendly in situ visualization framework. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV '13)* (October 2013), pp. 67–75. 7

[DZJ*14] Docan C., Zhang F., Jin T., Bui H., Sun Q., Cummings J., Podhorszki N., Klasky S., Parashar M.: Activespaces: Exploring dynamic code deployment for extreme scale data processing. *Concurrency and Computation: Practice and Experience* (2014). 11

[EGH*06] Ellsworth D., Green B., Henze C., Moran P., Sandstrom T.: Concurrent visualization in a production supercomputing environment. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (Sept 2006), 997–1004. doi:10.1109/TVCG.2006.128. 6

[eps] EPSN. http://www.labri.fr/projet/epsn/, last accessed Feb. 2016. 8

[FBF*15] Fernandes O., Blom D. S., Frey S., Van Zuijlen S. H., Bijl H., Ertl T.: On in-situ visualization for strongly coupled partitioned fluid-structure interaction. In *VI International Conference on Computational Methods for Coupled Problems in Science and Engineering* (2015). 6

[FFSE14] Fernandes O., Frey S., Sadlo F., Ertl T.: Space-time volumetric depth images for in-situ visualization. In *IEEE Large Data and Visualization 2014 (LDAV14)* (2014). 7

[FK15] Fogal T., Krüger J.: An approach to lowering the in situ visualization barrier. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (2015), ACM, pp. 7–12. 8

[FLA] FLASH user guide. http://flash.uchicago.edu/website/. 15

[FLPS15] Forsythe J. R., Lynch C. E., Polsky S., Spalart P.: Coupled Flight Simulator and CFD Calculations of Ship Airwake using HPCMP CREATE-AV Kestrel. In *53th AIAA Aerospace Sciences Meeting, SciTech 2015* (Jan. 2015), pp. 1–18. doi:http://dx.doi.org/10.2514/6.2015-0556. 13, 14

[FMM*14] Fabian N., Moreland K., Mauldin J., Boeckel B., Ayachit U., Geveci B.: Instruction memory overhead of in situ visualization and analysis libraries on hpc machines. Presented at Ultrascale Visualization Workshop, Supercomputing2014. 10

[FMT*11] Fabian N., Moreland K., Thompson D., Bauer A. C., Marion P., Geveci B., Rasquin M., Jansen K. E.: The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV) 2011* (October 2011), Institute of Electrical and Electronics Engineers, pp. 89–96. 8

[FOR*00] Fryxell B., Olson K., Ricker P., Timmes F. X., Zingale M., Lamb D. Q., MacNeice P., Rosner R., Tufo H.: FLASH: An Adaptive Mesh Hydrodynamics Code for Modelling Astrophysical Thermonuclear Flashes. *Astrophysical Journal Supplement 131* (2000), 273–334. 15

[FPS*14] Fogal T., Proch F., Schiewe A., Hasemann O., Kempf A., Krüger J.: Freeprocessing: Transparent in situ visualization via data interception. In *Eurographics Symposium on Parallel Graphics and Visualization: EG PGV:[proceedings]/sponsored by Eurographics Association in cooperation with ACM SIGGRAPH. Eurographics Symposium on Parallel Graphics and Visualization* (2014), vol. 2014, NIH Public Access, p. 49. 8

[fre] Freeprocessing. https://github.com/tfogal/freeprocessing, last accessed Feb. 2016. 8

[GAL*03] Goodale T., Allen G., Lanfermann G., Massó J., Radke T., Seidel E., Shalf J.: The Cactus Framework and Toolkit: Design and Applications. In *Vector and Parallel Processing - VECPAR '2002, 5th International Conference* (2003), Springer. 5, 7

[Glo95] Globus A.: A software model for visualization of large unsteady 3-d cfd results. In *33rd Aerospace Sciences Meeting and Exhibit* (1995), AIAA. URL: http://arc.aiaa.org/doi/abs/10.2514/6.1995-115. 5

[Hai] Haimes R.: *pV3 http://raphael.mit.edu/pv3/*. MIT. 8

[Hai94] Haimes R.: pV3: A Distributed System for Large-scale Unsteady Visualization. In *AIAA Paper 91-0794* (1994). 5, 8

[Hai95] Haimes R.: Concurrent Distributed Visualization and Steering. In *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers* (1995). 5

[HB98] Heiland R., Baker M. P.: *A Survey of Co-Processing Systems*. Tech. rep., Technical Report, NCSA University of Illinois, August 1998. URL: http://sda.iu.edu/docs/CoprocSurvey.pdf. 5

[HDF] HDF Group: HDF5: Hierarchical Data Format. http://www.hdfgroup.org/HDF5. 12

[Ice] IceT: *http://icet.sandia.gov/*. 8

[Ins15] Institute S.:, 2015. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: http://www.scirun.org. 5, 8

[ins16] The In Situ Terminology Project. https://ix.cs.uoregon.edu/~hank/insituterminology/index.cgi?n=Phase1D.Phase1DSurveyInput, Feb. 2016. 3

[JBDGH95] Jacobsen J., Bethel E. W., Datta-Gupta A., Holland P.: Petroleum Reservoir Simulation in a Virtual Environment. In *Proceedings of the 13th Symposium on Reservoir Simulation (SPE)* (San Antonio TX, USA, 1995). 5

[Ker] KerData: *Damaris http://damaris.gforge.inria.fr/doku.php*. INRIA Rennes. 7

[Koh] Kohl J.: *CUMULVS http://www.csm.ornl.gov/cs/cumulvs.html*. ORNL. 7

[LBC*15] Larsen M., Brugger E., Childs H., Eliot J., Griffin K., Harrison C.: Strawman: A batch in situ visualization and analysis infrastructure for multi-physics simulation codes. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2015), ISAV2015, ACM, pp. 30–35. URL: http://doi.acm.org/10.1145/2828612.2828625, doi:10.1145/2828612.2828625. 8

[LDL*] Latham R., Daley C., Liao W.-K., Gao K., Ross R., Dubey A., Choudhary A.: A Case Study for Scientific I/O: Improving the FLASH Astrophysics Code. http://www.mcs.anl.gov/uploads/cels/papers/P1819.pdf. 15

[LGP*15] Li S., Gruchalla K., Potter K., Clyne J., Childs H.: Evaluating the Efficacy of Wavelet Configurations on Turbulent-Flow Data. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Chicago, IL, Oct. 2015), pp. 81–89. 7

[LJ14] Lehmann H., Jung B.-I.: In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on* (2014), IEEE, pp. 51–58. 7

[LKS*08] Lofstead J. F., Klasky S., Schwan K., Podhorszki N., Jin C.: Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments* (2008), ACM, pp. 15–24. 11

[LLC*03] Li J., Liao W., Choudhary A., Ross R., Thakur R., Gropp W., Latham R., Siegel A., Gallagher B., Zingale M.: Parallel netCDF: A High-Performance Scientific I/O Interface. In *ACM/IEEE Conference on Supercomputing* (Phoenix, AZ, Nov. 2003). 12

[LLT*14] Liu Q., Logan J., Tian Y., Abbasi H., Podhorszki N., Choi J. Y., Klasky S., Tchoua R., Lofstead J., Oldfield R., Parashar M., Samatova N., Schwan K., Shoshani A., Wolf M., Wu K., Yu W.: Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience 26*, 7 (2014), 1453–1473. URL: http://dx.doi.org/10.1002/cpe.3125, doi:10.1002/cpe.3125. 11

[LOK12] Lofstead G. F., Oldfield R. A., Kordenbrock T.: *Experiences Applying Data Staging Technology in Unconventional Ways*. Nov 2012. 8

[LOKR11] Lofstead J., Oldfield R., Kordenbrock T., Reiss C.: Extending scalability of collective io through nessie and staging. In *Proceedings of the Sixth Workshop on Parallel Data Storage* (New York, NY, USA, 2011), PDSW '11, ACM, pp. 7–12. URL: http://doi.acm.org/10.1145/2159352.2159355, doi:10.1145/2159352.2159355. 8

[LVI*13] Leaf N., Vishwanath V., Insley J., Hereld M., Papka M., Ma K.-L.: Efficient parallel volume rendering of large-scale adaptive mesh refinement data. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on* (Oct 2013), pp. 35–42. doi:10.1109/LDAV.2013.6675156. 15

[LZKS09] Lofstead J., Zheng F., Klasky S., Schwan K.: Adaptable, metadata rich io methods for portable high performance io. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (May 2009), pp. 1–10. doi:10.1109/IPDPS.2009.5161052. 3, 9

[Ma95] Ma K.-L.: Runtime volume visualization of parallel cfd. In *Proceedings of Parallel CFD Conference* (1995), pp. 307–314. 5

[Ma09] Ma K.-L.: In situ visualization at extreme scale: Challenges and opportunities. *IEEE Computer Graphics and Applications 29*, 6 (2009), 14–19. URL: http://dx.doi.org/10.1109/MCG.2009.120, doi:10.1109/MCG.2009.120. 16

[MOM*00] MacNeice P., Olson K. M., Mobarry C., de Fainchtein R.,

Packer C.: PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications 126* (2000). 15

[Mor12] Moreland K.: Oh, $#*@! Exascale! The effect of emerging architectures on scientific discovery. In *2012 SC Companion (Proceedings of the Ultrascale Visualization Workshop)* (November 2012), pp. 224–231. DOI 10.1109/SC.Companion.2012.38. 3

[Mor16] Moreland K.: The tensions of in situ visualization. *IEEE Computer Graphics & Applications 36*, 2 (March/April 2016), 5–9. DOI 10.1109/MCG.2016.35. 2

[MVM*15] Malakar P., Vishwanath V., Munson T., Knight C., Hereld M., Leyffer S., Papka M. E.: Optimal scheduling of in-situ analysis for large-scale scientific simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2015), SC '15, ACM, pp. 52:1–52:11. URL: http://doi.acm.org/10.1145/2807591.2807656, doi:10.1145/2807591.2807656. 15

[MvWvL99] Mulder J. D., van Wijk J. J., van Liere R.: A survey of computational steering environments. *Future Gener. Comput. Syst. 15*, 1 (Feb. 1999), 119–129. URL: http://dx.doi.org/10.1016/S0167-739X(98)00047-8, doi:10.1016/S0167-739X(98)00047-8. 5

[MW13] Morozov D., Weber G.: Distributed Merge Trees. In *Proceedings of the Annual Symposium on Principles and Practice of Parallel Programming* (2013), pp. 93–102. 6

[NCA] NCAR Graphics. http://ngwww.ucar.edu/, last accessed Feb. 2016. 5

[NWP*14] Nouanesengsy B., Woodring J., Patchett J., Myers K., Ahrens J.: 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* (2014), IEEE, pp. 43–50. 7

[OAJ*15] O'Leary P., Ahrens J., Jourdain S., Wittenburg S., Rogers D. H., Petersen M.: Cinema image-based in situ analysis and visualization of mpas-ocean simulations. *Parallel Computing* (2015). URL: http://www.sciencedirect.com/science/article/pii/S0167819115001349, doi:http://dx.doi.org/10.1016/j.parco.2015.10.005. 16

[OCJ*15] O'Leary P., Christon M., Jourdain S., Harris C., Berndt M., Bauer A.: Hpccloud: A cloud/web-based simulation environment. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)* (2015), IEEE, pp. 25–33. 12

[OMFR14] Oldfield R. A., Moreland K., Fabian N., Rogers D.: Evaluation of methods to integrate analysis into a large-scale shock shock physics code. In *Proceedings of the 28th ACM International Conference on Supercomputing* (New York, NY, USA, 2014), ICS, ACM, pp. 83–92. URL: http://doi.acm.org/10.1145/2597652.2597668, doi:10.1145/2597652.2597668. 9

[PJB97] Parker S. G., Johnson C. R., Beazley D.: Computational steering software systems and strategies. *IEEE Comput. Sci. Eng. 4*, 4 (Oct. 1997), 50–59. URL: http://dx.doi.org/10.1109/99.641609, doi:10.1109/99.641609. 5, 8

[RCMS12] Rivi M., Calori L., Muscianisi G., Slavnic V.: *In-situ Visualization: State-of-the-art and Some Use Cases*. Tech. rep., Whitepaper PRACE, 2012. 7

[RLV*16] Rübel O., Loring B., Vay J.-L., Grote D. P., Lehe R., Bulanov S., Vincenti H., Bethel E. W.: *In Situ* Visualization and Analysis of Ion Accelerator Simulations Using Warp and VisIt. *IEEE Computer Graphics and Applications* (May 2016). to appear. 6, 7

[RPH*13] Ringler T., Petersen M., Higdon R. L., Jacobsen D., Jones P. W., Maltrud M.: A multi-resolution approach to global ocean modeling. *Ocean Modelling 69* (2013), 211–232. 16

[RSC*14] Rasquin M., Smith C., Chitale K., Seol E. S., Matthews B. A., Martin J. L., Sahni O., Loy R. M., Shephard M. S., Jansen K. E.: Scalable implicit flow solver for realistic wing simulations with flow control. *Computing in Science and Engineering 16*, 6 (Nov.-Dec. 2014), 13–21. 8

[SML04] Schroeder W., Martin K., Lorensen B.: *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*, fourth ed. Kitware, Inc., 2004. ISBN 1-930934-19-X. 8

[TCM10] Tikhonova A., Correa C. D., Ma K.-L.: Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1551–1559. 6

[Tri] Trilinos: *https://trilinos.org/*. 8

[TSO*11] Turk M. J., Smith B. D., Oishi J. S., Skory S., Skillman S. W., Abel T., Norman M. L.: yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series 192*, 1 (2011). DOI 10.1088/0067-0049/192/1/9. 8

[TYC*11] Tikhonova A., Yu H., Correa C. D., Chen J. H., Ma K.-L.: A preview and exploratory technique for large-scale scientific simulations. In *EGPGV* (2011), Citeseer, pp. 111–120. 6

[TYRG*06] Tu T., Yu H., Ramirez-Guzman L., Bielak J., Ghattas O., Ma K.-L., O'Hallaron D. R.: From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (November 2006). DOI 10.1109/SC.2006.32. 5

[UFK*89] Upson C., Faulhaber, Jr. T. A., Kamins D., Laidlaw D., Schlegel D., Vroom J., Gurwitz R., van Dam A.: The Application Visualization System: a computational environment for scientific visualization. *j-IEEE-CGA 9*, 4 (July 1989), 30–42. 5

[VBHP14] Vishwanath V., Bui H., Hereld M., Papka M.: *GLEAN*. High Performance Parallel I/O Book, CRC Press, Taylor and Francis Group, November 2014. 12

[VHMP11] Vishwanath V., Hereld M., Morozov V., Papka M. E.: Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 19:1–19:11. URL: http://doi.acm.org/10.1145/2063384.2063409, doi:10.1145/2063384.2063409. 3, 11

[vis] VisDebug. https://github.com/tfogal/visdebug, last accessed Feb. 2016. 8

[WFL16] Whitlock B. J., Forsythe J. R., Legensky S. M.: In Situ Infrastructure Enhancements for Data Extract Generation. In *54th AIAA Aerospace Sciences Meeting, SciTech 2016* (Jan. 2016), pp. 1–12. URL: http://dx.doi.org/10.2514/6.2016-1928, doi:doi:10.2514/6.2016-1928. 10

[WFM11] Whitlock B., Favre J. M., Meredith J. S.: Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization* (2011), Eurographics Association, pp. 101–109. 10

[YMM13] Ye Y., Miller R., Ma K.-L.: In situ pathtube visualization with explorable images. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2013), EGPGV '13, Eurographics Association, pp. 9–16. URL: http://dx.doi.org/10.2312/EGPGV/EGPGV13/009-016, doi:10.2312/EGPGV/EGPGV13/009-016. 6

[YWG*10] Yu H., Wang C., Grout R. W., Chen J. H., Ma K.-L.: In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications 30*, 3 (2010), 45–57. doi:http://doi.ieeecomputersociety.org/10.1109/MCG.2010.55. 6

[YWM*15] Ye Y. C., Wang Y., Miller R., Ma K.-L., Ono K.: In situ depth maps based feature extraction and tracking. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on* (2015), IEEE, pp. 1–8. 6

[ZABP15] Ziegeler S., Atkins C., Bauer A., Pettey L.: In situ analysis as a parallel i/o problem. In *Proceedings of the First Workshop*

on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (New York, NY, USA, 2015), ISAV2015, ACM, pp. 13–18. URL: http://doi.acm.org/10.1145/2828612.2828620, doi:10.1145/2828612.2828620. 8

[ZAD*10] Zheng F., Abbasi H., Docan C., Lofstead J., Liu Q., Klasky S., Parashar M., Podhorszki N., Schwan K., Wolf M.: Predata–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on* (2010), IEEE, pp. 1–12. 11

[Zaj64] Zajac E. E.: Computer-made perspective movies as a scientific and communication tool. *Communications of the ACM 7*, 3 (Mar. 1964), 169–170. 5

[ZCD*11] Zheng F., Cao J., Dayal J., Eisenhauer G., Schwan K., Wolf M., Abbasi H., Klasky S., Podhorszki N.: High end scientific codes with computational i/o pipelines: improving their end-to-end performance. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities* (2011), ACM, pp. 23–28. 11

[ZZC*13] Zheng F., Zou H., Cao J., Dayal J., Nugye T., Eisenhauer G., Klasky S.: Flexio: Location-flexible execution of in situ data analytics for large scale scientific applications. In *Proc. IEEE International Parallel and Distributed Processing Symp (IPDPSâĂŹ13)* (2013), pp. 320–331. 11