In Situ Visualization Techniques for High Performance Computing

James Kress
Department of Computer and Information Science,
University of Oregon, Eugene Oregon
jkress@cs.uoregon.edu

***Abstract*—Scientific visualization for exascale computing is very likely to require in situ processing. Traditional simulation checkpointing and post hoc visualization will likely be unsustainable on future systems at current trends due to the growing gap between I/O bandwidth and FLOPS. As a result, the majority of simulation data may be lost if in situ visualization techniques are not deployed. In situ visualization in this paradigm will be given unprecedented access to simulation output, potentially being able to process all relevant simulation output at every simulation time step, allowing for very high temporal fidelity compared to traditional post hoc visualization. However, this access poses many challenges in terms of data management, resource management and sharing, algorithm development and design, and implementation approaches. Currently, the community primarily relies on two in situ techniques: tightly coupled (on the same resource as the simulation) and loosely coupled (not sharing resources with the simulation). Each of these approaches have positive and negative factors which affect their performance under different simulation, resource, and visualization type constraints. Meaning, that for every given visualization task, it is not generally known which method would give the best performance on every data type, architecture, and set of resource constraints. Due to the lack of research and development on this topic it is still an open research problem requiring future research.**

I INTRODUCTION

As leading-edge supercomputers get increasingly powerful, scientific simulations running on these machines are generating ever larger volumes of data. However, the increasing cost of data movement, in particular moving data to disk, is increasingly limiting the ability to process, analyze, and fully comprehend simulation results [1], hampering knowledge extraction. Specifically, while I/O bandwidths regularly increase with each new supercomputer, these increases are well below corresponding increases in computational ability and data generated. Further, this trend is predicted to persist for the foreseeable future.

Relative decreases in I/O pose a problem for stakeholders running on these systems ranging from simulation scientists to visualization researchers. To that end, the Advanced Scientific Computing Research (ASCR) Scientific Grand Challenges Workshop Series produced reports spanning eight different scientific domains (High Energy Physics, Climate, Nuclear Physics, Fusion, Nuclear Energy, Basic Energy Sciences, Biology, National Security) [2], [3], [4], [5], [6], [7], [8], [9], that explored the computing challenges, including visualization and analysis challenges, for codes in each of those eight domains. Each report mentioned data movement, storage, and analysis as a major obstacle in the move to exascale. Many of these scientific domains will be required to deal with petabytes, or even exabytes, of data over the course of a simulation.

This trend poses a problem for the traditional post-processing visualization methodology. The traditional visualization workflow performs visualization as a post-processing task, where simulation outputs are read from disk, into the memory of a parallel tool where analysis and visualization are performed. Visualization is generally I/O bound [10], [11], and as the relative I/O bandwidth continues to decrease, the challenges of visualizing increasingly larger data will become more problematic. Post hoc visualization is particularly sensitive to the I/O bottleneck, as data is first written to disk by the simulation, and then read back from disk by the visualization routine.

Given this reality, many large-scale simulation codes are attempting to bypass the I/O bottleneck by using in situ visualization and analysis, i.e., processing simulation data when it is generated. Broadly speaking, two paradigms have emerged [12]. First, *co-processing*, or *tightly coupled*, methods, where the simulation and visualization code run in the same process using the same resources. Second, *concurrent-processing*, or *loosely coupled*, methods, where the simulation transfers data over the network to a separate set of visualization nodes for processing.

A key question for in situ analysis is whether there is a priori knowledge of which visualizations and analyses to produce. If this a priori knowledge exists, then co-processing techniques are an option, and will avoid network and additional resource usage. However, it is not guaranteed that the required visualizations and analyses are known a priori. That is, a domain scientist may need to explore the data in an interactive fashion, or even produce unanticipated analyses, lending itself more towards concurrent-processing methods. Under these constraints, it becomes clear that both types of in situ visualization have a place in the workflows of future simulations, but there are challenges for each that need to be addressed.

In situ processing poses many new challenges to both simulation and visualization scientists that were hidden or less predominant with the post-processing paradigm. A few of the issues facing in situ include: how the in situ routines are integrated with the simulation, how data is translated from the simulation representation to the visualization representation, how resources are allocated between the simulation and the visualization, how faults are isolated in the visualization routines, how to massively scale communication heavy visualization algorithms, and even how to do exploratory visualization in an in situ world. One avenue of approach that could aid in

solving these problems, is the ability to model them under varying computational setups and data loads. This modeling work is an exciting area of future research for in situ.

In the remainder of this paper, we survey and explore in situ visualization itself, key areas involved in in situ workflows, and identify areas where the research is incomplete, or requires further study. This paper is organized as follows: trends in high performance computing and work in scientific visualization and graphics are duscussed in Section II, the state of in situ visualization is discussed in Section III, and an overview of the limited area of in situ performance modeling is presented in Section IV.

II BACKGROUND

In this section, we will cover important areas of background research for in situ visualization and analysis. First, we look at trends in high performance computing and their implications for the future of visualization. Next, we explore the traditional scientific visualization and compositing pipelines, and discuss prevalent scientific visualization tools including current research in the area of data models, portable performance, and massive scaling.

II-A High Performance Computing

High Performance Computing (HPC) is a landscape of constant evolution. This evolution is seen in the composition of the HPC systems themselves, as well as the science that they enable. By using these systems, scientists have gained deeper understandings in fields ranging from medicine to energy to physics to even national security. Computers have seen nearly a 50 billion-fold increase in computing power over the last 70 years [13]. Compared to other technologies, this is virtually an unprecedented leap, enabling more than ever before, but bringing with it a vast set of challenges.

One of those primary challenges is power. The Department of Energy has set a nominal power cap for exascale systems at 20 MW per year. This roughly equates to a yearly energy bill of $20 million dollars. However, reaching this goal is not easy. It would be possible to construct an exascale system today using conventional hardware and components, but DARPA estimated in 2008 that this system's power requirements would reach into the 100's of MW, far beyond the maximum power bound [15]. This estimate has since dropped with new system designs being introduced, but it is still far beyond the 20 MW cap.

Therefore, to reach the performance goal given the maximum power bound, system designers are having to divert from the traditional approach for scaling HPC systems, by transitioning them from multi-core to many-core. This transition pares down the power of the traditional central processing unit in each node of the supercomputer, and instead, gets its performance by utilizing many low power cores on devices such as GPUs and Intel Xeon Phis. Indeed, this trend is already being seen as the current generation of Department of Energy computing systems are being prepared for their 2018 redesigns/upgrades.

Table I shows the three DOE supercomputing systems up for upgrades in the 2016 to 2018 time frame. Focusing on just Titan, a drastic change is scheduled to take place in the topology of this system. Currently, this system is the third fastest computer in the world [16]. It contains 18,688 nodes, consumes a total of 9 MW of power, and has a peak performance of 27 PF. However, new upgrades are going to drastically cut the number of nodes in the system down to just around 3,500 nodes, a total power consumption of 10 MW, and a peak performance of 150 PF. This change highlights that the challenges of exascale are already here. Moving from a system that currently has million-way concurrency towards a new system which will have billion-way concurrency will require a redesign of not only the simulations and codes running on this system (focusing on parallelizing the underlying algorithms) [17], but also in how data is saved and analyzed [18].

Taking it one step further, Table II shows the expected characteristics of an actual machine at exascale. This table focuses on the system performance versus the system I/O, in order to highlight the data challenge. The system peak performance is expected to rise by a factor of 500, yet the I/O capacity is only expected to rise by a factor of 20. This means that the current problems faced by simulation codes in terms of how frequently they can save data are only going to get worse. Take, for example, the leading-edge fusion simulation code XGC1 which saves time steps on average every 100 steps [20]. Moving this code to an exascale system without addressing the data problem is going to mean that time steps will now only be saved every 1,000 to 10,000 steps. This will drastically increase the likelihood that interesting physics will be lost between saves.

In situ processing can address this with faster analysis of data streams without having to first send data to disk. This means that a higher temporal fidelity of data will be available for analysis, while even potentially enabling the possibility of interactive steering of the simulation through the visualization [21]. In situ is an enabling technology and is discussed further in Section III.

II-B Scientific Visualization

Visualization is an enabling technology that facilitates insight into data across many domains. It is an essential tool for confirming and communicating trends in data to both the domain scientists as well as the general public [22]. Traditionally, scientific visualization has been performed as a post processing task. That is, a simulation will save all of the data needed for visualization to disk, and after the run is complete, visualization can begin. This approach has the benefit that the visualization software has access to all of the data from every step all at once, making algorithms and visualization workflows easier to develop.

Most of the parallelism in current scientific visualization tools relies on not just distributed memory parallelism, but specifically the message passing interface (MPI). MPI is heavyweight, and requires a whole copy of the visualization program per process. As we transition our visualization codes to higher and higher concurrencies on the march to exascale, this overhead can exceed the system memory and disk space before any data is even loaded [19]. This revelation is important to consider when running a visualization tool at scales approaching those the size of the scientific simulations themselves.

In order to achieve parallel scalability for massive thread-

TABLE I: Current and projected system statistics for Advanced Scientific Computing Research Programs computing resources after the next set of system procurements are completed. Two areas of critical importance to note are the node processors and the system size of the current machines compared to the next evolution. Visualization codes will be expected to work efficiently on concurrencies and architectures never seen before, meaning that the challenges from exascale computing are already emerging now (table from [14]).

System attributes	NERSC Now	OLCF Now	ALCF Now	NERSC Upgrade	OLCF Upgrade	ALCF Upgrades	
Name Planned Installation	**Edison**	**TITAN**	**MIRA**	**Cori 2016**	**Summit 2017-2018**	**Theta 2016**	**Aurora 2018-2019**
System peak (PF)	2.6	27	10	> 30	150	>8.5	180
Peak Power (MW)	2	9	4.8	< 3.7	10	1.7	13
Total system memory	357 TB	710TB	768TB	~1 PB DDR4 + High Bandwidth Memory (HBM)+1.5PB persistent memory	> 1.74 PB DDR4 + HBM + 2.8 PB persistent memory	>480 TB DDR4 + High Bandwidth Memory (HBM)	> 7 PB High Bandwidth On-Package Memory Local Memory and Persistent Memory
Node performance (TF)	0.460	1.452	0.204	> 3	> 40	> 3	> 17 times Mira
Node processors	Intel Ivy Bridge	AMD Opteron Nvidia Kepler	64-bit PowerPC A2	Intel Knights Landing many core CPUs Intel Haswell CPU in data partition	Multiple IBM Power9 CPUs & multiple Nvidia Voltas GPUS	Intel Knights Landing Xeon Phi many core CPUs	Knights Hill Xeon Phi many core CPUs
System size (nodes)	5,600 nodes	18,688 nodes	49,152	9,300 nodes 1,900 nodes in data partition	~3,500 nodes	>2,500 nodes	>50,000 nodes
System Interconnect	Aries	Gemini	5D Torus	Aries	Dual Rail EDR-IB	Aries	2nd Generation Intel Omni-Path Architecture
File System	7.6 PB 168 GB/s, Lustre®	32 PB 1 TB/s, Lustre®	26 PB 300 GB/s GPFS™	28 PB 744 GB/s Lustre®	120 PB 1 TB/s GPFS™	10PB, 210 GB/s Lustre initial	150 PB 1 TB/s Lustre®

TABLE II: Current petascale system performance compared against the design target for the 2023 exascale system. Moving to billion way concurrency and an exaflop in performance are critical challenges for visualization when compared to current visualization algorithm scaling and the network bandwidth when trying to move data to disk (adapted from [19]).

System Parameter	2011	"2023"		Factor Change
System Peak	2 PF	1 EF		500
Power	6 MW	<= 20 MW		3
System Memory	0.3 PB	32 PB	64 PB	100-200
Total Concurrency	225K	1Bx10		40,000
Node Performance	125 GF	1 TF	10 TF	8-80
Node Concurrency	12	1,000	10,000	83-830
Network BW	1.5 GB/s	100 GB/s	1000 GB/s	66-660
System Size (nodes)	18,700	100,000	1,000,000	50-500
I/O Capacity	15 PB	300 PB	1000 PB	20-67

ing, visualization algorithms will have to be redesigned [23]. The key in this redesign will be to focus on data model, data interdependencies, and portable performance. In the following subsections, we will focus on three specific themes in visualization:

1) Section II-B1 will look at tools currently being used and developed by the visualization community in terms of their scalability, data models, and challenges for exascale.
2) Section II-B2 will explore current trends in graphics for visualization, focusing on image generation in a highly parallel environment.
3) Section II-B3 will look at color for scientific visualization, as color map choices have large implications for interpreting and understanding a visualization.

II-B1 High Profile Scientific Visualization Tools

There are several tools for scientific visualization that have gained wide adoption and use in the community. Central to the performance of each of these tools are their underlying data models and implementations. In the following sections we will describe several high profile tools for scientific visualization, including how they handle data on a high level, describing

how this impacts performance and use on future systems, and implications for in situ visualization.

II-B1.1 AVS and OpenDX

The Application Visualization System (AVS) [24] and OpenDX [25] are two early versions of open source visualization tools. AVS is a system that provides a modular interactive approach to forming visualization pipelines. Visualization components are construted visually into flow graphs to create the final visualization product. OpenDX emerged a few years after AVS, and was the open source version of IBM's Data Explorer. OpenDX also had a visual programming interface for constructing visualization pipelines, and contrained many built-in visualization options. These tools have lost prominance with the emergence of newer tools with more refined API's that allow easier integration into existing scientific workflows and batch scheduling systems.

II-B1.2 VTK

VTK, also known as the Visualization Toolkit [26], is an ongoing software effort enabling extensible visualization and analysis for a wide variety of data set types and filters. The underlying design goals of this toolkit are to be portable, standards based, freely available, and simple [27]. Further, two scalable visualization tools, ParaView [28] and VisIt [29], make use of VTK's foundational data models.

The following discussion of VTK will focus on its data model, as data models are one of the most foundational elements of a visualization tool, and have wide implications in terms of the expressiveness of the data model and its memory overhead in a visualization pipeline.

VTK Data Model: VTK's data model exposes a few core mesh types, which are extensible and can be applied to a wide range of scientific domains. The main mesh types supported by VTK are rectilinear, structured, and unstructured. These three mesh types represent the geometric structure of the data set. Each mesh type consists of point locations in three-dimensional space, cells that reference the area between those points, and fields defined on the points or cells. The fields are stored as values in any number of arrays of data, which can be aligned on the points or cells, or unaligned. The values can range from simple scalar numeric quantities to vector or tensor quantities to more complicated types, such as strings.

VTK Data Model Shortcomings for In Situ and Future Architectures: The main shortcoming of VTK's data model is related to the expressiveness of the model itself in accurately and efficiently representing the multitude types of data produced by simulation codes. VTK's data model supports only a small number of mesh types, such as unstructured and rectilinear grids, but contemporary simulations are representing more complex data instantiations. Even if the data model can accurately represent the simulation data, the data is often forced into an inefficient data structure because VTK has assumed the data will fall into one of the few defined mesh types. Often times, the data does not fit into one of these structures, so it must be forced into a less efficient one.

Another shortcoming of VTK's data model is related to parallelism on future architectures. VTK's data model does not support the recent trends in hardware parallelism resulting from accelerators, such as GPUs. Its data model is also limited in that it does not leverage or support data parallelism.

Lastly, the VTK data model poses challenges when operating on very large data. In the general VTK visualization pipeline, a filter is applied to a data set, and the result is a completely new data set. This means that in general, each filter applied to a VTK data set results in a new data set being created, severely bloating memory. This approach to memory management in a data model does not scale well for in situ approaches, and will be even more problematic on the next generation of supercomputers.

In summary, VTK's data model lacks support for necessary features, such as general higher order polynomial elements, non-Cartesian space or dimensionalities greater than three, and mixed-dimensionality elements in a single data set. As we move to the next generation of architectures and continue evolving scientific simulation codes, there is an increasing demand for an improved and more advanced data model that is extensible and can enable us to represent a wider range of data types. These new representations and memory efficiency are especially important for use in situ, when memory use and an easy translation from simulation data representation to visualization data representation is needed.

II-B1.3 VisIt and ParaView

VisIt and ParaView are two open source visualization tools developed, at least in part, through the efforts of U.S. National Laboratories. The history of these tools span many years, and will not be presented here. Instead, the primary design philosophy and major features for end-users will be discussed and then compared to the needs of in situ visualization.

VisIt: VisIt is an end-user visualization and analysis tool designed to work on very large and diverse data [30]. Moreover, VisIt was designed for more than just data visualization. It lists five primary use cases that it focuses on [29]:

- Visual Exploration: the creation of images from data.
- Debugging: users can locate hard-to-find problems in their data.
- Quantitative Analysis: users can perform quantitative analysis through the interface to ask very complex questions of their data.
- Comparative Analysis: allows different simulation runs or multiple time steps to be compared.
- Communication: users present their findings to a large audience through movies, images, and plots.

Core to the VisIt design is its extensibility. It allows for new components to be inserted by end-users easily. This extensibility and ease of use makes it a very successful tool, one used across a multitude of scientific domains.

VisIt is designed to work as a distributed system. It has a server that utilizes parallel compute capabilities coupled with

the client running as the user interface. In addition, VisIt has capabilities of running in situ with LibSim [31], enabling users to utilize the full feature list of VisIt during in situ instrumentation (the in situ capabilities will be explored further in Section III). VisIt has been shown to scale effectively to tens of thousands of cores, and is widely used by scientists running on some of the largest systems all over the world.

In summary, VisIt is a very powerful visualization tool, that is applicable in a wide variety of use cases. However, two limitations do exist when looking at the use of VisIt in situ: VisIt utilizes VTK under the hood, so the data model issues from VTK come into play. In addition, the visualization library is fairly heavy weight, and can cause problems when performing different types of in situ integrations, potentially making it a sub-optimal approach.

ParaView: ParaView is another end-user tool for the visualization of large data. ParaView was designed with the philosophy of being open-source and multi-platform, extensible for different architectures, allowing support for distributed computation, and providing an intuitive user interface.

ParaView was designed as a layered architecture, with three distinct layers [28]. The first is VTK, which provides the data model and underlying algorithms. Second is the parallel extension to VTK to allow for streaming and distributed-memory parallel execution. Third is ParaView itself, predominantly composed of the GUI.

ParaView has been shown to scale well in distributed-memory parallel execution mode, on very large data. In addition, ParaView enables in situ integrations through ParaView Catalyst [32] (the in situ capabilities will be explored further in Section III).

In summary, ParaView is a very powerful visualization tool, that is applicable in a wide variety of use cases. However, two limitations do exist when looking at the use of ParaView in situ: ParaView utilizes VTK under the hood, so the data model issues from VTK come into play, in addition (as with VisIt), the visualization library is fairly heavy weight, and can cause problems when performing different types of in situ integrations, potentially making its a sub-optimal approach.

II-B1.4 EAVL, Dax, PISTON

EAVL [33], [34], Dax [35], and PISTON [36] are three frameworks developed with a mission to explore methods of transitioning visualization algorithms to the available parallelism of emerging many-core hardware architectures targeted for exascale [37].

- ⋆ EAVL (Extreme-scale Analysis and Visualization Library) was developed to address three primary objectives: update the traditional data model to handle modern simulation codes; investigate the efficiency of I/O, computation and memory on an updated data and execution model; and explore visualization algorithms on next-generation architectures.

  The heart of the EAVL approach is the data model. EAVL defines more flexible meshes, and data structures which more efficiently supports the traditional types of data supported by de-facto standards like VTK, but also allows for efficient representations of non-traditional data. Examples of non-traditional data includes graphs, mixed data types (e.g., molecular data, high order field data, unique mesh topologies (e.g., unstructured adaptive mesh refinement and quad-trees)).

  EAVL uses a functor concept in the execution model to allow users to write operations that are applied to data. The functor concept in EAVL has been abstracted to allow for execution on either the CPU or GPU, and the execution model manages the movement of data to the particular execution hardware.
- ⋆ The primary strength of the Dax Toolkit is its exploration of achieving high node-level concurrency, at the levels needed for efficient exascale visualization. This is accomplished through the use of *worklets*, which are functions that implement a given algorithm's behavior on an element of a mesh, or a small local neighborhood. The worklets are constrained to be serial and stateless, which enable concurrent scheduling on an unlimited number of threads.
- ⋆ PISTON was developed with the goal of facilitating the development of visualization and analysis operators that had highly portable performance. The idea being that there are many different architectures that a visualization algorithm may be run on, and developing and tuning algorithms specific to each architecture is an inefficient and undesirable approach for visualization. To that end, PISTON is built on top of Thrust [38], which provides implementations of data-parallel primitives in CUDA, OpenMP, and TBB. This approach allows algorithms to be implemented once, and ported to the correct architecture at compile time.

In summary, each of these frameworks provided valuable insight into methods for transitioning visualization pipelines to many-core architectures, and to natively supporting in situ visualization. The best elements from each of these frameworks were used to form the foundation for VTK-m.

II-B1.5 VTK-m

VTK-m is an effort that has merged the best aspects of three previously described projects, EAVL, Dax and PISTON [39]. The motivator behind VTK-m is to create a high-performance portable visualization library. The portable nature of VTK-m is achieved through its use of data-parallel primitives (DPPs), first described by Blelloch [40]. Data-parallel primitives are designed in a way such that a variety of algorithms can be expressed using a relatively small selection of DPPs, such as map, scan, reduce, and so on. These primitives allow VTK-m to be moved between many different architectures without having to redesign each individual visualization routine. Central to the portable nature of VTK-m is the underlying data model, which is similar to that of EAVL, but with even greater freedom.

The data model in VTK-m was designed to be flexible enough to accommodate the myriad of different data layouts of scientific domains that may use VTK-m, while still providing a clear set of semantics. Furthermore, the data representation must be space efficient and be accessible on the different

processor types in use (that is, work on both CPU and GPU). As shown in Figure 1, a VTK-m data set consists of three components: cell sets, coordinate systems, and fields. By

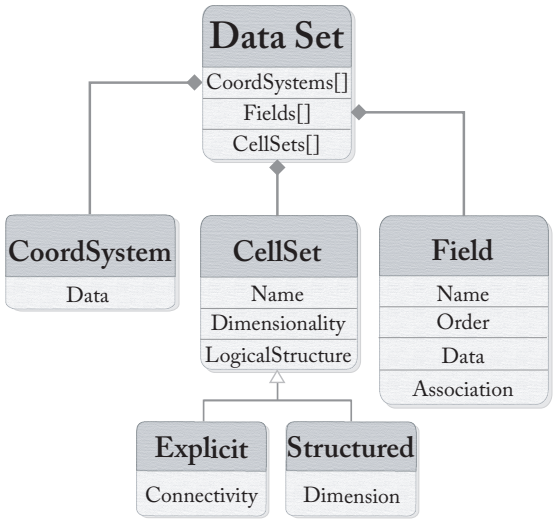


Fig. 1: VTK-m data model overview

allowing arbitrary combinations of coordinate systems, cell sets, and fields, VTK-m is able to overcome the inefficiencies and difficulties in data representation imposed by traditional data models. Traditional data models often choose a set of rigid characteristics for a data set. These rigid characteristics then are labeled as a specific type of mesh. For example, a uniform data set has regular axis-aligned coordinates and a logical $[i,\ j,\ k]$ cell arrangement. An unstructured data set has fully explicit coordinates (a $[x,\ y,\ z]$ value separately defined for each point) with fully explicit cell connectivity defined by arrays of indices. This fundamentally rigid way of looking at and representing data makes the traditional data model the less expressive and less efficient choice for high performance computing applications.

VTK-m allows for the much needed more exact representation, and with the burden of the traditional data model removed, VTK-m programmers can create more expressive data layouts. In fact, it is much easier to represent data types such as non-physical or high dimensional data in a VTK-m data model versus that in the traditional paradigm. Another important example of this efficiency is that VTK-m is designed to function with zero copy. This is an important motivator for in situ programming as VTK-m can utilize the data arrays from the simulation in place, saving both time and space.

In summary, the design directions taken by VTK-m are pushing the current boundaries of visualization from the multi-core realm into the many-core realm, prepping the visualization community for this inevitable transition. VTK-m is being developed as a header only library, which should ease integration issues when using VTK-m in situ, giving it great flexibility.

II-B2 Graphics in Support of Scientific Visualization

The creation of a graphics system that performs tasks in real-time is a challenging area of study for both graphic system designers as well as scientists employing new graphics algorithms in that space [41]. However, the challenges are justified, as visualization can be one of the most informative methods for communicating the essence of an experiment or data to scientists or the public [42], [43]. With ever increasing geometry and pixel counts, the task of employing an algorithm with a sufficient level of parallelism has become paramount. To that end, there are three basic classes of parallel rendering algorithms recognized in this space, sort-first, sort-middle, and sort-last rending. These algorithms each have been designed for applications in different domains. Sort-last rendering performs best when the geometry is massive compared to the pixel count, commonly seen in HPC visualization. Sort-first on the other hand, is the reverse of sort-last, performing best on low geometry counts with high pixel densities, commonly seen in virtual environment generation. Sort-middle is a hybrid approach that attempts to take the best elements form both sort-first and sort-last.

In this section, we will first describe a basic parallel graphics pipeline and the three techniques for geometry sorting, followed by a discussion of optimized algorithms for sort-last rendering, and a framework designed to composite images at massive scale. This analysis is important for when we move to discuss in situ visualization, as rendering can be a major bottleneck for in situ visualization tasks.

II-B2.1 A Parallel Graphics Pipeline

The heart of a parallel graphics pipeline can be viewed as a sorting problem, where the contribution of each object in a given view by each pixel must be determined. The location of this sort determines the entire structure of the resulting parallel algorithm. The sort can, in general, take place anywhere in the rendering pipeline: during geometry processing (sort-first), between geometry processing and rasterization (sort-middle), or during rasterization (sort-last). Sort-first means redistributing raw primitives (before their screen-space parameters are known). Sort-middle means redistributing screen-space primitives. Sort-last means redistributing pixels, samples, or pixel fragments [44]. Using any one of these choices leads to a completely different class of parallel rendering algorithms.

The pipeline in a parallel graphics system can be thought of as having two primary parts, geometry processing and rasterization (see Figure 2). Image geometry is generally parallelized by assigning each processor to a subset of the objects in the scene. Rasterization is often parallelized by assigning each processor a portion of the pixel calculations [44]. Each of these steps, depending on the algorithm, may incur redistribution costs as well. Image geometry may incur redistribution costs as volume data moves between nodes to facilitate interpolation of the assigned points, while rasterization may occur costs as local images are moved to facilitate their combination into a complete image [45].

Sort-first: In sort-first rendering, the primitives are distributed as early in the rendering pipeline as possible (during geometry processing) to the processors that will be performing the remainder of the calculations. This method is most often used when there is a very large pixel count (as compared to geometry), as screen regions are divided among the available

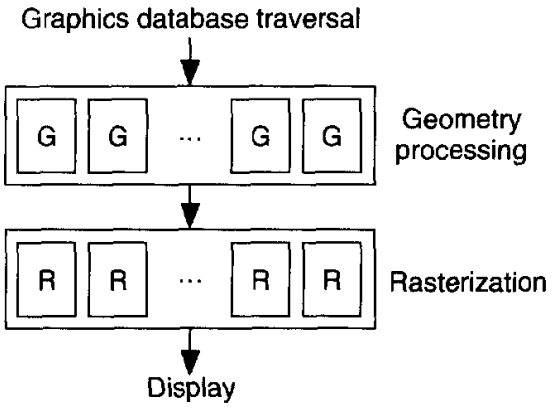


Fig. 2: Graphics pipeline in a fully parallel rendering system. Processors G perform geometry processing, while processors R perform rasterization (image from [44]).

processors (in essence parallelizing over the screen space).

These algorithms begin with each processor being assigned a region of the screen and taking an arbitrary portion of the data, and then beginning a transformation on that data. The transformation is applied until it can be determined to which portion of the scene that primitive falls (usually calculating the bounding box [44]). Once the scene space for all of the primitives are found, those that are located on processors to which they do not belong (according to the screen space that has been assigned to that processor), are redistributed over the network to the appropriate processors.

In summary, sort-first rendering is advantageous due to its low communication requirements when data primitives are sparse, and due to a single processor carrying out the entire pipeline for a portion of the screen. This method's drawbacks include its susceptibility to load imbalance when primitives clump into regions on the screen, giving certain processors much more work.

Sort-middle: In Sort-middle rendering, the data is redistributed in the middle of the rendering pipeline. At this stage, all primitives have been transformed into screen coordinates and are ready for rasterization [44]. Each frame is first transformed by the geometry processor, and then transmitted to the appropriate rasterizer (may or may not be the same processor depending on the implementation).

The general advantage of the sort-middle technique is its straightforward implementation, and the redistribution occurs at a natural place. The disadvantages are that it can have high communication costs and is susceptible to load imbalance when primitives are not evenly distributed across the screen.

Sort-last: The sort-last technique defers sorting until the end of the rendering pipeline. Each processor in this paradigm are assigned arbitrary subsets of the primitives [44]. Each of the processors computes pixel values for its subsets, irregardless of where they fall on the screen. This means that this algorithm scales well and gets a performance boost through the utilization of more and more processors [46]. At the end of the pipeline, pixels are transmitted over the network to be composited and their visibility resolved. It is at this point, however, that a bottleneck can develop. Interactive or real-time applications which rely heavily on the network to transmit all of the pixel data will suffer in performance due to the distributed pixels. Depending on the algorithm's implementation, this can be a major drawback in sort-last techniques.

In general, sort-last parallel rendering is the only proven way of parallel rendering at scale. This is mainly because the full rendering pipeline is carried out by individual processors until pixel merging is required. In addition, this approach is less prone to load imbalance. One disadvantage, however, is that the performance of sort-last parallel rendering drops sharply as the resolution of the display increases [47]. Furthermore, the final compositing step is generally regarded as *the* bottleneck for sort-last algorithms, so methods reducing the prevalence of this bottleneck will be of great value to scientific visualization at scale [48].

II-B2.2 Optimized Algorithms for Sort-Last Rendering

With sort-last rendering being the widely accepted choice for performing image compositing at scale, a lot of work has been done in creating algorithms in this space that are highly efficient. In this section, we will list a few of the most well known and used algorithms, as well as look at a piece of open source software that integrates some of the most recent advances in compositing algorithms.

Direct Send: In sort-last parallel rendering, the hardest task is the final image composting. Generally, n rendering channels will generate n full-size partial images, containing color and potentially depth [49]. These images must then be merged to form the final rendering. Direct send compositing divides the final image gathering task into n screen-space tiles to avoid exchanging full size images between the n processes. Each of the tiles is associated to a single channel for compositing, and at the end of the compositing process all of the partial tiles are assembled to form the final image.

Another strength of this algorithm are the number of synchronization points required. In this algorithm, only two synchronization points are needed, meaning less communication overhead on the system. Communication in this method does become a problem with larger geometries. The amount of data that must be transferred across the network is proportional to the rendering resolution as the pixels from each of the sub images must be sent across the network and finally composited. This process is particularly slow when using the TCP/IP stack. Eilemann suggests that this bottleneck can be reduced by using faster network technologies such as tunneling or asynchronous transfers [49], but the overall data transfer in this scenario still remains high, and as resolutions and data set sizes increase at a much higher rate than network speed, this bottleneck becomes a major obstacle.

Binary-Swap: The binary-swap method is an efficient and simple compositing algorithm that repeatedly splits the sub-

images and distributes them to the appropriate processor for compositing [50]. At every compositing stage, all processors participate by being paired with another processor, splitting their image plane in half, and each one taking responsibility for one half of the plane. This means that this method will take exactly $log(n)$ compositing stages to complete.

The idea behind binary-swap, is that only non-blank pixels affect the composited results, meaning that binary-swap exploits the sparsity of the sub-images by creating a bounding rectangle that exactly encompasses the non blank region in an image. The determination of this bounding rectangle takes $O(A)$ time, where A is the number of pixels. Most importantly however, once all of the bounding rectangles are determined, it only takes $O(1)$ time to merge two bounding rectangles, making updates to the bounding box of the composited image very efficient [50].

The primary problem of this method is the occurrence of load imbalance. Load imbalance may occur when the split of an image takes place in such a way that paired processors are given grossly different amounts of work. This means that one of the processors will have a much larger run time compared to its mate.

2-3 Swap: At its core, the 2-3 swap image compositing algorithm is a generalization of a binary-swap to an arbitrary number of processors [51]. This algorithm is derived from the observation that any integer greater than one can be decomposed into a summation of a list of twos and threes, meaning that the initial partition of processors in this algorithm can be done using combinations of twos and threes. In fact, it follows that if the number of processors is a power of two, then 2-3 swap essentially becomes a binary-swap in execution stage.

This algorithm is initially started by creation a tree of the number of given processors. Each non-leaf node in this tree has either two or three children, which determines the groups of processors during each stage of the image compositing algorithm. The initial work is evenly distributed among M participating processors in a group.

The primary pros of the 2-3 swap algorithm are that it is highly flexible and can utilize any number of processors for compositing, and each processor participates in all stages of compositing, giving maximum resource utilization.

Radix-K: Radix-K is a configurable algorithm for parallel image compositing [43], [52]. A unique aspect of Radix-K is its ability to overlap communication and computation, making this algorithm very customizable to the underlying hardware of a system.

In general, the Radix-k algorithm for image compositing builds on the previous contributions of binary-swap and direct send. By parameterizing the number of message partners in a round, it unifies these two algorithms by factoring the number of processes into a number of rounds with a separate radix for each round [43].

Improving Compositing Performance with IceT: Of the previous four algorithms, Radix-K is the leader in terms of work division. This algorithm performs highly parallel computation in conjunction with communication. The worst algorithm in terms of work division is direct send. Direct send is highly susceptible to work imbalance and suffers when it comes to having to communicate much larger segments to the final image. 2-3 swap and binary-swap are also susceptible to work imbalance, with 2-3 swap being more resilient. However, as stated previously, as Radix-K is able to communicate while running computation asynchronously, it mitigates imbalance and uses it to its advantage.

IceT, a leading production-quality image compositing framework, takes the problem of image compositing a step further, creating a testbed for enhancing these and other leading edge image compositing algorithms [48]. In this work, Moreland et al. found that not only were they able to create a testing ground for many different compositing algorithms simultaneously, but further, they were able to drastically improve compositing algorithms (Radix-K especially) while efficiently scaling to 64K cores. Their work demonstrates that image compositing still has room for improvement, and that through works like theirs, image compositing may soon scale efficiently for exascale sized runs. For more discussion of the challenges of scaling visualization tasks to exascale, see Section III-F1.

II-B3 The Importance of Color Map Choice in Visualization

A fundamental aspect of visualization is choice of color, usually through a color map. A color map is typically defined by mapping a range of numeric values to a continuum of colors [53]. The choice of color map however, is not just a matter of aesthetics, it must be carefully chosen to highlight the data and add extra insight [54].

Color choice for visualization is a well studied topic, yet many researchers and visualization tools still make poor choices. By making a poor choice in a color map, a researcher may get an exaggerated perspective of a data feature, or even worse, completely miss an important occurrence because the color scheme occluded it. One prime example of a bad color map is the rainbow color map [55].

There are several primary issues with the rainbow color map. First, the rainbow color map has no natural ordering. For example, the gray-scale color map has a definite ordering from dark to light, where contrarily, the colors red, green, yellow, and blue, have no apparent ordering. Second, the rainbow color map obscures data change because it is largely isoluminant, meaning that viewers mostly only see value changes at color boundaries. Lastly, the colors in the rainbow color map are poorly suited for those with vision deficiencies [56], those who cannot distinguish the differences between red and green colors.

The left image of Figure 3 shows an image colored with the traditional rainbow color map. Notice how the central green band of the rainbow color map consumes a large portion of the color scale. This property of the rainbow color map potentially hides many features within a data set as seen in the image on the right in Figure 3. This image is colored with a normalized rainbow color map, illuminating many more features of the data.

Color maps and color perception are a large and well

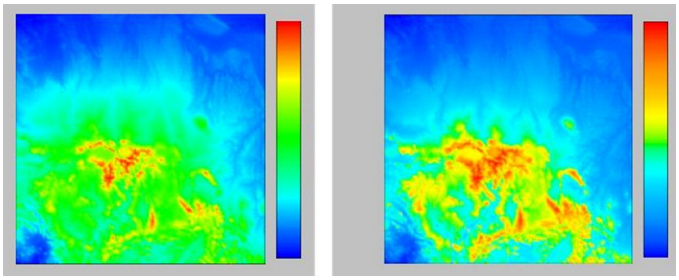


Fig. 3: Left, example of the data obscuring effects of the standard rainbow color map applied to topographic data. Right, normalized rainbow color map applied to the same data demonstrating better contrast between the different levels of the color scale (image from [54]).

studied topic that any visualization researcher needs to be aware of. The choice and use of color in visualization must be carefully chosen for the audience and type of data being displayed, or else the final visualization may end up being incomplete at best, or a misrepresentation at worst.

III IN SITU VISUALIZATION

Traditionally, scientific visualization has been performed as a post processing task. That is, all of the simulation data was written to persistent storage, and then after a simulation was complete, it would be analyzed and visualized. This model worked well up to the petascale era, but is beginning to break down [57]. The total amount of data that a supercomputer can generate with a simulation far surpasses its ability to write all of that data to persistent storage. For example, Figure 4 shows the current relative bandwidth of the total compute capability of the Titan supercomputer at Oak Ridge National Laboratory versus its storage bandwidth. The five orders of magnitude difference between the two demonstrate the intractability of writing all scientific data to disk prior to performing visualization. This reality leads to an alternative model for visualization that bypasses this limitation, termed in situ.

In situ as a technology is not new, with the earliest production-quality in situ graphics being seen as early as the 1960's [59]. Therefore, it is not surprising that several past surveys of in situ and in situ techniques have been published. In 1998 Heiland et al. [60] presented a survey of co-processing systems, which covered some of the basic use and availability of predominant co-processing frameworks. A year later in 1999, Mulder et al. [61] surveyed predominant computational steering environments, whose roots lie in in situ visualization and analysis. Recently in 2016, Ayachit et al. [62] and Bauer et al. [58] present two different takes on the state of in situ technology and challenges, as well as discussions of in situ frameworks. This section builds on the ideas presented in those surveys, and presents current in situ terminology, challenges, frameworks, and in situ research covering different motivations and use cases for in situ.

III-A In Situ Terminology

In situ visualization is an umbrella term used to describe many different visualization configurations where the visualization and analysis routines are run while the simulation is still in progress, reducing the amount of data that must be transferred over the network and saved to disk [63]. The visualization community has played fast and loose with the term in situ, and it has come to mean many different things. Clarifying terms have been introduced such as co-processing or tightly coupled, and concurrent processing or loosely coupled [12], but even these terms are starting to degrade. Current efforts are underway to bring the visualization community all onto the same page about terminology, with an effort termed the "In Situ Terminology Project." The terminology being developed in this report will go a long ways towards clarifying the meaning of in situ terms for the community and our stakeholders, but will not be presented here as the report is still under development. Instead, I will stick with the more loose and general terms currently in use by the community, and will make the switch to the new terminology set as it is introduced to the larger visualization community.

The terms I will stick to in this section are as follows:

- **In situ:** Umbrella term used to describe all different types of in situ setups.
- **Tight coupling:** We define tightly coupled to mean when the simulation and visualization code run in the same process using the same resources as the simulation.
- **Loose Coupling:** We define loosely coupled to mean when the simulation transfers data over the network to a separate set of visualization nodes for processing.
- **Hybrid Coupling:** We define hybrid coupling to mean when there are visualization components being run on the same process as the simulation and data is still being transferred over the network to separate visualization processes on a separate set of visualization resources.

For simplification as shown in Figure 5, we view the tight and loose coupling paradigms as on-node and off-node respectively. Tightly coupled can be thought of as running on the same node as the simulation, and not utilizing asynchronous data transfers from the simulation to the visualization routines, while loosely coupled can be viewed as on-node. Now that the definitions of in situ have been presented, we will present an overview of the challenges of using in situ techniques, and its barriers to adoption by the simulation community.

III-B In Situ Challenges and Opportunities

It has only been recently that some scientists have begun to see the need to adopt the in situ approach for visualization and analysis of large-scale simulations [63]. This hesitancy is due to essentially three primary factors. First, the traditional paradigm of post-hoc visualization has meant that scientists rarely had to use supercomputer time to perform their visualizations. The in situ paradigm would break this tradition, and scientists see visualization as a new cost and overhead to their science. Second, integrating in situ into a simulation has the potential to be a monumental task. In addition to the integration costs, the overhead of having visualization routines packaged into the simulation code in the tight coupling case can cause

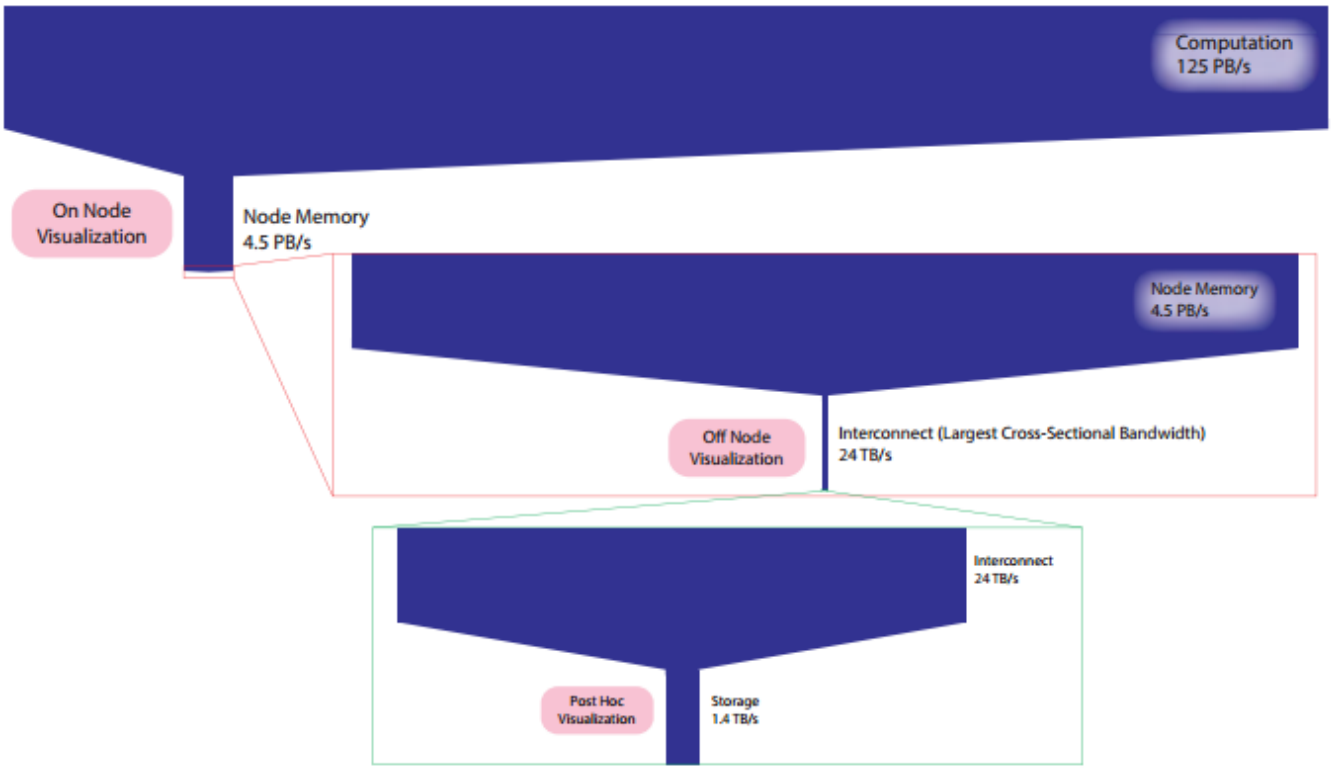


Fig. 4: A plot of the relative bandwidth of system components in the Titan supercomputer at the Oak Ridge Leadership Class Facility. The widths of the blue boxes are proportional to the bandwidth of the associated component. Multiple scales are shown to demonstrate the 5 orders of magnitude difference between the computational bandwidth and the storage bandwidth (adapted from [58]).

dependency issues between the simulation and visualization routines, while also bloating the size of the simulation binary. An additional side effect of this integration is the sharing of memory between the simulation and visualization routines, which can cause contention on compute nodes. Third, and the most challenging problem with in situ, is the need to know what to visualize a priori. This is, with in situ, it is required to know what to visualize including regions, values, as well as the type of visualization before the simulation starts.

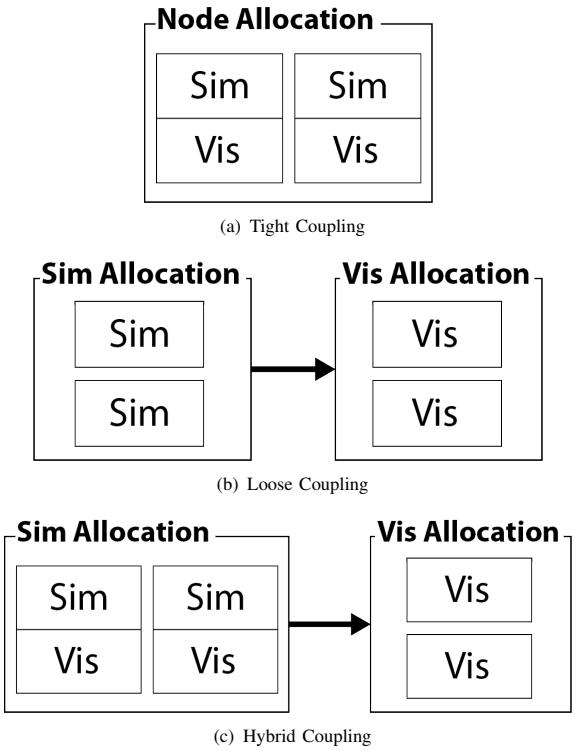
These problems may seem daunting at first glance, but the issues associated with each can be mitigated through different in situ implementations. Not all in situ work has to be about visualization. In fact, a great strength of in situ methods is the ability to access all of a simulations data during the course of a simulation, and only save what is interesting. This means, in situ is a great tool for visualization, but also for data manipulations such as data reductions, explorable feature extractions, simulation monitoring, and the generation of statistics [64]. Some example work in this area includes reducing data output to an alternate explorable form, computing collections of images, and storing images enhanced with fields and meta data for post hoc exploration.

An example of creating an reduced alternate data form is by Agranovsky et al. [65]. They describe a novel process for improved post hoc data exploration using particle advection. Instead of saving out vector fields every nth iteration, a basis trajectory is saved. A basis trajectory is a snapshot of a particle movement between the saved snapshots. This means that a representative set of particles are traced in situ while the simulation runs, and their trajectories are output. This technique allows for new particle trajectories to be interpolated between known trajectories, increasing both speed and accuracy.

Examples of computing collections of images for post hoc exploration comes from Yen et al. [66] and Chen et al. [67]. Yen et al. enable post hoc interaction with images through lighting and color transfer function changes, performing slices, and changing view. Chen et al. take the approach of visualizing a large sampling of possible visualization configurations in situ (various isocontour levels, different views, etc.), and then providing an interface to explore the collection interactively.

Finally, examples of generating images with enhanced meta data for post hoc exploration comes from Tikhonova et al. [68], [69] and Fernandes et al. [70]. Tikhonova et al. describe a method of storing layers of isosurface images that could later be composited together for post hoc exploration. Fernandes et. al. used a similar technique for volumetric renderings (saving areas of interest along with depth information) that could be explored post hoc.

The following three sections will present more in depth information about the three in situ techniques. They will discuss the strengths and weaknesses of each technique, provide a look at in situ frameworks in those categories, and give examples of past works performed using each paradigm to motivate the



(a) Tight Coupling

(b) Loose Coupling

(c) Hybrid Coupling

Fig. 5: Simulation and visualization resource configurations given the definitions of each of the in situ terms.

technique.

III-C Tightly Coupled In Situ

With tightly coupled in situ, in situ routines will directly share the same resources as a simulation. This has many different advantages and disadvantages. By sharing the same compute nodes the simulation and visualization codes will compete for memory, making the careful design of in situ routines critical with tight coupling. An inefficient or buggy implementation could slow the simulation, or worse, cause it to crash.

Further, by sharing the same resources, the in situ routine will be required to operate on the same level of concurrency as the simulation, which could cause slow performance with some in situ routines. Moreover, with this approach, the simulation code must wait for the in situ processing to complete after each simulation time step before it can carry on with computation [71]. This lock-step approach to computation is not attractive to many simulation scientists, which is part of their angst against using in situ techniques.

Even with these potential issues with tightly coupled in situ it is a widely used technique, with many different works taking advantage of data locality and computing power available to a full scale simulation.

III-C1 Tightly Coupled In Situ Frameworks

This section presents a look at tightly coupled in situ frameworks, and explores their features and restrictions. While many in situ frameworks have the potential to operate in several different modes, the frameworks presented here either operate fully or primarily in the tightly coupled model. For each framework we give a short description of functionality and categorize them according to the in situ methodologies they employ.

Cactus: Cactus [72], [73] is a development environment in which an application can be developed and run. In addition, Cactus has the capability of instrumenting legacy codes, to prevent the need for redesign within the Cactus framework. The remote visualization and data analysis capabilities of Cactus are achieved with tight coupling. Visualization operations are performed on the computational nodes, and the resultant geometry can then be sent to a remote viewer or saved to disk. An additional capability of Cactus, is that computational steering can be accomplished through the remote viewer, on predefined variables in the instrumented code.

CUMULVS: The Collaborative User Migration User Library for Visualization and Steering (CUMULVS) [74] is an infrastructure to allow multiple users the ability to monitor and steer of a simulation remotely. Users can connect and disconnect at will during the course of the running simulation. CUMULVS is capable of text and 2D output and visualization from an instrumented simulation. The original 2D visualization was supported through the use of AVS. A downside of the CUMULVS system is that it does not support the output of images, graphics are used purely for simulation monitoring.

ParaView Catalyst: ParaView Catalyst [32], [75] is the ParaView library which allows for in situ visualization of simulation output using the full visualization feature-set of ParaView, or subsets of features, by using reduced size binaries when minimal memory overhead to the simulation is required. Catalyst operates in a tightly coupled fashion, pausing the simulation while data operations take place.

Catalyst also allows for simulation steering and monitoring by connecting the Catalyst routines instrumented into the simulation to the ParaView application. This is a powerful feature that allows researchers to step through their code and dynamically modify visualizations based on the progress of the simulation.

In order to use Catalyst it must be instrumented into the simulation code, and an adapter needs to be written to define the interface between the simulation and Catalyst. This adapter defines how the simulation can call Catalyst as well as maps the simulation data to the VTK data model used by Catalyst. Catalyst has proven to be highly scalable, with the current largest run being on 256 thousand cores.

Strawman: Strawman [76] is a system designed to explore in situ visualization and analysis needs for physics codes on exascale architectures. An additional use for the infrastructure is as light weight prototyping environment for in situ analysis and visualization routines. This prototyping environment allows for fast implementations of in situ ideas. It uses Conduit [77] for a data model, EAVL/VTK-m for the visualization and analysis pipeline, and IceT [78] for parallel image compositing.

Strawman supports execution on many core environments, multiple programming languages, and works within a batch environment. It does limit its in situ focus however to tightly coupled methods, so does not support research loose coupling techniques. It does take advantage of this fact however, by enabling zero-copy of the data when possible, taking full advantage of the tightly coupled approach.

VisIO: VisIO [79] is an I/O library for use on distributed file systems within visualization applications. It includes a new scheduling algorithm to help preserve data locality within a simulation by assigning visualization intelligently to co-locate computation and data. The core of this framework revolves around the use of the Hadoop distributed file system in conjunction with a VisIO enabled reader in ParaView. One drawback of this approach is that it requires the use of the Hadoop file system, which could prove very time consuming to use in an existing application.

VisIt Libsim: VisIt Libsim [80] is the VisIt library which allows for in situ visualization of simulation output using the full visualization feature-set of VisIt. Libsim operates in a tightly coupled fashion, pausing the simulation while data operations take place. In fact, when the Libsim library is inserted into a simulation program, it makes each process of the simulation act much like a VisIt compute engine, operating in the same data space as the simulation.

One interesting feature that stems from the engine-viewer approach used in VisIt, is that the Libsim routines within the simulation listen for a request to connect by a VisIt process, meaning that users can connect and disconnect from the in situ routines as needed to perform periodic simulation steering or to check validity.

One drawback of the Libsim approach is that it requires instrumentation of the simulation code. Several calls need to be inserted into the simulation, as well as the Libsim binary itself. In some cases, if a simulation does not have a well-defined loop to simulate a single timestep, Libsim suggests restructuring of the simulation code.

Nevertheless, Libsim remains a powerful in situ visualization tool, largely due to the large array of visualization capabilities within the VisIt tool itself. It has also been shown to scale well, nearly as well as VisIt itself, up to 62 thousand cores [81].

III-C2 Related Work: Tightly Coupled In Situ

Implementations using tight coupling are often concerned most with the full utilization of a resource. That is, the desire is to run the simulation at the largest capacity possible, not reserving nodes for visualization or I/O. This implementation does have the advantage that the visualization routines have direct access to the full simulation output, and the full parallel capacity of the simulation machine. The following are several works that utilize tight coupling for visualization.

Yu et al. [82] demonstrate a tightly coupled system for volume rendering of jet fuel combustion data, in addition to a remote viewer application used to view the volume rendered images during the simulation run, as well as send requests for different viewing angles or transfer functions to the simulation code. The visualization code in their case was directly integrated into the simulation code, and worked off of pointers to the simulation results in order to reduce data duplication. As this system required the simulation to pause while visualization was taking place, it had a large effect on simulation runtime, with combined visualization and I/O times (from compositing) taking up to 4x more time than the simulation when done at every time step. This was reduced to two orders of magnitude less than the simulation time though, when the temporal fidelity was dropped to every ten time steps.

Woodring et al. [83] describe an in situ workflow for saving a simulation-time random sampling of large-scale particle datga from a cosmological simulation. Their workflow uses an extension of the kd-tree stratified random samping algorithm to generate level-of-detail output files for post hoc visualization. The level of detail approach is used in order to reduce storage bottlenecks and give them an integrated approximation error for their views. Using the kd-tree approach they are able to tune the output size to their specific needs by changing how many levels of the tree are written to disk, and show that at the lowest level of detail that they can write only 1/64th of the total simulation data to disk. This approach is useful in that it still allows for exploration of the data post-hoc, which is advantageous to static images.

Lorendeau et al. [84] describe a workflow using the Catalyst in situ visualization library for visualizing a computational fluid dynamics code. Catalyst is a ParaView library that defines in situ workflows using parallel VTK. In the described workflow the authors developed an adapter to their simulation worfklow for Catalyst and use it to perform their visualization operations. By introducing Catalyst they were able to perform their visualization operations in situ and save on the amount of data written to disk. They saw a 20 to 30% overhead associated with their initial implementation, but predict it can be reduced with better memory management in their adapter.

Ahrens et al. [85], [86] describes a tightly coupled system that takes the approach of saving many images from many angles from a simulation instead of writing simulation data to disk. The system is called ParaView Cinema. The idea is that if hundreds or thousands of images are created for a given timestep, that it will be possible to create an interactive database for a timestep that will allow interactive exploration much like that of VisIt or ParaView. In addition, this system has the capability of recreating a facsimile of the surface of the data based on the many saved images, letting different color maps and scalar fields be applied to the images during the post hoc exploration. The ParaView Cinema approach was demonstrated using a large-scale model for prediction across scales ocean simulation, and it was shown that the interactive database could be generated at twice the cost of generating an equal number of traditional tightly coupled in situ images. This cost may seem high, but the interactive database has a lot more functionality than a traditional image, allowing for the greater flexibility of post hoc exploration.

III-D Loosely Couple In Situ

Loosely coupled in situ offers many new configurations for visualization not seen with tight coupling. The most common

configuration is to have a set of dedicated visualization nodes on the same machine as the simulation, which reduces the effects of network latency that is seen when moving data to another machine. This separate allocation allows the visualization routines to run concurrently with the simulation, not impacting its runtime as with tightly coupled methods.

This benefit of a separate set of visualization nodes is also a primary downside of the loosely coupled visualization, as simulation scientists rarely want to give up portions of compute power for visualization tasks. Recently however, it has been shown that by streaming simulation data to an allocation of staging nodes, that the effects of disk latency can be hidden by staging the disk writes to the separate allocation, and letting them run while the simulation continues [87], [88], [89]. Given this, the approach of dedicating a set of the simulations nodes to staging becomes more palatable to simulation scientists, and further allows the introduction of in situ visualization techniques on that separate allocation, which can further benefit the simulation.

III-D1 Loosely Coupled In Situ Frameworks

This section presents a look at loosely coupled in situ frameworks, and explores their features and restrictions. While many in situ frameworks have the potential to operate in several different modes, the frameworks presented here either operate fully or primarily in the loosely coupled model. For each framework we give a short description of functionality and categorize them according to the in situ methodologies they employ.

EPIC: The Extract Plug-in Components Toolkit (EPIC) [90] is designed to create in situ data surface extracts from a running simulation. These extracts can be viewed in situ using a prototype version of FieldView, or extracts can be saved to disk. One downside of EPIC is that it requires the simulation to use the EPIC defined MPI communicator. This requirement could cause substantial integration issues for codes wishing to employ EPIC.

Freeprocessing: Freeprocessing [91], [92] is an in situ interposition library designed to reduced the barrier to entry for simulations to introduce in situ visualization. The premise is that many visualization codes avoid in situ technology as it has a large upfront cost for integration, and worse, if it requires direct manipulation of the simulation source code, it could have negative repercussions for performance and code stability. Freeprocessing has the ability to do loosely coupled visualization using staging nodes, in either a synchronous or asynchronous mode. Further, Freeprocessing can connect to existing visualization tools such as VisIt Libsim or ParaView Catalyst to take advantage of existing work in high performance visualization routines.

ICARUS: Initialize Compute Analyze Render Update Steer (ICARUS) [71] is a ParaView plug-in for in situ visualization and computational steering. It operates in the loosely coupled environment using a shared memory mapped HDF5 file for data access. It has minimal modification requirements for a simulation code, but only operates on the HDF5 file format. Simulation steering is accomplished through the use of the shared file interface, where each side can read and write from the files to pass steering messages.

pV3: Parallel Visual3 (pV3) [93], [94] is a parallel visualization system primarily targeted at computational fluid dynamics codes. It utilizes a client-server architecture, and has built in visualization capabilities. The client-server architectures allows the system to connect to an instrumented simulation at will. The pV3 system allows for computational steering, tightly coupled, and post-hoc visualization. pV3 is no longer under development.

III-D2 Related Work: Loosely Coupled In Situ

Past works that utilize loosely coupled in situ are most often concerned with the impact that visualization has on a running simulation. Works in this category often try to reduce the effect that visualization has on the simulation time as much as possible, and often do so by running on a separate allocation. The following are several different approaches to loosely coupled visualization.

Ellsworth et al. [95] describe a time-critical pipeline for weather forecasting using the GEOS4 simulation code. This code is run under very tight time constraints four times a day, which requires the visualization to be performed with minimal overhead. The visualization is achieved in this workflow by copying the simulation data to a separate shared memory segment where a discrete visualization system then accesses and operates on the data. This setup does require that the simulation be instrumented, and several new calls had to be added directly to the simulation code to redirect the output to the desired shared-memory segment. The resultant time-varying visualizations are then saved to disk or displayed on a tiled wall display.

Ma et al. [64] describe a visualization system for an earthquake simulation that uses a remote viewer over the wide are network to interactively change the visualization operations, view angles, color, etc. of rendering operations being done on the simulation machine itself. The integration of their visualization system requires that a simulation provide an API to access the internal data structures of the simulation, so the integration is visible from the perspective of the simulation scientist. However, this approach does limit the amount of integration needed compared to other more intrusive methods. The authors then demonstrated the viability of their system by interactively visualizing the results of a 2048 process simulation.

Pugmire et al. [96] introduce a visualization workflow that utilizes ADIOS to intercept the I/O calls of a simulation and stage the simulation data on a separate allocation of nodes. Their workflow then used EAVL to perform parallel visualization operations on the staged data, Mesa [97] to perform rendering, and IceT to perform parallel image compositing. Their experiments show that by incorporating Mesa and IceT into the parallel visualization environment EAVL, that they were able to further reduce the time to completion by between 5% and 14% versus an MPI compositor.

III-E Hybrid In Situ and Computational Steering

Hybrid methods [12] are composed of both tightly and loosely coupled components being utilized simultaneously. These methods support the flexibility of processing and reducing data on the simulation resources before they are either written to disk, or transferred to the visualization resource for additional processing. In other words, it offers the ability to achieve the best of both the tight and loose coupling paradigms.

Computational steering systems are methods related to hybrid in situ, as they allow a user to control all aspects of the computational science pipeline [98]. This control can range from simple monitoring controls to check that a simulation is in a valid state, to advanced controls that allow a user to step through a simulation and change key simulation variables while a simulation is in progress. One advantage of computational steering is that it can enable a user to steer a simulation back to a valid state, or stop an invalid simulation before computing time is wasted on invalid computations.

III-E1 Hybrid In Situ and Computational Steering Frameworks

This section presents a look at hybrid in situ frameworks, and explores their features and restrictions. For each framework we give a short description of functionality and categorize them according to the in situ methodologies they employ.

ADIOS: The Adaptable I/O System (ADIOS) [99], [100], is a componentization of the I/O layer used by high-end simulations and/or for high-end scientific data management, providing an easy-to-use programming interface, which can be as simple as file I/O statements. ADIOS abstracts the API away from implementation, allowing users to compose their applications without detailed knowledge of the underlying software and hardware stack. The ADIOS framework has been designed with a dual purpose: to increase the I/O throughput of simulations using well-known optimization techniques, and also to serve as the platform for introducing novel data management solutions for production-use without extensive modifications to the target applications.

ADIOS is used by a variety of mission critical applications running at DOE and NSF facilities, including combustion, materials science, fusion, seismology, and others. At the same time, ADIOS offers the community a framework for developing next generation I/O and data analytics techniques. Recent advances in this area include FlexIO [101], an infrastructure for the flexible placement of in situ analytics at different levels of the memory hierarchy, and PreDatA [102], a strategy for characterizing data while it is being generated in order to support faster data manipulations on staging resources.

To address the growing imbalance between computational capability and I/O performance, ADIOS introduced the concept of data staging, where rather than writing data directly to shared backend storage devices, a staging pipeline moves data to a transient location, on separate physical nodes and/or on memory resources on the same node where data is generated. Once on the *staging* nodes, data can be aggregated, processed, indexed, filtered, and eventually written out to persistent storage [103]. A key outcome of staging has been dramatic reductions in the total volume of data to be stored through the use of tightly coupled and loosely coupled data analytics. ADIOS contains a variety of transport methods for the movement of data, including DataSpaces [104], which allows memory coupling between processes running on different sets of nodes, FlexPath [105], which supports a publish/subscribe interface for direct memory access, and ICEE [106] which supports RDMA transfers over wide area networks.

Damaris/Viz: Damaris/Viz [107], [108] is an in situ framework based off of the I/O middleware framework Damaris [109]. Damaris/Viz was developed with the goals of having low impact on simulation runtime, low impact for in situ integration, and high adaptability. It achieves these goals by having low instrumentation costs. Visualization capabilities consist of user-defined modules, or connections to the VisIt Libsim or Paraview Catalyst interfaces. Damaris/Viz can operate in either a tightly coupled approach, utilizing a subset of cores on each simulation node, or loosely coupled, by using a dedicated set of visualization nodes.

EPSN: EPSN [110] is a library designed to provide a software environment for computational steering. There are two methods of interacting with EPSN, a lightweight network user interface, or through a distributed parallel visualization tool. The visualization and steering tools utilize VTK and IceT. EPSN has a client server relationship allowing multiple clients to connect and disconnect to the simulation on-the-fly.

GLEAN: GLEAN [111] is a non-intrusive framework for real time data analysis and I/O acceleration. It achieves this by being semantically aware of the data it is transporting, and by mitigating the variability of filesystem I/O performance through asynchronous data staging nodes using the network. GLEAN follows a similar model to ADIOS, and allows for custom data analyses to be performed on both the compute and staging resources. This model can mitigate the overall data saved to disk, improving application performance.

GLEAN supports both the tightly and loosely coupled in situ paradigms. Tightly coupled workflows are supported when GLEAN is embedded as part of the simulation, sharing the same address spaces and resources, and the simulation is semantically aware when it calls GLEAN. Loosely coupled workflows are supported when GLEAN asynchronously moves simulation data to a separate allocation of staging nodes though standard I/O libraries like HDF5.

Numerous performance studies exist using GLEAN, and it has been shown to be scalable and has drastically improved I/O performance on test codes that traditionally used HDF5 or pnetcdf. Overall, GLEAN is a powerful framework that requires minimal or no modifications to existing applications to implement, and can improve application performance on applications experiencing network bottlenecks. That is, simulation scientists can focus on simulation development, and let GLEAN focus on data transport enhancements, while also giving the simulation new opportunities to insert data analysis methods on both the simulation and data staging nodes.

Magellan: Magellan [112] is a framework for computational steering of a simulation. To instrument a code with Magellan it must be annotated to reveal specific steering parameters to the Magellan interface. This interface consists of two components, steering servers and steering clients. The steering client is a mechanism to interface with the steering servers and interactively change parameters. Magellan allows for multiple applications to be steered simultaneously, but is very limited in its graphical capabilities. It must be linked with outside visualization systems for the creation of visualizations. Magellan is no longer under development.

SCIRun: SCIRun [113] is a programming environment that allows for the construction, debugging, and steering of scientific computations. The computational steering aspect of SCIRun is one of its more highly developed aspects, allowing users to vary different aspects of a simulation while it is running. This interactivity is performed lock-step, so it follows the tightly coupled approach of stalling the simulation while it performs its steering and analysis. SCIRun is modular, so further extensions can be added through the modular interface.

SENSEI: SENSEI [59] is an effort to both streamline the in situ instrumentation of a scientific code and allow for flexibility in the choice of analysis infrastructure. This flexibility is achieved through the use of the underlying technologies that SENSEI employs. It allows for the use of both VisIt Libsim and ParaView Catalyst as visualization platforms, and either GLEAN or ADIOS for data staging. The analysis routines in SENSEI use the standard VTK data model for cross-platform compatibility.

SENSEI has even addressed some of the drawbacks of the VTK data model discussed earlier in section II-B1, by adapting the VTK data model to support structures-of-arrays, array-of-structures, and zero-copy.

To instrument a code with SENSEI, there are two adapters that need to be created. First, a data adapter API is created. This adapter is used to provide the analysis code with access to simulation mesh and array attributes. Second, an analysis adapter API is created. This adapter provides a concrete instance of an analysis adapter, which is a mechanism for interfacing with different in situ infrastructures. Figure 6 gives an overview of possible SENSEI instrumentation layouts. It is possible to perform both loosely coupled and tightly coupled analysis with this interface, with multiple options for staging and visualization technologies.

III-E2 Related Work: Hybrid In Situ

Past work in the area of hybrid in situ and computational steering often focus on making in situ more accessible to simulation teams, providing greater temporal locality of simulation visualizations, and providing a channel for the simulation team to interact with the running simulation directly. Some of the works presented below take advantage of the different frameworks presented above, while others roll their own approaches to specific simulation needs.

Past work in the area of simulation monitoring and steering has focused a lot of effort into designing methods for quickly

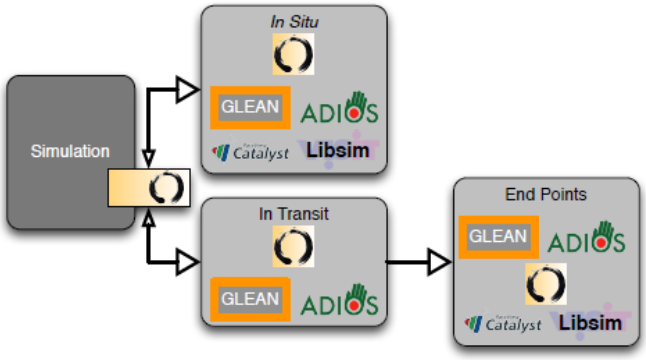


Fig. 6: A depiction of the SENSEI generic data interface for tight coupling, loose coupling, and hybrid implementations. It enables the dynamic choice of instrumentation technology depending on user circumstances though the use of its generic interface (adapted from [59]).

and efficiently visualizing data across a network. Some notable examples include Visapult [114], Visualization Dot Com [115], VisPortal [116], and a Real-Time Monitoring framework for large scientific simulations [117]. VisPortal and Visualization Dot Com build on the foundations of Visapult, and provide a remote distributed visualization framework for efficient visualization of remote simulation data. This framework uses both the local visualization client and the remote data client to perform parallel renderings, decreasing the time to produce the final visualizations. By leveraging Visapult, VisPortal and Visualization Dot Com are able to provide convenient access to simulation data to scientists through an easy to use and accessable online interface.

A different approach to simulation monitoring is the online dashboard. One successful instance of an online dashboard is eSimon [118], used for the XGC1 simulation. This dashboard was launched with each simulation run and was responsible for several different common visualization and analysis tasks in XGC1. First, the dashboard was responsible for creating and updating plots of approximately 150 different variables every 30 seconds and plotting 65 different planes for the live simulation. At the conclusion of a run, the dashboard would automatically output movies of each of these plots of interest for quick review. In addition, this dashboard cataloged simulation output allowing users to search for and retrieve data of interest, without having to locate and search through simulation output files. Finally, this dashboard was available to scientists anywhere in the world through their internet browsers. This approach to simulation monitoring is powerful, as it is easy-to-use from the point-of-view of the simulation scientist and is easy to access.

Moving on now to works on visualization, we look at a few works utilizing ADIOS. ADIOS is an enabling technology, and a number of past visualization works have taken advantage of the easy integration and data transfer and translation capabilities of the platform. Some recent examples include work by Bennett et al. [119], Pugmire et al. [120], and Kress et al. [121].

The work by Bennett et al. makes the insight that many analysis algorithms can be formulated to perform various

amounts of filtering and aggregation, resulting in intermediate data that can be orders of magnitude smaller than simulation output. They put this insight into practice by creating a two stage pipeline using a combustion simulation, in which data is first filtered and reduced on the simulation nodes before being transffered to a staging area using ADIOS. Once in the staging area they performed topological analysis, gathered descriptive statistics, and performed visualization. They validated this approach at moderate scale showing that it was possible and fast to perform these operations in a hybrid fashion.

The work by Pugmire et al. focused on the development of scalable visualization plugins that operate within the data staging of ADIOS. They show the creation of an interactive visualization system which utilizes the RDMA transfer capabilities of ADIOS for data transport, and VisIt for visualization. ADIOS would send subsets of data requested by the visualization client to a visualization cluster where VisIt scripts would operate on the data, with the final results being viewed by a remote visualization client. Figure 7 shows the result of a visualization using their system, which is the visualization of a turbulent eddy and its accompanying particles within the fusion simulation code XGC1.

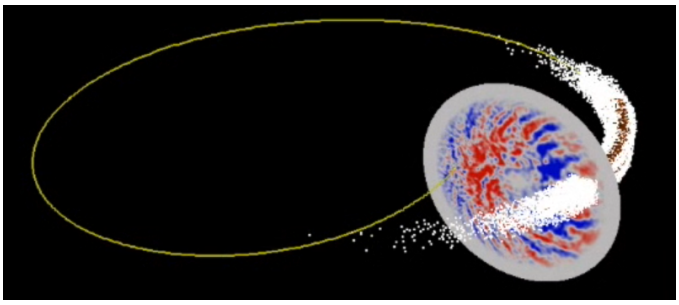


Fig. 7: The VisIt interface window demonstrating particle tracking by ID of particles that were inside a 3D eddy at a particular timestep in the past (from [120]).

Kress et al. focused primarily on data reduction using ADIOS and a separate analysis node allocation. Their premise is that at exascale, simulation data reduction will be required in order to gain a reasonable temporal view for visualizations. They present two different types of data reductions that can be done in staging by altering the underlying data representations. One interesting approach they present is representing data with reduced precision formats. That is, simulations are typically over-resolved, so for visualization it is not necessary to maintain full precision, and they demonstrate that visualizations are comparable at different digits of precision. They caution however, that data reduction must be done with domain knowledge. Data features may be lost when doing visualizations of derived variables.

A further example that does not utilize ADIOS is by Vishwanath et al. [122]. They describe a test of the GLEAN framework on an adaptive mesh hydrodynamics code, in which they increased I/O speed and computed fractal dimensions of the data as it was being written to disk. In this work, they were able to instrument the simulation code without adding anything to the simulation code itself, instead the I/O libraries already in use by the simulation were instrumented to use GLEAN. Through their tests they say that it was much faster to compute the fractal dimensions in situ versus their traditional post hoc approach, and that they were able to increase I/O speed between 10-117x vs HDF5 and pnetcdf.

A more basic example not utilizing a framework is by Buffat et al. [123]. They describe a client-server system for in situ analysis of computational fluid dynamics. Their workflow has the capability of performing computational steering, and can use VisIt Libsim for remote visualization. The core of their workflow is a separate allocation of nodes where the visualization tasks take place in Python, and the data is asynchronously transffered to this allocation from the simulation using MPI.

III-F Comparing In Situ Paradigms

Now that the two basic and one hybrid in situ coupling variants have been discussed, it is important to begin to consider the costs and benefits that each scenario brings to a simulation code. In order to have concrete comparisons on which to base judgments on the costs and benefits of each scenario, I will present ten in situ comparison factors [124]. These comparison factors were selected to span the range of issues relevant to both scientists that are running simulations, and computer science researchers and developers that are deploying analysis and visualization methods. These factors consider required HPC resources (both shared and dedicated), impact on the running simulation, fault tolerance, and usability.

III-F1 In Situ Comparison Metrics

Data Access: With simulations producing more data than can be saved to disk, a different data set is available for visualization and analysis depending on when the data is accessed. Generally speaking, there is more data and time steps available on the simulation resources than there will be once the data is transferred and saved to disk. Thus, the correct set of operations should be performed on the data at each stage. For operations that require all data and all time steps, that operation should be performed on the simulation nodes before data is culled. However, if an operation or simulation team can handle performing analysis on a sparser data set, such an operation could take place after data is saved to disk.

With tightly coupled in situ, visualization and analysis routines can take advantage of having the full richness of the simulation output. Operations exist that take into account all of the produced data for every time step.

On the other hand, loosely coupled in situ visualization routines must often operate with a sparser set of data. However, it should be noted that this data set can be more complete than those that are saved to disk because the network transfer can allow for a greater volume of data to be sent. Therefore, loosely coupled in situ routines often work with less data than is available in situ, but more than is available post hoc.

Data Movement: Moving large quantities of data from one location to another can be an expensive task. The cost of this task varies substantially depending on where the data is being sent, i.e., between nodes in an allocation or off node over the network, so data movement should be kept to a minimum.

Often the amount of data needed varies by the visualization algorithm employed. For a simulation using tightly coupled in situ visualization and analysis, the amount of data moved can range from none to simulation-stalling levels. This is because some visualization algorithms traditionally require large amounts of data to be sent between the ranks, which complicates the problem when using tightly coupled in situ. Communicating between every node in the simulation can be enormously expensive compared to a smaller node allocation.

Loosely coupled in situ visualization has a different issue with regards to data movement. Before loosely coupled in situ visualization can take place, the data must be sent from the simulation to a visualization resource for processing. This dump from the simulation to the visualization resource can saturate the network, and could even cause a slowdown in the simulation while it sends the data over the network. This data dump has the potential to move far less data, in total, during the visualization routine as compared to that of tightly coupled in situ. This is due to visualization allocations traditionally being much smaller than simulation node allocations, meaning that communication takes place over a much smaller domain.

Data Duplication: At the conclusion of each time step of a simulation, a new set of data is available and ready for use. On node resources may take immediate advantage of this data, while off node resources require a copy to be made. The act of making this copy means that the data now exists in two places, doubling the memory footprint.

Tightly coupled in situ visualization does not have a data duplication problem. All data is already available within the simulation, so no duplication will take place.

Loosely coupled in situ visualization must work on a copy of the data by definition. That is, the data is copied from the simulation nodes to whatever loosely coupled in situ visualization solution is being used. This duplication now doubles the RAM usage for each time step, possibly making it the less efficient choice.

Data Translation: Simulation codes store mesh and field data in myriad ways that visualization programs must be able to interpret and work with. The foundation for performing such a translation is a data model (which describes what data can be represented) and its implementation (which describes how to lay out arrays).

In the in situ world, there are two basic options. First, the visualization code can allocate new arrays that match its own data model implementation and then copy data from the simulation code's arrays into its own arrays. Obviously, this memory bloat is often viewed as undesirable. However, this approach is still used in VisIt's LibSim and ParaView's Catalyst. The second option is to ensure that the visualization code can work directly on the simulation data layout. This is straightforward when writing custom code specifically for that simulation, but much harder when trying to design a general-purpose visualization infrastructure that can be re-used with many simulation codes. The approaches used by the community so far involve redirection of data accesses through virtual functions (done in some cases with Catalyst), designing a data model implementation that supports many different array organizations to increase the chances that the simulation code uses an array layout that the visualization code can support (as with EAVL), or writing templated code customized to the simulation code during the compilation process (as with SciRun).

To date, the two basic options have proven difficult for doing easy and overhead-free data translation. Instead, we note that this problem has been addressed previously for data I/O, where simulation codes write arrays to disk and visualization codes read them. Establishing schemas, interfaces, and conventions was a non-trivial task in this space, but one that is now generally considered "solved." With respect to in situ, the loosely coupled approach can take advantage of this existing solution by using the simulation code's I/O calls as a way to pass data. As a result, the path to integrating in situ technology with the loosely coupled approach is significantly less of a burden.

Coordination: Coordination is required between the simulation and the visualization. This coordination lets the visualization know that the next iteration of simulation data is ready and that visualization can begin.

In a tightly coupled in situ paradigm, coordination is minimal. If visualization code is directly embedded into the simulation, this could be as simple as calling the visualization routine at the end of the simulation main loop. For production tools like LibSim and Catalyst, the coordination is very similar, but the call is made into the particular library.

In a loosely coupled in situ paradigm, much more coordination is required. At the end of each cycle in the main loop a call must be made to transfer the data to the visualization resource. This transfer requires use of the network and coordination on both the sending and receiving side to ensure the data is successfully sent and received. To guard against faults, care must be taken to recover from situations when a network call fails, or the visualization resource is not available.

Resource Requirements: All in situ paradigms require additional resources of some sort. In a tightly coupled in situ paradigm the simulation and visualization share the same resources, including execution, memory, and network. In an era when memory per core is steadily decreasing, visualization tools are required to operate under very tight memory restrictions. In cases where intermediate results need to be computed and held in memory, this can be a challenge. Additionally, supercomputing time is in high demand and very expensive. Therefore, simulations will generally dedicate a fixed window of time for visualization. These restrictions place challenges on visualization, which generally run on dedicated resources with large memory, or on the development of new techniques that operate within tight time and memory requirements.

In a loosely coupled in situ paradigm, additional visualization nodes are required. These additional nodes are requested at the time the simulation is run, increasing the cost of running a simulation. However, these additional nodes can be used asynchronously once the data is transferred. The visualization can run while the next time step is being computed by the simulation, and there are no restrictions on memory usage. However, care must be taken to handle the arrival of the next

time step, if the visualization routines are still running. But otherwise, the restrictions are minimal.

Exploratory Visualization: Exploratory visualization, a task most associated with post-processing of data on disk, is generally not a strength in any in situ paradigms. Typically, the visualization must be specified a priori, and so care must be taken to decide when the simulation is launched and which particular operations will be performed. However, tools like LibSim and Catalyst do allow fully-featured visualization tools access to specified parts of simulation data, making free-form exploratory visualization possible, but at the expense of pausing the simulation while the user interacts with the data.

Scalability: Any in situ paradigm is constrained to use the concurrency of the allocated resources. In a tightly coupled paradigm, this is the allocation for the entire simulation. While this level of concurrency might be advantageous for embarrassingly parallel routines that require little synchronization or communication, it can be a bottleneck for visualization routines that require significant communication (e.g., particle tracking, etc) or algorithms that don't exhibit scaling up to the levels of simulation codes (e.g., hundreds of thousands of cores). Conversely, in a loosely coupled paradigm, the concurrency of the visualization resource can be appropriately configured for the tasks to be performed. Algorithms that require significant synchronization and communication will generally perform much better at lower levels of concurrency, and this can be used to optimize performance.

Fault Tolerance: As supercomputers continue to grow in size and complexity, resilience and fault tolerance at all levels become increasingly important. For tightly coupled in situ paradigms where visualization and simulation run together, fault tolerance becomes imperative. Simulations are directly exposed to data corruption, infinite loops, and errors in visualization routines, and could result in faults or crashes. Because of the expense of supercomputing time, and the drastic impact of faults on simulation codes, fault tolerance is a requirement. In practice, this is difficult to achieve.

Because of the clear and distinct separation between the simulation and the visualization in a loosely coupled paradigm, the exposure to faults is greatly reduced. In this paradigm, the data transfer to the visualization resource becomes the only point of exposure to faults. The exposure can be further reduced by using asynchronous transfers.

Ease of Use: Usability spans a wide range of topics, and includes things such as integration, deployment, development, and dependencies. For tightly coupled in situ where there is a fundamental connection between the simulation and visualization code, software engineering practices become very important. Because of this basic interdependence, changes in either the simulation or visualization code, or dependencies on third party libraries, need to be carefully managed. In the case of stand-alone production packages where there is a more separated interface point, careful coordination of releases and patches is still required.

TABLE III: Summarization of the benefits of both the tightly and loosely coupled in situ paradigms. The paradigm which is strongest in a given category is indicated with a check mark, and a dash is used when the paradigms are equally as good.

		Favored Paradigm	
	Comparison Factor	**Tightly Coupled**	**Loosely Coupled**
Data Factors	Data Access	✓	
	Data Movement	—	—
	Data Duplication	✓	
	Data Translation		✓
	Exploratory Visualization	—	—
Implementation Factors	Scalability		✓
	Ease of Use		✓
	Coordination	✓	
	Fault Tolerance		✓
	Resource Requirements	—	—

For loosely coupled in situ, the interface between the simulation and visualization takes place through the API. Here, a cleanly defined, concise, and small set of APIs determine the usability of the system.

Finally, there is no free lunch. Development costs must be taken into account. While writing custom visualization code has the advantage of maintaining full control and making domain-specific optimizations easy, there is the cost of not taking advantage of community-wide investments devoted to making standard tools and libraries. On the other hand, developing loosely coupled in situ frameworks is a large undertaking, and providing the flexibility to handle a wide variety of uses cases is a challenge.

III-F2 Summarizing In Situ Paradigm Strengths

Based on the evaluation of the 10 factors we discussed previously, there are clearly very good reasons for using each of the techniques. Table III summarizes the evaluation metrics, and which in situ paradigm is best in that area. In cases with very specific needs, there is often a clear choice of in situ method. In practice however, there are generally many factors under consideration, and the optimal in situ approach will be situationally dependent.

Tightly coupled approaches will work very well when the simulation has a predefined list of images and analyses that it needs produced. These can be directly coded into the simulation. We emphasize that if a visualization task can be tightly coupled, it should be.

On the other hand, if interactive exploration is required, if subsets of data should be saved for further analysis, or an open source visualization solution needs to be employed for data visualization, loose coupling might be the best approach. These approaches avoid many of the pitfalls of being fault intolerant, they are generally easier to deploy and maintain, they generally scale better, and the data translation from the simulation representation to the visualization library representation is solved through existing I/O calls.

Since both paradigms are strong under varying circumstances, in an ideal world, there would be a model that

would predict for a given visualization task whether in situ visualization (including which paradigm) is viable or not given a list of computational, time, and resource constraints. This is an area that merits further research, and is an area that interests me. A summary of research in performance modeling for in situ visualization is presented in Section IV.

IV PERFORMANCE MODELING FOR IN SITU APPLICATIONS

In general, performance modeling is the process of simulating various user and system loads for specific tasks through the use of a mathematical approximation of how the tasks work. This mathematical approximation provides users with the applicability or viability of the specific task given the system loads and constraints. This type of approximation is needed for in situ applications. That is, the ability to predict if a given in situ visualization task is viable given a set of user and system constraints would be useful for the community as they schedule and prioritize in situ tasks in workflows.

IV-A Approaches to Performance Modeling

There are three main approaches to performance modeling, as well as one combination of approaches. The approaches are (1) analytic, (2) heuristic/statistical, (3) history-based methodologies [125], and (4) the combination approach, or semi-empirical [126]. Each of these methods has sets of strengths and weaknesses, and have been used extensively in the past.

Analytic Models: Analytic performance models take the approach of reducing a set of steps to a mathematical process that describes the cost of the steps, and can quickly generate performance predictions for known input parameters. Examples of analytic performance models and tools include PANORAMA [127], Aspen [128], LogP [129], and the PERC framework [130].

PANORAMA is an extension of the Aspen domain specific performance modeling language, and is designed to model the run-time performance of complex scientific workflows. It incorporates task-level profiling and monitoring using Aspen, and can predict runtimes of different tasks.

LogP was designed to aid in the development of fast and portable parallel algorithms. It models computation based on four main parameters that abstract the computing bandwidth, communication bandwidth, communication delay, and the efficiency of the coupling of communication and computation. The designers chose these four parameters to capture the breadth of the computation process, while keeping the model as simple as possible.

The PERC Framework combines tools used for gathering machine profiles and application signatures to model underlying application performance. They base their model on the assumption that an applications performance is dominated by two overarching factors: the single processor performance and its use of the network. These assumptions simplify their model and requires only the understanding of single processor performance in combination with the network performance model to approximate an application's performance

Heuristic/Statistical Models: Heuristic and statistical-based modeling techniques model the underlying features of a target system using a variety of different modeling levels from the microprocessor instructions, cache use, and network messaging. Examples in this area include GEMS [131], ROSS [132], PACE [133], SST [134], [135], and performance and power modeling [136].

The General Execution-driven Multiprocessor Simulator (GEMS) is a series of modules for timing the memory and microprocessor subsystems of a machine. This simulator creates detailed timings of processor performance and cache use. This product is highly limited due to its complexity of use and limited forms of release, making it difficult to use and slow to create models.

The Rensselaer's Optimistic Simulation System (ROSS) is a modular system for simulating network performance. ROSS is fast and accurate, outperforming other similar network simulators. The downsides of this system are that it only models network performance, requires many different parameters for estimation, and has long runtimes compared to analytical performance models.

The Performance Analysis and Characterization Environment (PACE) is a layered system for modeling software, hardware, and parallelism. It requires the creation of a specialized workload definition using a domain specific language to capture the characteristics of a given application. Once completed, it will model the fine-grained features of an application and overall system performance to generate runtime predictions.

The Structural Simulation Toolkit (SST) is a simulator that estimates the performance of large-scale parallel machines. There are two primary ways to utilize SST. First, an MPI trace from a previous simulation run can be replayed in the simulator, allowing estimates to be generated for new architectures and configurations. Second, a skeleton application can be provided which mimics the control and messaging pattern of a real application. Together, these approaches allow this toolkit to do cycle-specific analysis of an application and its performance. The downside of this approach is that it takes a very fine-grained view of performance modeling, and is more time intensive.

The work on performance and power monitoring for parallel scientific applications focuses on creating detailed statistics on performance and power use during a simulation. It accomplishes its power modeling by leveraging on-chip performance counters for CPU and GPU systems. The performance modeling is aided by the TAU Parallel Performance System [137] which is capable of instrumenting and analyzing the performance of parallel applications. This system enables application tuning through iterative application runs, where performance data is collected followed by application tuning to address the bottlenecks shown in the collected performance data.

History-based Models: History-based performance models rely on instrumenting a series of past runs of a specific workflow task and gathering performance traces. The traces are then aggregated into a numerical model that can approximate a known modeled task [138]. One good example of this type of performance prediction is from the Large Hadron

Collider [139]. The approach given in this work looks at a long history of completed runs of a specific analysis task, and can estimate the job runtime of that task within 75% accuracy.

Semi-empirical Models: Semi-empirical models are combinations of parts from both the heuristic and analytical performance modeling approaches. The idea is that known hardware parameters (e.g., network performance rates, disk read/write speed) and empirical measurements like execution time or performance counters are combined with the analytical approach to create a hybrid model. This hybrid model bridges the gap in terms of complexity from the heuristic to the analytical approach and has the potential to be as accurate using less resources and evaluation factors. One such work in this space focuses on in situ rendering, and creates accurate performance models for ray tracing, volume rendering, and rasterization [140].

Summary of Performance Modeling: Comparing these four different paradigms, we can see a large difference in the complexity of gathering the data for creating the performance models. The history-based methods in particular are not feasible at massive scale and require many different simulation runs using different simulation configurations to develop predictions. Worse, history-based models do not transfer between machines of different configurations and require the whole history process to be done again on new machines at full-scale.

Heuristic-based models can be extremely accurate, but getting cycle-specific information for a simulation is a time intensive process, and just as with the history-based methods, it needs to be repeated on each new machine. These types of models are not feasible for exascale simulations due to enormous costs.

The class of performance models for exascale applications that shows promise are the analytical models. Analytical models reduce the complexity of a simulation down to a set of overarching factors that determine the performance of an application. This reduction means models for single processor performance can be combined with models of network performance to get a holistic picture of an applications performance without running the full application. New models will still need to be created for different architectures, but these models will require much smaller runs requiring less time than a full simulation run. Combine these types of performance models with some known heuristics, and the resultant semi-empirical model is even more powerful.

IV-B Performance Modeling for In Situ Visualization

There are no known works that focus on performance modeling for in situ visualization workflows. This is where my interest and dissertation work lies. In order to do performance modeling for in situ visualization, two different pieces of information need to be combined to create overall models. First, studies need to be done to determine what data will look like on exascale machines in terms of size, frequency, and variety. This data is important for performance models because it will enable us to create models based on current data trends with an eye on the future. Second, for in situ tasks that require visualization, a total model is needed that covers the entire in situ pipeline, including models for in situ rendering and different types of visualization operations (this is where my interest and dissertation work lies).

IV-B1 Cataloging Large-scale Simulation Visualization and Analysis Tasks

We know of only one work that focused specifically on cataloging and categorizing the different visualization and analysis tasks of a simulation code. That work took an in depth look at the XGC1 [141] simulation code through a series of interviews with developers. Those interviews cataloged the different visualization and analysis tasks and requirements within XGC1, as well as the different simulation input and output types and sizes [20]. This work found a wide breadth of output sizes from the simulation with large variations in temporal data availability. In addition, many of the visualization requirements proposed by the simulation team required very high fidelity from the largest simulation outputs. These requirements from the simulation team point to the real need for in situ visualization and analysis for large simulations, and highlight the need for in situ researchers to be able to provide concrete answers about the performance and viability of their in situ routines at the temporal fidelity required by researchers.

This in-depth work provides a valuable look at one simulation code team and their projections for exascale, however, it would be valuable to conduct this type of research with more simulation teams. Short of that, there are instances of specific visualization and analysis requirements being reported in conjunction with a study, which also provide valuable insight when preparing for performance modeling.

A work by Bennett et al. [119] reports on a use case with combustion simulations using S3D, where features are tracked, identified, and visualized both in situ and in transit. Their work utilized in situ and in transit methods using a volume of nearly 1 billion cells and 16 seconds average wall time per time step using 4896 cores. Through a combination of in situ and in transit methods they were able to perform the feature tracking and identification use case within their time budget, while reducing the amount of data transferred over the network and to disk.

Pugmire et al. [142] explore a feature tracking and identification use case in the XGC1 simulation code, using a data set of nearly 1 billion particles and a time budget of 10 seconds per simulation time step. In this work, the authors describe a system that intelligently handles the tracking of particles and features of a simulation in real time in a user specified area of interest.

Ellsworth et al. [95] describe a time-critical pipeline for weather forecasting using the GEOS4 simulation code. This code is run under very tight time constraints four times a day, which requires the visualization to be performed with minimal overhead. The visualization was performed on data consisting of 23 million cells with up to seven 3D and four 2D fields per cell.

Malakar et al. [143] describe a series of visualization tasks done with the LAMMPS simulation code. The data contained 1 billion atoms, using 91 GB per simulation time step. Typical

runs consisted of 1000 time steps, with output every 100 time steps.

Slawinska et al. [144] demonstrate the incorporation of ADIOS into Maya for an astrophysics simulation workflow. Using in situ techniques, they reduced the amount of data needed to perform their visualization and analysis task from 4.5 TB down to 24 GB that would normally be saved to disk without in situ.

Dungque et al. [145] report on work done with computation fluid dynamic data sets, visualizing helicopter rotor and wind turbine movement. Their work utilized 18 million grid points and data sets in excess of 60 GB per five revolutions. Their final visualizations averaged 140,000 triangles, 45,000 quads, and 121,000 vertices.

Two additional works focusing on the S3D simulation code report large gains in visualization and analysis accuracy by incorporating in situ techniques [82], [146]. Both papers utilized similar data sets containing $1.3x10^9$ grid points, 22 variables, and a simulation checkpoint size of 140 GB. Landge et al. [146] show that in situ techniques enable feature extraction and tracking every 50 simulation time steps instead of every 500 as was traditional with negligible overhead added to the simulation.

From these past works, we have been able to get a sense of some of the data sizes and visualization and analysis requirements from other large-scale simulation codes. Through the combination of all of these works, we can get a sense of both the breadth and depth of the needs of these codes in terms of data movement and usage. This information will be valuable for the completion of the overall in situ visualization performance model.

IV-B2 Rendering Performance Modeling

A recent work by Larsen et al. [140] developed a performance model specifically for in situ rendering. This performance model was designed to provide an a priori answer for the cost of using an in situ rendering approach. It accomplished this through the use of statistical performance models based on algorithmic complexity, and was able to accurately predict the runtime cost of a set of rendering algorithms.

One example of a performance model from this work is for volume rendering. They determined that the time to perform volume rendering could be characterized by three primary variables (1) Active Pixels (AP), (2) Cells Spanned (CS), and (3) Samples Per Ray (SPR). By their definition, active pixels was the number of pixels that were updated as a result of rendering. Cells spanned was the maximum number of cells that a ray could span. Samples per ray was the average number of samples along a ray that were inside of the data set. The resultant model was:

$$T_{VR} = c_0 * (AP * CS) + c_1 * (AP * SPR) + c_2 \quad (1)$$

This model was then combined with a compositing model they created in order to predict rendering times in a distributed setting. Overall, this model was very accurate, predicting the runtime of many consecutive rendering tasks within a 10%-20% on average.

One example question that was addressed by the paper using the models developed was how many images could be generated on different architectures using each of the three rendering methods within a 60 second time budget. Figure 8 shows the trends in the number of images generated for both the CPU and GPU architectures using their predictive models. Being able to ask concrete questions like this one using a performance model is a valuable contribution to the in situ visualization community, allowing researchers to optimize their workflows to meet desired performance and visualization goals within specified time budgets.

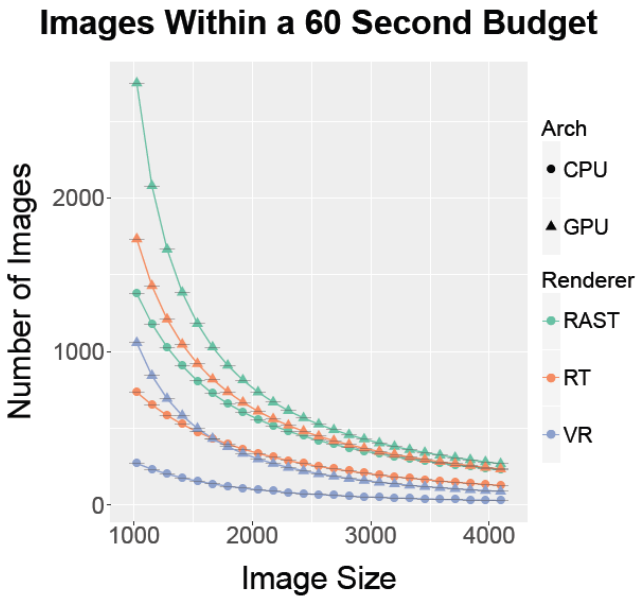


Fig. 8: Predictions using each of the performance models developed by Larsen et al. to determine how many images could be generated by each rendering technique given a 60 second budget on each of the two different test architectures (from [140]).

V CONCLUSION AND FUTURE WORK SUMMARY

In this survey, we have presented the current research in important areas to in situ visualization and analysis. This included a background look at high performance computing, visualization, and graphics and rendering, which are critical areas to understand when developing and working with in situ visualization frameworks. This was followed by an in-depth look at the state-of-the-art of in situ systems and performance modeling.

REFERENCES

[1] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, J Ahrens, EW Bethel, H Childs, et al. Scientific discovery at the exascale. *Report from the DOE ASCR 2011 Workshop on Exascale Data Management*, 2011.

[2] Roger Blandford, Young-Kee Kim, Norman Christ, et al. Challenges for the understanding the quantum universe and the role of computing at the extreme scale. In *ASCR Scientific Grand Challenges Workshop Series, Tech. Rep*, 2008.

[3] W Washington. Challenges in climate change science and the role of computing at the extreme scale. In *Proc. of the Workshop on Climate Science*, 2008.

[4] Glenn Young, David Dean, Martin Savage, et al. Forefront questions in nuclear science and the role of high performance computing. In *Technical report, ASCR Scientific Grand Challenges Workshop Series*, 2009.

[5] William Tang, David Keyes, N Sauthoff, N Gorelenkov, J Cary, A Kritz, S Zinkle, J Brooks, R Betti, W Mori, et al. Scientific grand challenges: Fusion energy sciences and the role of computing at the extreme scale. In *DoE-SC Peer-reviewed report on major workshop held March*, pages 18–20, 2009.

[6] Robert Rosner, Ernie Moniz, et al. Science based nuclear energy systems enabled by advanced modeling and simulation at the extreme scale. In *ASCR Scientific Grand Challenges Workshop Series, Tech. Rep*, 2009.

[7] Guilia Galli, Thom Dunning, et al. Discovery in basic energy sciences: The role of computing at the extreme scale. In *ASCR Scientific Grand Challenges Workshop Series, Tech. Rep*, 2009.

[8] Rick Stevens, Mark Ellisman, et al. Opportunities in biology at the extreme scale of computing. In *ASCR Scientific Grand Challenges Workshop Series, Tech. Rep*, 2009.

[9] A Bishop, P Messina, et al. Scientific grand challenges in national security: The role of computing at the extreme scale. In *ASCR Scientific Grand Challenges Workshop Series, Tech. Rep*, 2009.

[10] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Comput. Graph. Appl.*, 30(3):22–31, May 2010.

[11] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Visualization at extreme scale concurrency. In E. Wes Bethel, Hank Childs, and Charles Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, Boca Raton, FL, 2012.

[12] Hank Childs, Ma Kwan-Liu, Yu Hongfeng, Whitlock Brad, Meredith Jeremy, Favre Jean, Klasky Scott, Podhorszki Norbert, Schwan Karsten, Wolf Matthew, Parashar Manish, and Zhang Fan. In situ processing. In E. Wes Bethel, Hank Childs, and Charles Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, Boca Raton, FL, 2012.

[13] JR Neely. The us government role in hpc: Technology, mission, and policy. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2014.

[14] Salman Habib, Robert Roser, Richard Gerber, Katie Antypas, Katherine Riley, Tim Williams, Jack Wells, Tjerk Straatsma, A Almgren, J Amundson, et al. Ascr/hep exascale requirements review report. *arXiv preprint arXiv:1603.09303*, 2016.

[15] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, W Carson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[16] Top 500d. https://www.top500.org/lists/2016/06/. Accessed: 2016-10-25.

[17] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[18] G Aloisioa, S Fiorea, Ian Foster, and D Williams. Scientific big data analytics challenges at large scale. *Proceedings of Big Data and Extreme-scale Computing (BDEC)*, 2013.

[19] Kenneth Moreland. Oh, $#*@! exascale! the effect of emerging architectures on scientific discovery. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 224–231. IEEE, 2012.

[20] James Kress, David Pugmire, Scott Klasky, and Hank Childs. Visualization and analysis requirements for in situ processing for a large-scale fusion simulation code. In *To Appear: Proceedings of the Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), held in conjunction with SC16*, Salt Lake, UT, November 2016.

[21] James Ahrens, Bruce Hendrickson, Gabrielle Long, Steve Miller, Rob Ross, and Dean Williams. Data-intensive science in the us doe: case studies and future challenges. *Computing in Science & Engineering*, 13(6):14–24, 2011.

[22] Hank Childs, Berk Geveci, Will Schroeder, Jeremy Meredith, Kenneth Moreland, Christopher Sewell, Torsten Kuhlen, and E Wes Bethel. Research challenges for visualization software. *Computer*, (5):34–42, 2013.

[23] Kenneth Moreland, Berk Geveci, Kwan-Liu Ma, and Robert Maynard. A classification of scientific visualization algorithms for massive threading. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 2. ACM, 2013.

[24] Craig Upson, TA Faulhaber, David Kamins, David Laidlaw, David Schlegel, Jefrey Vroom, Robert Gurwitz, and Andries Van Dam. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[25] David Thompson, Jeff Braun, and Ray Ford. *OpenDX: paths to visualization; materials used for learning OpenDX the open source derivative of IBM's visualization Data Explorer*. Visualization and Imagery Solutions, 2004.

[26] Will J Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit*. Kitware, 2004.

[27] William J Schroeder, Kenneth M Martin, and William E Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Proceedings of the 7th conference on Visualization'96*, pages 93–ff. IEEE Computer Society Press, 1996.

[28] James Ahrens, Berk Geveci, and Charles Law. Visualization in the paraview framework. In Chuck Hansen and Chris Johnson, editors, *The Visualization Handbook*, pages 162–170, 2005.

[29] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pages 357–372. Oct 2012.

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

[31] T Kuhlen, R Pajarola, and K Zhou. Parallel in situ coupling of simulation with a fully featured visualization system. 2011.

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

[33] Jeremy S Meredith, Sean Ahern, Dave Pugmire, and Robert Sisneros. EAVL: the extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 21–30. The Eurographics Association, 2012.

[34] Jeremy S Meredith, Robert Sisneros, David Pugmire, and Sean Ahern. A distributed data-parallel framework for analysis and visualization algorithm development. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, pages 11–19. ACM, 2012.

[35] K. Moreland, U. Ayachit, B. Geveci, and Kwan-Liu Ma. Dax toolkit: A proposed framework for data analysis and visualization at extreme scale. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 97–104, Oct 2011.

[36] Li-ta Lo, Christopher Sewell, and James P Ahrens. Piston: A portable cross-platform framework for data-parallel visualization operators. In *EGPGV*, pages 11–20, 2012.

[37] Christopher Sewell, Jeremy Meredith, Kenneth Moreland, Tom Peterka, Dave DeMarle, Li-ta Lo, James Ahrens, Robert Maynard, and Berk Geveci. The sdav software frameworks for visualization and analysis on next-generation multi-core and many-core architectures. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 206–214. IEEE, 2012.

[38] Nathan Bell and Jared Hoberock. Thrust: A productivity-oriented library for cuda. *GPU computing gems Jade edition*, 2:359–371, 2011.

[39] Kenneth Moreland, Christopher Sewell, William Usher, Lita Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots,

Kwan-Liu Ma, Hank Childs, Matthew Larsen, Chun-Ming Chen, Robert Maynard, and Berk Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A)*, 36(3):48–58, May/June 2016.

[40] Guy E Blelloch. *Vector models for data-parallel computing*, volume 356.

[41] Carl Mueller. The sort-first rendering architecture for high-performance graphics. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 75–ff. ACM, 1995.

[42] T Todd Elvins. A survey of algorithms for volume visualization. *ACM Siggraph Computer Graphics*, 26(3):194–201, 1992.

[43] Tom Peterka, David Goodell, Robert Ross, Han-Wei Shen, and Rajeev Thakur. A configurable algorithm for parallel image-compositing applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 4. ACM, 2009.

[44] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *Computer Graphics and Applications, IEEE*, 14(4):23–32, 1994.

[45] Ulrich Neumann. Communication costs for parallel volume-rendering algorithms. *Computer Graphics and Applications, IEEE*, 14(4):49–58, 1994.

[46] Don-Lin Yang, Jen-Chih Yu, and Yeh-Ching Chung. Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers. *The Journal of Supercomputing*, 18(2):201–220, 2001.

[47] Kenneth Moreland, Brian Wylie, and Constantine Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 85–92. IEEE Press, 2001.

[48] Kenneth Moreland, Wesley Kendall, Tom Peterka, and Jian Huang. An image compositing solution at scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2011.

[49] Stefan Eilemann and Renato Pajarola. Direct send compositing for parallel sort-last rendering. In *Proceedings of the 7th Eurographics conference on Parallel Graphics and Visualization*, pages 29–36. Eurographics Association, 2007.

[50] Akira Takeuchi, Fumihiko Ino, and Kenichi Hagihara. An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. *Parallel Computing*, 29(11):1745–1762, 2003.

[51] Hongfeng Yu, Chaoli Wang, and Kwan-Liu Ma. Massively parallel volume rendering using 2–3 swap image compositing. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11. IEEE, 2008.

[52] Wesley Kendall, Tom Peterka, Jian Huang, Han-Wei Shen, and Robert Ross. Accelerating and benchmarking radix-k image compositing at large scale. In *Proceedings of the 10th Eurographics conference on Parallel Graphics and Visualization*, pages 101–110. Eurographics Association, 2010.

[53] Kenneth Moreland. Why we use bad color maps and what you can do about it. In *To Appear In Proceedings of Human Vision and Electronic Imaging*, 2016.

[54] Samuel Silva, Beatriz Sousa Santos, and Joaquim Madeira. Using color in visualization: A survey. *Computers & Graphics*, 35(2):320–333, 2011.

[55] David Borland and Russell M Taylor II. Rainbow color map (still) considered harmful. *IEEE computer graphics and applications*, 27(2):14–17, 2007.

[56] Kenneth Moreland. Diverging color maps for scientific visualization. In *International Symposium on Visual Computing*, pages 92–103. Springer, 2009.

[57] Pak Chung Wong, Han-Wei Shen, Christopher R Johnson, Chaomei Chen, and Robert B Ross. The top 10 challenges in extreme-scale visual analytics. *IEEE computer graphics and applications*, 32(4):63, 2012.

[58] Andrew C Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, et al. In situ methods, infrastructures, and applications on high performance computing platforms. In *Computer Graphics Forum*, volume 35, pages 577–597. Wiley Online Library, 2016.

[59] Utkarsh Ayachit, Andrew Bauer, Earl P. N. Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E. Jansen, Burlen Loring, Zarija Lukić, Suresh Menon, Dmitriy Morozov, Patrick O'Leary, Reetesh Ranjan, Michel Rasquin, Christopher P. Stone, Venkat Vishwanath, Gunther H. Weber, Brad Whitlock, Matthew Wolf, K. John Wu, and E. Wes Bethel. Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 79:1–79:12, Piscataway, NJ, USA, 2016. IEEE Press.

[60] Randy Heiland and M Pauline Baker. A survey of co-processing systems. *CEWES MSRC PET Technical Report*, pages 98–52, 1998.

[61] Jurriaan D Mulder, Jarke J van Wijk, and Robert van Liere. A survey of computational steering environments. *Future generation computer systems*, 15(1):119–129, 1999.

[62] Utkarsh Ayachit, Andrew Bauer, Earl PN Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E Jansen, Burlen Loring, Zarija Lukić, Suresh Menon, et al. Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 79. IEEE Press, 2016.

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

[64] Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, and Anna Tikhonova. In-situ processing and visualization for ultrascale simulations. In *Journal of Physics: Conference Series*, volume 78, page 012043. IOP Publishing, 2007.

[65] Alexy Agranovsky, David Camp, Christoph Garth, E Wes Bethel, Kenneth I Joy, and Hank Childs. Improved post hoc flow analysis via lagrangian representations. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 67–75. IEEE, 2014.

[66] Yucong Ye, Robert Miller, and Kwan-Liu Ma. In situ pathtube visualization with explorable images. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, pages 9–16. Eurographics Association, 2013.

[67] Jerry Chen, Ilmi Yoon, and Wes Bethel. Interactive, internet delivery of visualization via structured prerendered multiresolution imagery. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):302–312, 2008.

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

[69] Anna Tikhonova, Hongfeng Yu, Carlos D Correa, Jacqueline H Chen, and Kwan-Liu Ma. A preview and exploratory technique for large-scale scientific simulations. In *EGPGV*, pages 111–120, 2011.

[70] Oliver Fernandes, David S Blom, Steffen Frey, Alexander H Van Zuijlen, Hester Bijl, and Thomas Ertl. On in-situ visualization for strongly coupled partitioned fluid-structure interaction. In *Coupled Problems 2015: Proceedings of the 6th International Conference on Computational Methods for Coupled Problems in Science and Engineering, Venice, Italy, 18-20 May 2015*. CIMNE, 2015.

[71] Marzia Rivi, Luigi Calori, Giuseppa Muscianisi, and Vladimir Slavnic. In-situ visualization: State-of-the-art and some use cases. *PRACE White Paper; PRACE: Brussels, Belgium*, 2012.

[72] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf. The cactus framework and toolkit: Design and applications. In *International Conference on High Performance Computing for Computational Science*, pages 197–227. Springer, 2002.

[73] Cactus. .http://cactuscode.org/. Accessed: 2016-11-27.

[74] J Kohl. Cumulvs. http://www.csm.ornl.gov/cs/cumulvs.html. Accessed: 2016-11-27.

[75] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the*

First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, pages 25–29. ACM, 2015.

[76] Matthew Larsen, Eric Brugger, Hank Childs, Jim Eliot, Kevin Griffin, and Cyrus Harrison. 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*, pages 30–35. ACM, 2015.

[77] Conduit. https://www.github.com/LLNL/conduit. Accessed: 2016-11-27.

[78] Kenneth Moreland. Icet users' guide and reference. Technical Report 2011-5011, Sandia National Laboratory, 2011.

[79] Christopher Mitchell, James Ahrens, and Jun Wang. Visio: Enabling interactive visualization of ultra-scale, time series data via high-bandwidth distributed i/o systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 68–79. IEEE, 2011.

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

[81] Brad J Whitlock, Steve M Legensky, and Jim Forsythe. In situ infrastructure enhancements for data extract generation. In *54th AIAA Aerospace Sciences Meeting*, page 1928, 2016.

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

[83] Jonathan Woodring, J Ahrens, J Figg, Joanne Wendelberger, Salman Habib, and Katrin Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Computer Graphics Forum*, volume 30, pages 1151–1160. Wiley Online Library, 2011.

[84] Benjamin Lorendeau, Yvan Fournier, and Alejandro Ribes. In-situ visualization in fluid mechanics using catalyst: A case study for code saturne. In *LDAV*, pages 53–57, 2013.

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

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

[87] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. Datastager: scalable data staging services for petascale applications. *Cluster Computing*, 13(3):277–290, 2010.

[88] Arifa Nisar, Wei-keng Liao, and Alok Choudhary. Scaling parallel i/o performance through i/o delegate and caching system. In *2008 SC-International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2008.

[89] Kenneth Moreland, Ron Oldfield, Pat Marion, Sebastien Jourdain, Norbert Podhorszki, Venkatram Vishwanath, Nathan Fabian, Ciprian Docan, Manish Parashar, Mark Hereld, et al. Examples of in transit visualization. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, pages 1–6. ACM, 2011.

[90] Earl P Duque, Daniel E Hiepler, Robert Haimes, Christopher P Stone, Steven E Gorrell, Matthew Jones, and Ron Spencer. Epic–an extract plug-in components toolkit for in situ data extracts architecture. In *22nd AIAA Computational Fluid Dynamics Conference*, page 3410, 2015.

[91] Thomas Fogal, Fabian Proch, Alexander Schiewe, Olaf Hasemann, Andreas Kempf, and Jens Krüger. 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*, volume 2014, page 49. NIH Public Access, 2014.

[92] Freprocessing. https://www.github.com/tfogal/freeprocessing. Accessed: 2016-11-27.

[93] Robert Haimes. Alaa 94-0321 pv3: A distributed system for large-scale unsteady cfd visualization. 1994.

[94] R Haimes. pv3. 3http://raphael.mit.edu/pv3/. Accessed: 2016-11-27.

[95] David Ellsworth, Bryan Green, Chris Henze, Patrick Moran, and Timothy Sandstrom. Concurrent visualization in a production supercomputing environment. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):997–1004, 2006.

[96] David Pugmire, James Kress, Jeremy Meredith, Norbert Podhorszki, Jong Choi, and Scott Klasky. Towards scalable visualization plugins for data staging workflows. In *Big Data Analytics: Challenges and Opportunities (BDAC-14) Workshop at Supercomputing Conference*, 2014.

[97] The mesa 3d graphics library. http://www.mesa3d.org/. Accessed: 2016-11-27.

[98] Christopher Johnson, Steven G Parker, Charles Hansen, Gordon L Kindlmann, and Yarden Livnat. Interactive simulation and visualization. *Computer*, 32(12):59–65, 1999.

[99] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, Manish Parashar, Nagiza Samatova, Karsten Schwan, Arie Shoshani, Matthew Wolf, Kesheng Wu, and Weikuan Yu. Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.

[100] Scott Klasky, Hasan Abbasi, Jeremy Logan, Manish Parashar, Karsten Schwan, Arie Shoshani, Matthew Wolf, Sean Ahern, Ilkay Altintas, Wes Bethel, et al. In situ data processing for extreme-scale computing. *Scientific Discovery through Advanced Computing Program (SciDAC11)*, 2011.

[101] Fang Zheng, Hongbo Zou, J Cao, J Dayal, T Nugye, G Eisenhauer, and S Klasky. Flexio: location-flexible execution of in-situ data analytics for large scale scientific applications. In *Proceedings IEEE international parallel and distributed processing symposium (IPDPS13)*, pages 320–331, 2013.

[102] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. Predata–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[103] David A Boyuka, Sriram Lakshminarasimham, Xiaocheng Zou, Zhenhuan Gong, John Jenkins, Eric R Schendel, Norbert Podhorszki, Qing Liu, Scott Klasky, and Nagiza F Samatova. Transparent in situ data transformations in adios. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 256–266. IEEE, 2014.

[104] Ciprian Docan, Manish Parashar, and Scott Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[105] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID '14)*, 2014.

[106] Jong Y Choi, Kesheng Wu, Jacky C Wu, Alex Sim, Qing G Liu, Matthew Wolf, C Chang, and Scott Klasky. 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.

[107] Matthieu Dorier, Roberto Sisneros, Tom Peterka, Gabriel Antoniu, and Dave Semeraro. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *LDAV-IEEE Symposium on Large-Scale Data Analysis and Visualization*, 2013.

[108] Damaris/viz. http://damaris.gforge.inria.fr/doku.php. Accessed: 2016-11-27.

[109] Damaris. http://damaris.gforge.inria.fr/. Accessed: 2016-11-27.

[110] Epsn. http://www.labri.fr/projet/epsn/. Accessed: 2016-11-27.

[111] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E Papka. 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*, page 19. ACM, 2011.

[112] Jeffrey Vetter and Karsten Schwan. High performance computational steering of physical simulations. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 128–132. IEEE, 1997.

[113] Steven G Parker and Christopher R Johnson. Scirun: a scientific programming environment for computational steering. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 52. ACM, 1995.

[114] Wes Bethel. Visapult: A prototype remote and distributed visualization application and framework. *Lawrence Berkeley National Laboratory*, 2000.

[115] Wes Bethel. Visualization dot com. *IEEE Computer Graphics and Applications*, 20(3):17–20, 2000.

[116] Wes Bethel, Cristina Siegerist, John Shalf, Praveenkumar Shetty, TJ Jankun-Kelly, Oliver Kreylos, and Kwan-Liu Ma. Visportal: Deploying grid-enabled visualization tools through a web-portal interface. *Lawrence Berkeley National Laboratory*, 2003.

[117] Valerio Pascucci, Daniel E Laney, Ray J Frank, Giorgio Scorzelli, Lars Linsen, Bernd Hamann, and Francois Gygi. Real-time monitoring of large scientific simulations. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 194–198. ACM, 2003.

[118] Roselyne Tchoua, Jong Choi, Scott Klasky, Qing Liu, Jeremy Logan, Kenneth Moreland, Jingqing Mu, Manish Parashar, Norbert Podhorszki, David Pugmire, et al. Adios visualization schema: A first step towards improving interdisciplinary collaboration in high performance computing. In *eScience (eScience), 2013 IEEE 9th International Conference on*, pages 27–34. IEEE, 2013.

[119] Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9. IEEE, 2012.

[120] David Pugmire, James Kress, Jong Choi, Scott Klasky, Tahsin Kurc, Randy Michael Churchill, Matthew Wolf, Greg Eisenhower, Hank Childs, Kesheng Wu, et al. Visualization and analysis for near-real-time decision making in distributed workflows. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1007–1013. IEEE, 2016.

[121] James Kress, Randy Churchill, Scott Klasky, Mark Kim, Hank Childs, and David Pugmire. Preparing for in situ processing on upcoming leading-edge supercomputers. *To Appear: Supercomputing frontiers and innovations*, 2016.

[122] Venkatram Vishwanath, Mark Hereld, Michael E Papka, Randy Hudson, G Cal Jordan IV, and C Daley. In situ data analysis and i/o acceleration of flash astrophysics simulation on leadership-class system using glean. In *Proc. SciDAC, Journal of Physics: Conference Series*, 2011.

[123] Marc Buffat, Anne Cadiou, Lionel Le Penven, and Christophe Pera. In situ analysis and visualization of massively parallel computations. *International Journal of High Performance Computing Applications*, page 1094342015597081, 2015.

[124] James Kress, Scott Klasky, Norbert Podhorszki, Jong Choi, Hank Childs, and David Pugmire. Loosely coupled in situ visualization: A perspective on why it's here to stay. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, pages 1–6, New York, NY, USA, 2015. ACM.

[125] Ian Bowman, John Shalf, Kwan-Liu Ma, and Wes Bethel. Performance modeling for 3d visualization in a heterogeneous computing environment. *Lawrence Berkeley National Laboratory*, 2004.

[126] Torsten Hoefler, William Gropp, William Kramer, and Marc Snir. Performance modeling for systematic performance tuning. In *State of the Practice Reports*, page 6. ACM, 2011.

[127] Ewa Deelman, Christopher Carothers, Anirban Mandal, Brian Tierney, Jeffrey S Vetter, Ilya Baldin, Claris Castillo, Gideon Juve, Dariusz Król, Vickie Lynch, et al. Panorama: An approach to performance modeling and diagnosis of extreme-scale workflows. *International Journal of High Performance Computing Applications*, page 1094342015594515, 2015.

[128] Kyle L Spafford and Jeffrey S Vetter. Aspen: a domain specific language for performance modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 84. IEEE Computer Society Press, 2012.

[129] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. Logp: Towards a realistic model of parallel computation. In *ACM Sigplan Notices*, volume 28, pages 1–12. ACM, 1993.

[130] Allan Snavely, Laura Carrington, Nicole Wolter, Jesus Labarta, Rosa Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 21–21. IEEE, 2002.

[131] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[132] Christopher D Carothers, David Bauer, and Shawn Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.

[133] AM Alkindi, Darren J Kerbyson, Efstathios Papaefstathiou, and Graham R Nudd. Run-time optimization using dynamic performance prediction. In *International Conference on High-Performance Computing and Networking*, pages 280–289. Springer, 2000.

[134] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, R Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.

[135] Curtis L Janssen, Helgi Adalsteinsson, and Joseph P Kenny. Using simulation to design extremescale applications and architectures: programming model exploration. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):4–8, 2011.

[136] Van Bui, Boyana Norris, Kevin Huck, Lois Curfman McInnes, Li Li, Oscar Hernandez, and Barbara Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance*, page 6. ACM, 2008.

[137] Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[138] Rich Wolski, Neil T Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5):757–768, 1999.

[139] Rafael Ferreira da Silva, Mats Rynge, Gideon Juve, Igor Sfiligoi, Ewa Deelman, James Letts, Frank Würthwein, and Miron Livny. Characterizing a high throughput computing workload: The compact muon solenoid (cms) experiment at lhc. *Procedia Computer Science*, 51:39–48, 2015.

[140] Matthew Larsen, Cyrus Harrison, James Kress, David Pugmire, Jeremy S. Meredith, and Hank Childs. Performance Modeling of In Situ Rendering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*, pages 24:1–24:12, Salt Lake City, UT, November 2016.

[141] CS Chang, S Ku, PH Diamond, Z Lin, S Parker, TS Hahm, and N Samatova. Compressed ion temperature gradient turbulence in diverted tokamak edge). *Physics of Plasmas (1994-present)*, 16(5):056108, 2009.

[142] David Pugmire, James Kress, Hank Childs, Matthew Wolf, Greg Eisenhauer, Randy Churchill, Tahsin Kurc, Jong Choi, Scott Klasky, Kesheng Wu, Alex Sim, and Junmin Gu. Visualization and analysis for near-real-time decision making in distributed workflows. In *High Performance Data Analysis and Visualization (HPDAV) 2016 held in conjuction with IPDPS 2016*, May 2016.

[143] Preeti Malakar, Venkatram Vishwanath, Todd Munson, Christopher Knight, Mark Hereld, Sven Leyffer, and Michael E Papka. 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, page 52. ACM, 2015.

[144] Magdalena Slawinska, Michael Clark, Matthew Wolf, Tanja Bode, Hongbo Zou, Pablo Laguna, Jeremy Logan, Matthew Kinsey, and Scott Klasky. A maya use case: adaptable scientific workflows with adios for general relativistic astrophysics. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, page 54. ACM, 2013.

[145] Earl PN Duque and Steve M Legensky. Visualization of large-scale unsteady computational fluid dynamics datasets. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 73–73. IEEE, 2005.

[146] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jiann-Jong Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 1020–1031. IEEE, 2014.