Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay

James Kress
University of Oregon &
Oak Ridge Nat'l Laboratory
james@jameskress.com

Scott Klasky, Norbert Podhorszki, Jong Choi
Oak Ridge Nat'l Laboratory
klasky@ornl.gov, pnorbert@ornl.gov, choij@ornl.gov

Hank Childs
University of Oregon &
Lawrence Berkeley Nat'l Laboratory
hank@cs.uoregon.edu

David Pugmire
Oak Ridge Nat'l Laboratory
pugmire@ornl.gov

ABSTRACT

In this position paper, we argue that the loosely coupled in situ processing paradigm will play an important role in high performance computing for the foreseeable future. Loosely coupled in situ is an enabling technique that addresses many of the current issues with tightly coupled in situ, including, ease-of-integration, usability, and fault tolerance. We survey the prominent positives and negatives of both tightly coupled and loosely coupled in situ and present our recommendation as to why loosely coupled in situ is an enabling technique that is here to stay. We then report on some recent experiences with loosely coupled in situ processing, in an effort to explore each of the discussed factors in a real-world environment.

CCS Concepts

•Human-centered computing → Visualization; Visualization theory, concepts and paradigms; *Visualization application domains;* **•General and reference** → Reference works;

Keywords

Visualization techniques and methodologies; Scientific visualization; In situ; Tightly coupled in situ; Loosely coupled in situ; perspective

1. INTRODUCTION

The amount of data available to scientists is quickly outpacing the ability to move, process, analyze and thereby fully comprehend. One of the fundamental barriers to knowledge extraction from scientific data are the increasing costs of data movement, particularly moving data to disk [1]. There appears to be no end to this trend as recently deployed supercomputers, as well as those planned for the future, provide far more computational capacity than I/O bandwidth.

Because visualization is particularly sensitive to I/O bandwidth [5, 6] the community has turned to in situ techniques to alleviate this growing problem. Broadly speaking, two paradigms have emerged [4]. First, *co-processing*, or *tightly coupled* methods. We define tightly coupled to mean when the simulation and visualization code run in the same process using the same resources. Second, *concurrent-processing*, or *loosely coupled* methods. We define loosely coupled to mean when the simulation transfers data over the network to a separate set of visualization nodes for processing. For simplification, we view these two paradigms as on-node and off-node. 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 off-node.

There has been significant work and successes with both paradigms [9, 12, 16, 19, 20]. As a result, it is unclear which paradigm simulation code groups and visualization software developers should back.

The remainder of this position paper is organized as follows: Section 2 compares and contrasts both paradigms against the following set of 10 factors: data access, data movement, data duplication, data translation, coordination, resource requirements, exploratory visualization, scalability, fault tolerance, and ease of use. Section 3 presents our perspective for why loosely coupled in situ visualization is best suited for future and current work using these paradigms. Section 4 provides a motivating use case of tightly coupled in situ in practice. Section 5 presents our final thoughts on terminology and the long-term benefits of loosely coupled in situ.

2. COMPARISON FACTORS

The comparison factors selected were intended 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 tol-

ISAV 2015, November 15-20, 2015, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2828612.2828623

erance, and usability.

2.1 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 are accessed. Generally speaking, there are more data and timesteps available on the simulation resources than there will be once the data are transfered and saved to disk. This makes it important that the correct set of operations are performed on the data at each stage. For operations that require all data and all timesteps, that operation should be performed on the simulation nodes before data are culled. However, if an operation or simulation team can handle performing analysis on a sparser data set, that operation could take place after data are 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 can be done that take into account all of the produced data for every timestep.

Loosely coupled in situ visualization routines on the other hand, often must 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.

Winner: tightly coupled in situ

2.2 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 are being sent, i.e. between nodes in an allocation or off 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 off over the network. This data dump though has the potential to end up moving far less data, in total, during the visualization routine vs. 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.

Winner: draw

2.3 Data Duplication

At the conclusion of each time step of a simulation, a new set of data are 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 are 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 are 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 timestep, possibly making it the less efficient choice.

Winner: tightly coupled in situ

2.4 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 on 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 support 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 that is customized to the simulation code during the compilation process (as with SciRun).

To date, the two basic options have proven to be 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.

Winner: loosely coupled in situ

2.5 Coordination

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

In a tightly coupled in situ paradigm coordination is mini-

mal. 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 are 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.

Winner: tightly coupled in situ

2.6 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, super computing 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 have 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 additionally nodes are requested at the time the simulation is run, add to the cost of running a simulation. However, these additional nodes can be used asynchronously once the data are 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.

Winner: draw

2.7 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 that is done must be specified a priori, and so care must be taken to decide when the simulation is launched 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 the data.

Winner: draw

2.8 Scalability

Any in situ paradigm is constrained to use the concurrency of the allocated resource. 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 the performance.

Winner: loosely coupled in situ

2.9 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, or errors in visualization routines, and could result in faults or crashes. Because of the expense of super computing time, and the drastic impact of faults on simulation codes, fault tolerance is a requirement. Something that in practice is very hard 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.

Winner: loosely coupled in situ

2.10 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.

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.

However, given the advantages afforded by the separation of simulation and visualization, the loosely coupled paradigm occupies a much stronger position.

Winner: loosely coupled in situ

3. DISCUSSION

Based on the evaluation of the 10 factors we considered, there are clearly very good reasons for using a different tech-

niques. In cases with very specific needs, there is often a clear choice. In practice however, there are generally many factors under consideration, and we hold that some factors are much more important than others. In particular, we hold that fault tolerance, ease of use, and data translation are the most important of the 10 factors discussed.

As discussed in Section 2.9 the increasing complexity of supercomputers and the workflows being run on them makes fault tolerance of paramount importance. The ability of loosely coupled in situ to completely separate the simulation from the visualization makes it the clear choice.

On a related note, the complete separation of simulation and visualization in a loosely coupled paradigm is a large contributor to the win for ease of use concerns (see Section 2.10). This minimization of contact points between the two, along with the flexibility provided with configuration of simulation runs and setup of visualization choices make loosely coupled in situ the clear choice.

Finally, as discussed in Section 2.4, the diversity of data models and data layouts in simulation codes makes efficient interfacing of simulation outputs and visualization a daunting challenge. Loosely coupled in situ methods solve this problem by doing what simulations and visualization routines already do, writing and reading data. Simulations do not even have to be aware of what happens after data transfer calls are made, the underlying system takes care of transferring the data, and the visualization access the data by making data read calls.

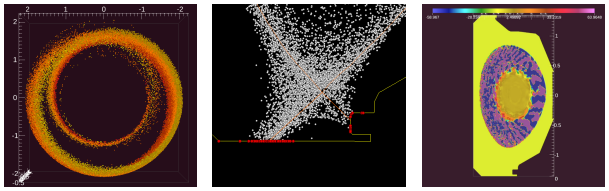
The advantages of loosely coupled in situ in these key areas makes it clear that this paradigm should be a staple in visualization now, and going forward. As a testament to the viability of this paradigm, loosely coupled techniques have been demonstrated with production runs on some of the largest super computers in the world [2, 17, 8].

Finally, there is one final and critical point for consideration. Hybrid methods [4], where both tightly and loosely coupled paradigms are used at the same time, are an exciting and very promising direction. These methods support the flexibility of processing data on the simulation resource 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 paradigms. However, hybrid methods are *only* possible within a context that supports loosely coupled in situ. It is otherwise impossible.

4. IN PRACTICE

The setup we employ places an emphasis on several of the factors discussed in Section 2, including ease of use, fault tolerance, data translation, scalability, and resource requirements. This maps most directly onto a loosely coupled in situ paradigm. Our workflow consists of three primary elements: (1) the simulation code; (2) a data transfer system to move data from the simulation to the visualization nodes; and (3) an efficient parallel visualization library. The simulation code, XGC1 [3], is a highly scalable physics code used study plasmas in fusion tokamak devices. For the latter two elements, we utilize three important libraries which are described below: ADIOS and DataSpaces for data management and transfer, and VTK-m as a framework for light weight visualization plugins.

The loosely coupled paradigm in ADIOS and DataSpaces provides for a clean interface and separation from XGC1 that provides ease of use, and fault tolerance. The ability to control the concurrency of the visualization tasks independent of the concurrency of XGC1 is important for ensuring good scalability on the visualization nodes. Further, the resource requirements can be specified based on the types of visualization that will be performed. The VTK-m framework offers a data model with the flexibility to efficiently, and optimally represent the output format for XGC1.



(a) Selection of particles of interest (b) Particle interaction with vessel boundary (c) Slice plane of field data

Figure 1: Representative examples of XGC1 particle and field visualization performed with a loosely coupled in situ paradigm.

4.1 ADIOS

The Adaptable I/O System (ADIOS) [10], is a componentization of the I/O layer that is accessible via a posix-style interface. The ADIOS API abstracts the operation away from implementation, allowing users to compose their applications independent of the underlying software and hardware. This capability, along with the functionality of DataSpaces [7] allows this same API to support read/write operations from/to the memory space of visualization nodes.

This type of loosely coupled in situ provides significant advantage for one of the most important factors considered, namely ease of use. It is worth emphasizing that loosely coupled in situ is achieved with minimal modifications to the simulation code. It uses something the simulation is already doing, namely I/O. These further address two of the most important factors, ease of use and fault tolerance.

4.2 Visualization Plugins

We designed our visualization routines as flexible, light weight plugins. Our plugins are based on an emerging community standard, VTK-m [18], which is a project building upon the success of three existing visualization frameworks, Dax [15], PISTON [11], and EAVL [13, 14]. The VTK-m framework is targeted to emerging computational systems where parallelism and the use of accelerators are dramatically increasing, and memory per core is decreasing. An emphasis has been placed on much more powerful data models that allow efficiencies in representing the various mesh types and data layouts used by simulation codes.

4.3 Visualization Workflows for XGC1

In previous work we utilized the features of ADIOS and EAVL (as a precursor to VTK-m), and demonstrated the effectiveness of loosely coupled in situ visualization for large scale simulation codes using a workflow consisting ADIOS, data staging and EAVL [17]. In that work we focused on the performance, scalability, and ease of use of visualization plugins that were used on the output of the XGC1 simulation code.

In that study we performed visualization on two different output fields from XGC1, the plasma particles (both ions and electrons), and field variables from the unstructured mesh. The ease of use of this system was highlighted with the fact that no changes to XGC1 were required. All modifications to data movement were accomplished with only a change to the ADIOS configuration file. At each simulation step, particles of interest were identified and visualized (Figures 1(a) and 1(b)) in parallel along with the visualization of a slice plane through the mesh, allowing us to monitor simulation field data, such as plasma turbulence (Figure 1(c)). These images were then used for monitoring the simulation and for post run analysis.

Using the factors from Section 2 to compare the two paradigms highlights the advantages of a loosely coupled paradigm. Using the the ADIOS API, no modifications are made to the simulation code to send data to the visualization nodes via DataSpaces.The only change required is to the ADIOS configuration file which is read when the simulation starts. This affords large advantages in both ease of use, and fault tolerance.Further, the data translation issues are avoided since the simulation code writes data in a known format to ADIOS, which flows to the visualization nodes, and is then read by the visualization plugin. This also highlights the ease of use advantage since the visualization is doing something that it already does, namely, read data. Further, the separation of simulation and visualization resources further highlights ease of use by eliminating any dependencies between the simulation and visualization code, as well as providing a layer of protection through fault tolerance.

5. FINAL THOUGHTS

The terms tightly and loosely coupled in situ are overladed in the community and are often used to mean disparately different things. This becomes especially true when the case for asynchronous data transfers on the simulation nodes are brought in, when GPUs and accelerators are considered, and when visualization codes begin to exploit deeper memory hierarchies. We presented our evaluation of the 10 factors based on one definition, but maintain however, that even in these more complicated scenarios, loosely coupled in situ visualization would remain the winner.

In conclusion, we presented 10 factors for comparing tightly and loosely coupled in situ paradigms. We have presented our perspective of each paradigm with respect to each of the factors and have found that the loosely coupled paradigm is significantly better suited for both the near term, and the foreseeable future. And finally, we note that the significant advantages to be gained using a hybrid paradigm can *only* be realized within a system that is based on a loosely coupled paradigm.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the use of the resources of the Oak Ridge Leadership Computing Facility (OLCF). We are especially grateful to Choong-Seock Chang and Seung-Hoe Ku of the Princeton Plasma Physics Laboratory for the use of XGC data. This work was done supported by the Scalable Data Analysis and Visualization Institute (SDAV) which is funded by the Advanced Scientific Computing Research Office, Office of Science, U.S. Department of Energy, under contract DE-SC0007446.

7. REFERENCES

[1] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, E. Bethel, H. Childs, et al. Scientific discovery at the exascale. *Report from the DOE ASCR 2011 Workshop on Exascale Data Management*, 2011.

[2] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. 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.

[3] C. Chang, S. Ku, P. Diamond, Z. Lin, S. Parker, T. Hahm, and N. Samatova. Compressed ion temperature gradient turbulence in diverted tokamak edgea). *Physics of Plasmas (1994-present)*, 16(5):056108, 2009.

[4] H. Childs, M. Kwan-Liu, Y. Hongfeng, W. Brad, M. Jeremy, F. Jean, K. Scott, P. Norbert, S. Karsten, W. Matthew, P. Manish, and Z. Fan. In situ processing. In E. W. Bethel, H. Childs, and C. Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, Boca Raton, FL, 2012.

[5] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. H. Weber, and E. W. Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Comput. Graph. Appl.*, 30(3):22–31, May 2010.

[6] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. H. Weber, and E. W. Bethel. Visualization at extreme scale concurrency. In E. W. Bethel, H. Childs, and C. Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, Boca Raton, FL, 2012.

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

[8] C. Docan, F. Zhang, T. Jin, H. Bui, Q. Sun, J. Cummings, N. Podhorszki, S. Klasky, and M. Parashar. Activespaces: Exploring dynamic code deployment for extreme scale data processing. *Concurrency and Computation: Practice and Experience*, pages 1–22, 2014.

[9] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. 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.

[10] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. 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.

[11] L.-t. Lo, C. Sewell, and J. P. Ahrens. Piston: A portable cross-platform framework for data-parallel visualization operators. In *EGPGV*, pages 11–20, 2012.

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

[13] J. S. Meredith, S. Ahern, D. Pugmire, and R. Sisneros. EAVL: the extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 21–30. The Eurographics Association, 2012.

[14] J. S. Meredith, R. Sisneros, D. Pugmire, and S. 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*, GPGPU-5, pages 11–19, New York, NY, USA, 2012. ACM.

[15] K. Moreland, U. Ayachit, B. Geveci, and K.-L. 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.

[16] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, et al. Examples of in transit visualization. In *Proceedings of the 2nd international workshop on Petascale data analytics: challenges and opportunities*, pages 1–6. ACM, 2011.

[17] D. Pugmire, J. Kress, J. Meredith, N. Podhorszki, J. Choi, and S. Klasky. Towards scalable visualization plugins for data staging workflows. In *Big Data Analytics: Challenges and Opportunities (BDAC-14) Workshop at Supercomputing Conference*, November 2014.

[18] C. Sewell, J. Meredith, K. Moreland, T. Peterka, D. DeMarle, L.-t. Lo, J. Ahrens, R. Maynard, and B. 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.

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

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