In Situ Analysis and Visualization of Fusion Simulations: Lessons Learned

Mark Kim[1(✉)], James Kress[1,5], Jong Choi[1], Norbert Podhorszki[1], Scott Klasky[1], Matthew Wolf[1], Kshitij Mehta[1], Kevin Huck[5], Berk Geveci[3], Sujin Phillip[3], Robert Maynard[3], Hanqi Guo[2], Tom Peterka[2], Kenneth Moreland[4], Choong-Seock Chang[6], Julien Dominski[6], Michael Churchill[6], and David Pugmire[1]

[1] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
kimmb@ornl.gov
[2] Argonne National Laboratory, Lemont, IL 60439, USA
[3] Kitware Inc., Clifton Park, NY 21065, USA
[4] Sandia National Laboratories, Albuquerque, NM 87185, USA
[5] University of Oregon, Eugene, OR 97403, USA
[6] Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA

Abstract. The trends in high performance computing, where far more data can be computed that can ever be stored, have made in situ techniques an important area of research and development. Simulation campaigns, where domain scientists work with computer scientists to run a simulation and perform in situ analysis and visualization are important, and complex undertakings. In this paper we report our experiences performing in situ analysis and visualization on two campaigns. The two campaigns were related, but had important differences in terms of the codes that were used, the types of analysis and visualization required, and the visualization tools used. Further, we report the lessons learned from each campaign.

Keywords: In situ · Scientific · Visualization

1 Introduction

The traditional *post hoc* model for analysis and visualization has been for the simulation to write data to disk, and for a set of tools to read the data back into memory at a later time to perform analysis and visualization. Because supercomputers can produce far more data than can be saved, this traditional model is breaking down. This imbalance in supercomputers has made it very attractive to perform the analysis and visualization operations *in situ*, as soon as the data are produced and resident in memory.

Because the traditional model is breaking down, new models, frameworks, and tools need to be developed that allow simulations to make use of *in situ* visualization. These same models, frameworks and tools, where possible, need to

R. Yokota et al. (Eds.): ISC 2018 Workshops, LNCS 11203, pp. 230–242, 2018.
https://doi.org/10.1007/978-3-030-02465-9_16

be able to work with existing analysis and visualization tools, or provide a foundation for the development of the next generation of libraries and tools. While *in situ* visualization is a straightforward concept, the implementation details, constraints, and requirements for resilience make this much harder to achieve for production use. Much of the difficulty is a result of the visualization being much closer to the running simulation. In a *post hoc* model, where communication between the simulation and visualization is done through files, there are far fewer constraints on time and resource utilization, as well as resilience. In an *in situ* model, strict constraints are placed on the analysis and visualization routines. Further, any error in the analysis or visualization that causes the simulation to crash are unacceptable. Requirements like these, and many others, make robust deployment of in situ visualization a challenge.

In this paper we describe our experience performing analysis and visualization with two different simulation campaigns. The two campaigns were performed with a plasma physics simulation that each had different analysis and visualization requirements. In addition, each campaign used performance tools to monitor the simulation while it was running and report different statistics. Finally, data compression techniques were used on the simulation outputs. The performance of these compression techniques was also monitored and reported. Each campaign used a different set of analysis and visualization tools. We discuss the workflows that were constructed, the resulting visualizations, and the lessons learned from each. Also, each the visualization of each campaign was focused on different aspects of the simulation. We discuss these requirements, and the lessons learned.

We begin with a brief review of related work in Sect. 2. In Sect. 3 we provide some motivation and background on the structure of each campaign. In Sect. 4 we describe how the simulation, analysis and visualization components of each campaign were configured. In Sect. 5 we show the analysis and visualization results that were produced for each campaign. Finally, in Sects. 6 and 7 we discuss the lessons learned throughout both simulation campaigns, and make some concluding remarks, and thoughts for future work.

2 Related Work

Current and future trends in high performance computing continue to point to systems where the ability to produce data far exceeds the ability to save that data. This system imbalance has made in situ methods an important topic for simulation science. Bauer et al. [6] provides a detailed survey of the broad and long history of in situ visualization.

A number of frameworks for in situ visualization have been developed and used over the years. These include SCIRun [19], a problem solving environment that supports in situ visualization and computational steering. LibSim [23] is a library that allows simulations to connect to the VisIt [9] visualization tool. Catalyst [5] allows a simulation to connect to ParaView [1] for visualization.

ADIOS [17] is an I/O middleware library that exposes in situ processing to a simulation through a POSIX-like API. ADIOS supports memory-to-memory

coupling through a number of different transports methods, including DataSpaces [12], DIMES [24], and FlexPath [10].

Several studies have been done to explore using these frameworks with simulations. Bennett et al. [7] do in situ visualization and analysis using the ADIOS. Ayachit et al. [4] use the SENSEI framework to study in situ visualization using ADIOS, LibSim and Catalyst.

3 Motivation

The two simulation campaigns in this paper were motivated by trying to understand new approaches for simulation science on HPC systems. These efforts were focused in several directions. First, as mentioned previously, the growing compute and I/O imbalance, and a need to monitor running simulations is necessitating the use of in situ methods for analysis and visualization. Second, there is a need to monitor and understand the different performance aspects of the simulation, including FLOPS, memory usage, network bandwidth, compression ratios, and I/O throughput. Third, the physics teams are focusing on using code coupling to solve the multi-scale, multi-physics aspects of the simulations. Code coupling is a powerful technique for using multiple simulation codes to work together on difference pieces of the problem, and then coordinate their results for whole-physics solution. In this context, the simulation team was interested in analysis and visualization, performance, and compression for the individual codes, as well as the coupled solutions.

The code coupling was performed in an environment enabled by in-memory communication. The analysis and visualization also made use of this in-memory environment for access to simulation data. This environment enabled the launching of executables to perform the various operations (simulation, performance monitoring, analysis and visualization). A workflow system was used that launched the various executables. These executables include the two simulations, the performance monitoring executables, and the analysis and visualization executables. The workflow also specified the connectivity between these executables. This environment allows flexibility in what operations are executed, where they are executed, and what types of pipelines are constructed.

4 Setup

This section describes the components, and their configurations for each of the two simulation campaigns. In each of the simulation campaigns, two physics codes were used to simulate plasma physics in a tokamak device. One code was responsible for the core region of the tokamak, while the other code was responsible for the edge region of the tokamak. The particular codes used in each campaign are discussed in Sects. 4.1 and 4.2.

Each campaign was initiated using the Savanna [15] workflow system. Savanna is a python-based workflow composition and dispatch engine that to

composes and orchestrates complex workflows that consist of multiple applications. Savanna specifies the parameters for each component in the workflow, as well as how the components are configured and connected to one another. These components all make use of the ADIOS middleware layer for memory-to-memory data transfer, and I/O operations. Two different memory-to-memory data transport methods were used. FlexPath [10], a peer-to-peer data staging service, and DIMES [24], a client-server based model data transfer service. These transport methods in ADIOS enabled the coupling of the physics codes, as well as the transport of simulation output to other data consumers in the workflow. The TAU [20] performance tool was used to monitor the running simulations, and used ADIOS for memory-to-memory data transfer to visualization resources. The in situ analysis and visualization operations use ADIOS to obtain the data from both simulations, as well as the performance tools. The data compression capability in ADIOS was also used to reduce the amount of data transfered in situ, as well as written to disk. Two different lossy compression methods were used in the two campaigns; MGARD [2,3], and SZ [11]. To assess the quality of the compression, Z-checker [21] was also used.

For analysis and visualization, two different tools were used: VisIt [9], and VTK-m [18]. VisIt is a fully featured parallel production tool for scientific analysis and visualization. VTK-m is a scientific visualization toolkit that is enabled to take advantage of the multi/many-core nodes in HPC systems. Small, lightweight analysis and visualization services were built using VTK-m to perform specific tasks. These services used the ADIOS in situ environment, and the ADIOS visualization schema [22] to ingest simulation data, and to output data extracts for subsequent processing. The visualization schema is a convenience layer in ADIOS that allows an application to add additional markup to data as it is being written. This additional markup then allows the reader application to know exactly how to ingest the data. For example, it could specify that it was writing a rectilinear mesh, and that mesh had two fields, one called "energy" which is cell centered, and one called "velocity" which is point centered (Fig. 1).

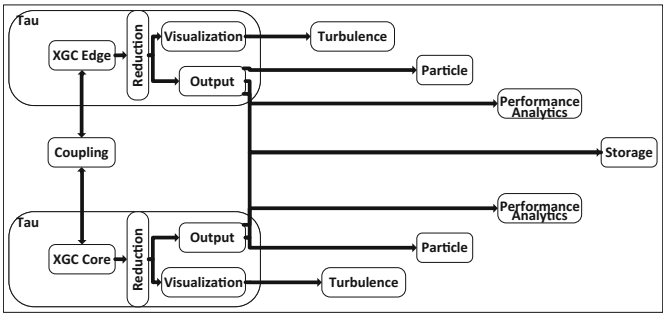


Fig. 1. Simulation 1 code coupled fusion simulation diagram.

4.1 Campaign 1

In the first campaign, two instances of the same code were coupled together. This choice permitted to focus on the code coupling model when avoiding the extra-complications related to the use of different codes. Two instances of a single code, XGC1 [8] were used, each configured for a specific region of the computational domain, see reference [13] for a description of the coupling scheme. TAU was used performance monitoring, and both VisIt and VTK-m were used for analysis and visualization.

Computation Code. The XGC1 computation code is a highly scalable physics code used to study plasmas in fusion tokamak devices. XGC1 is a particle-in-cell code. The simulation will solve for the motion of very large number of charged particles inside the plasma, and then statistically deposit the particles onto the mesh. Scientists are interested in both the field variables on the mesh, and the particles.

In this example, two instances of XGC1 were run. One instance was configured for the core region of the plasma, and another instance was configured for the edge region of the plasma. There was a coupling region of the mesh where the core and edge codes would communicate the state of the physics. The ADIOS memory-to-memory data transport is used for the simulations to communicate this coupling data.

Visualization. In this demo, VisIt was used as the dashboard to display the progress of the running simulation through different visualizations. First, the simulation and the visualization services were launched. Subsequently, the VisIt GUI interface would be launched and it would connect to the running simulation, and display the simulation progress in real time.

VisIt was configured using a Python script which setup the different windows and positions of each of the visualizations within the dashboard, as well as handling the updating of plots as new data became available. Due to time constraints, we were unable to run VisIt using the ADIOS memory-to-memory data transport, and we used ADIOS files output by the analysis services, and the simulation. Each analysis service would update a ".visit" file as soon as it had new data ready to be visualized. The Python script would then check for a ".visit" file to be updated, indicating that new data was ready to be visualized, and as soon as each analysis application had new data available, VisIt was able to perform an update (Fig. 2).

4.2 Campaign 2

In the second campaign, two different codes were used coupling. GENE [16] was used for the core region, and XGC1 was used for the edge region. The same coupling model that was used for coupling two instance of XGC and described

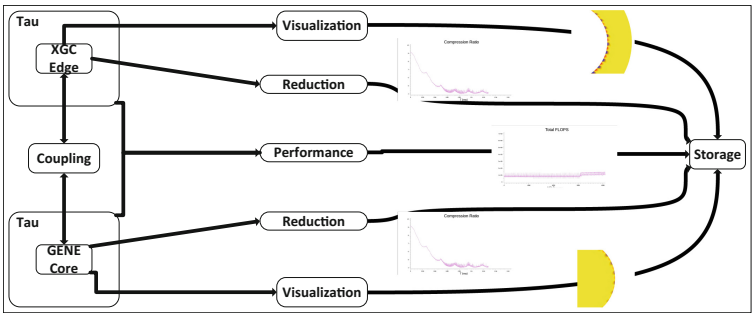


Fig. 2. Simulation 2 code coupled fusion simulation diagram.

in Ref. [13] has been used. Additional difficulties, including the extra complications of having different grids in GENE and XGC, will be described in an upcoming publication [14]. As in campaign 1, TAU was used performance monitoring. MGARD, and SZ were used for compression of simulation data, and Z-checker used to monitor the compression quality. Lightweight in situ analysis and visualization services were used for all of the visualizations.

Computation Code. GENE is an Eulerian gyrokinetic code that is used to simulate the core region of the tokamak. The same code in Campaign 1, XGC1 is used to simulate the edge region of the tokamak. As before, a coupling region is used between the core and edge for both codes to communicate the state of the physics. The ADIOS memory-to-memory data transport is used for the simulations to communicate this coupling data.

Visualization. The analysis and visualizations for Campaign 2 were generated using a collection of lightweight services. Each service used the ADIOS environment, ADIOS visualization schema, and memory-to-memory data transport for the communication of data. At each timestep of the simulation, data would flow through the ADIOS environment to each services where analysis and visualizations were performed. The terminal rendering services would produce output images that are saved to disk.

5 Results

5.1 Campaign 1

The code coupled fusion simulation was run on OLCF Titan, with XGC1 utilizing 2048 MPI processors and GENE using: 2048 MPI processors. Each visualization and analysis process is assigned an MPI processor as well.

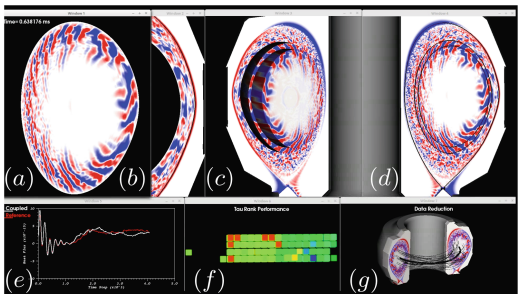


Fig. 3. Dashboard for Campaign 1.

Visualization. The fusion scientists were interested in visualizations of several physics quantities, and performance metrics. First, they were interested in the energy concentrations that form within the plasma. This quantity, called the *scalar potential*, is derived from the particles, and is a field variable on the core and edge meshes (Fig. 3(a–b), respectively). Second, they are interested in the behavior of the particle data in the simulation. This was visualized using a reduction technique where the bulk particle velocity field was computed from the motion of the particles. The bulk velocity field was then visualized using streamlines. Third, in order to understand the convergence, they wanted to see a time varying 1D plot of the heat flux. Finally, to monitor the system performance, a visualization of the total FLOPS and memory usage for each node was done.

Streamline Visualization. Particles from XGC1 are the largest data produced by the simulation. There are too many particles to save to disk every timestep, so they are only saved during simulation checkpointing. This means that performing any analysis or visualization on the particles is essentially impossible, since once a new checkpoint is saved, the old one is deleted to clear space. To overcome this limitation, we processed the particles coming from the simulation every simulation cycle in situ. This processing consisted of a spatial reduction of the particles by binning. All particles from the simulation were binned into a decimated version of the simulation mesh. Due to the large number of particles, 3 billion++, this operation was performed in parallel across several nodes. This process allowed every particle from the simulation to contribute to the final binned values. Following binning the binned representation of the particles was saved to disk for visualization and analysis. Using this approach we gained a high temporal frequency view of the particles as they progressed during the simulation. After the data was saved to disk it was loaded into VisIt were a streamline operation was performed, with the final rendered image (Fig. 3(f)) showing up on the simulation dashboard.

5.2 Campaign 2

The code coupled fusion simulation was run on OLCF Titan, with XGC1 utilizing 3072 MPI processors and GENE using: 1024 MPI processors. Each visualization and analysis process is assigned an MPI processor as well.

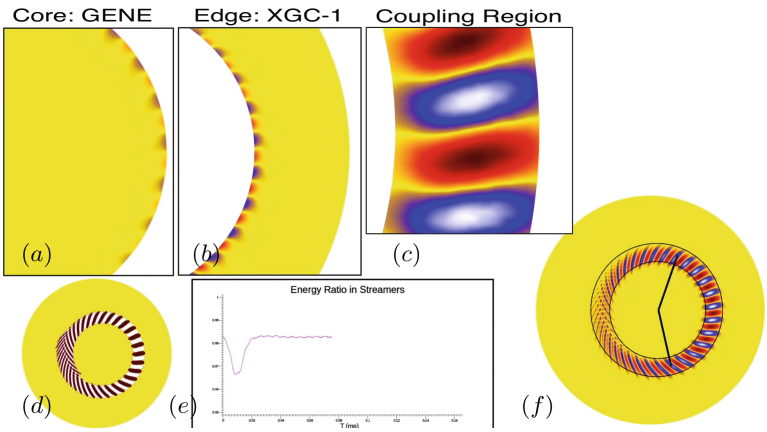
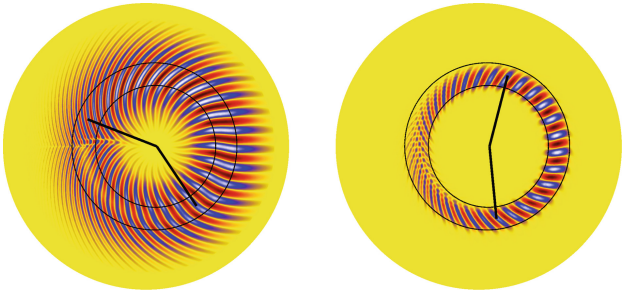


Fig. 4. The dashboard visualizations for Campaign 2.

Visualization. Similar to the visualization performed for Simulation 1 in Sect. 5.1, scientists needed to visualize the potential energy of the simulation. The potential energy is visualized for the core, edge, and coupling (Fig. 4(a–c)) of the fusion simulation.

Further, the scientists were looking for a specific energy interaction between the streamers in the coupling region of the simulation. The two circular black lines denote the coupling region of the simulation where XGC1 and Gene interact. The two lines emanating from the center of the tokomak plane, resembling clock hands, change dynamically as the simulation progresses, and denote the region where the highest 70% of streamer energy lies. In the beginning of the simulation this region is large, and as the simulation progresses the region shrinks (Fig. 5), and the area denoted by the hands is essentially the strong side of the plasma. This visualization helps researchers to confirm that the simulation is progressing normally and that things have converged.

Analytics. The analytic services in this demo were broken into four light-weight visualization services. First was the SZ visualization service. SZ was used to perform compression on data coming from XGC. The visualization service connected to staging and read ten individual scalar values each cycle from the simulation. These scalars values were then plotted as line graphs using the VTK-m software rasterizer, and an image was stored to disk at each time step. Second, MGARD



(a) Cycle 1 (b) Cycle 1980

Fig. 5. (a) and (b) denote the regions of highest energies at step 1 and 1980, respectively.

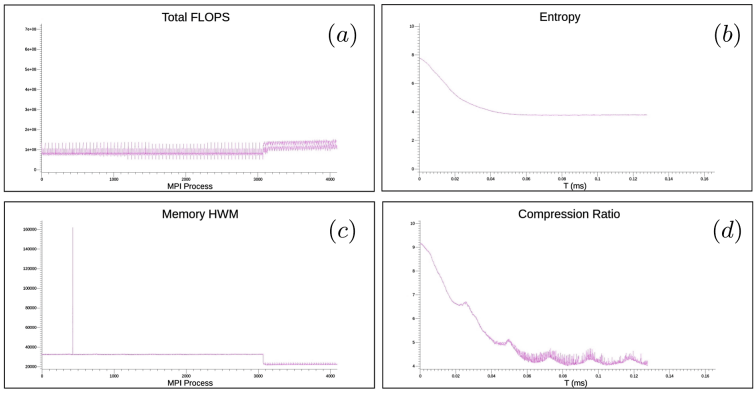


Fig. 6. The dashboard graphs for Campaign 2.

was used to perform compression on data coming from GENE. The visualization service connected to staging and read five individual scalar values each cycle from the simulation. These scalar values were then plotted as line graphs using the VTK-m software rasterizer, and an image was stored to disk at each time step. Third was the TAU visualization service. Finally, Z-checker provided further analytics for the compression, the ratio and the entropy (Fig. 6(b) and (d), respectively). TAU was used to gather simulation performance data from XGC1 and GENE. The visualization service connected to staging and read eight scalar arrays from staging each simulation cycle. The length of these arrays were 4096, and gave performance data for each simulation process in XGC1 and GENE. The data was then plotted as a line graph (Fig. 6(a)), and an image was stored to disk for each time step.

The TAU visualization service allowed us to see when failures were occurring during our testing runs. Specifically, the ability to monitor the memory usage per process in the simulation aided in finding a bug in a different visualization service.

During a test run, there were several nodes that consumed all the memory as the simulation ran, until it eventually crashed the simulation itself. This can be seen in the spike in the graph in Fig. 6(c). As it turns out, this memory increase was due to a leak in the Z-checker compression service in ADIOS. The memory usage visualization was useful in this case because we could see that the simulation crash was only due to a small number of nodes running out of memory, which narrowed down where the issue could be coming from.

The fourth and final analysis service was a feature detection service for extracting streamers in the plasma. The streamers, shown in Fig. 4(d) are areas of high energy density, and are a key driver in the evolution of the plasma.

6 Discussion

In this section we discuss the lessons learned from the two campaigns. The backbone of both campaigns was the ADIOS environment. It provide the memory-to-memory data transport, as well as the I/O for all the components in both campaigns. Using a single environment made it much easier to setup the workflows, and provide the connections between individual components. This was especially important in Campaign 2 where the analysis and visualization requirements were changing, and the plug-and-play nature of the system made it easy to develop a service that would easily work.

A significant advantage of the ADIOS environment is that it is easily configurable to use either files, or memory-to-memory data transfer. Because of this, services could be developed and tested using files, and then deployed in situ by setting a flag. This was invaluable in the development, testing, and debugging of services. This also enabled the development of services to be performed independent of the simulation, and the supercomputer. It also made it easier to groups of developers to build services independently, and the coordinate at the end. Once the services run independently in a development environment using files, a parameter is modified, and the services will run on the HPC system.

The ADIOS environment also made it trivial to chain multiple services together for more complex operations. Because we were using the ADIOS visualization schema to annotate the semantics of the data being transported, the services could easily communicate to each other. The ability also made it trivial to chain multiple services together, as was done with the streamer detection algorithm in Campaign 2.

Another advantage of the ADIOS environment is the runtime resilience provided. ADIOS is designed to allow the dynamic connection and disconnection of services from the ADIOS stream. That is, the simulation that is writing data via ADIOS will not crash if the process consuming that data crashes. A process can then reconnect to the ADIOS data stream after it has recovered from the failure. It is important to note that the analysis process itself is responsible for being resilient enough to be restarted and reconnected to the ADIOS stream, ADIOS will not maintain any state for a crashed analysis process.

An example of the resilience in ADIOS comes from Campaign 2. In Campaign 2, there was a memory leak in the base layer used by VTK-m services. Each

time a service read in a timestep, tens of megabytes of memory would leak. After enough timesteps, the node would run out of memory and crash. However, since the simulation and visualization are not directly connected, the simulation continued to run.

Because of time constraints, and technicalities associated with Campaign 1, we were not able to connect VisIt to the ADIOS environment using the memory-to-memory data transfer. As a workaround, VisIt read ADIOS files from disk. There were three different files ready by VisIt that were generated by the simulation, and analysis services: simulation output, TAU performance data, vector field data produced by the in situ particle reduction services. The ability to avoid disk altogether would have provided flexibility for the visualization, and made it more efficient.

For visualization, each campaign used different modes. In Campaign 1, VisIt, a fully featured visualization tool was used. In Campaign 2, a set of lightweight visualization services were used. At a high level, VisIt is designed for interactive visualization while a lightweight visualization service is not. This can affect how scientists interface with a simulation as it runs. With the service oriented strategy, the output is either a data extract, or an image. This means that scientists are limited with exploring the dataset beyond the visualizations they have invoked through the framework. Understanding these limitations, and the requirements by the scientists can inform which choice is most appropriate.

Further, VisIt is a mature, full featured tool, while the lightweight visualization service is based on VTK-m, which is a relatively young, and growing library. Because of this, some more advanced visualizations are possible using VisIt, as shown in Fig. 3(c), (d). However, the flexibility provided by lightweight service model makes it much easier to provide custom visualizations, like the ballooning angle visualization in Fig. 5.

Most of the visualizations done in each of the two campaigns were not performance intensive. They ran one a single node, and were easily able to keep up with the output frequency of the simulations. The particle visualization was the most time and resource intensive analysis that was run. It was launched with multiple nodes and multiple processes per node in order to keep up with the simulation output. However, there were times when it did not maintain pace with the simulation. When the particle visualization was unable to keep up, some simulation steps were dropped, but it would continue by reading the next available one once it had caught up. Again, the simulation continued regardless of the whether the particle visualization kept up. This demonstrated the important resilience of ADIOS and the visualization services.

Finally, one problem encountered was rendering on the HPC resource using lightweight services. This is because of the challenges having system administrators install and maintain OpenGL and EGL drivers on the compute nodes. To get around this problem, we used a software based rasterizer in VTK-m to visualize the graphs, and a software based ray casting algorithm to render the visualizations.

7 Conclusion and Future Works

The growing complexity in high performance computing has led to new frameworks, models, and tools for simulation codes. These tools need to be able to work with existing visualization tools, or provide a way forward to enable new models and frameworks to be implemented. One attractive approach to allow this is to perform visualization *in situ*, where the data resides in memory.

In this paper, we discussed various aspects of performing *in situ* visualization for two plasma physics simulation campaigns. Each campaign had varying visualization and analysis requirements and used different visualization and analysis algorithms. We discussed two different workflows, one that is interactive and another that is service oriented. Finally, we discussed the resulting visualizations and the lessons learned.

In the future, we would like to further explore this interactive versus service oriented paradigms. In particular, when one strategy is beneficial over the other, and vice versa. Finally, we would like to explore a mix of interactive and service oriented visualization, as well. Ultimately, we believe that a combination of both interactive, and a service oriented visualization provides the most flexibility to scientists.

References

1. Ahrens, J., Geveci, B., Law, C.: Visualization in the paraview framework. In: Hansen, C., Johnson, C. (eds.) The Visualization Handbook, pp. 162–170 (2005)
2. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: MGARD: a multilevel technique for compression of floating-point data. In: DRBSD-2 Workshop at Supercomputing 2017, Colorado, USA (2017)
3. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: Multilevel techniques for compression and reduction of scientific data-the univariate case. Comput. Vis. Sci. (2017, submitted)
4. Ayachit, U., et al.: The SENSEI generic in situ interface. In: 2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), pp. 40–44, November 2016. https://doi.org/10.1109/ISAV.2016.013
5. Ayachit, U., et al.: 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, pp. 25–29. ACM (2015)
6. Bauer, A.C., et al.: In situ methods, infrastructures, and applications on high performance computing platforms, a state-of-the-art (STAR) report. In: Computer Graphics Forum, Proceedings of EuroVis 2016, vol. 35, no. 3, June 2016. LBNL-1005709
7. Bennett, J.C., et al.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC 2012, pp. 49:1–49:9. IEEE Computer Society Press, Los Alamitos (2012). http://dl.acm.org/citation.cfm?id=2388996.2389063
8. Chang, C., et al.: Compressed ion temperature gradient turbulence in diverted tokamak edgea. Phys. Plasmas (1994-present) **16**(5), 056108 (2009)

9. Childs, H., et al.: VisIt: an end-user tool for visualizing and analyzing very large data. In: High Performance Visualization-Enabling Extreme-Scale Scientific Insight, pp. 357–372, October 2012
10. Dayal, J., et al.: Flexpath: type-based publish/subscribe system for large-scale science analytics. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 246–255. IEEE (2014)
11. Di, S., Cappello, F.: Fast error-bounded lossy HPC data compression with SZ. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, 23–27 May 2016, pp. 730–739 (2016)
12. Docan, C., Parashar, M., Klasky, S.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. Cluster Comput. **15**(2), 163–181 (2012)
13. Dominski, J., et al.: A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes. Phys. Plasmas **25**(7), 072308 (2018). https://doi.org/10.1063/1.5044707
14. Dominski, J., Merlo, G., et al.: Gyrokinetic core-edge coupling of the continuum code GENE with the particle-in-cell code XGC (temporary title). (in preparation)
15. Foster, I., et al.: Computing just what you need: online data analysis and reduction at extreme scales. In: Rivera, F.F., Pena, T.F., Cabaleiro, J.C. (eds.) Euro-Par 2017. LNCS, vol. 10417, pp. 3–19. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64203-1_1
16. Görler, T., et al.: The global version of the gyrokinetic turbulence code gene. J. Comput. Phys. **230**(18), 7053–7071 (2011). https://doi.org/10.1016/j.jcp.2011.05.034. http://www.sciencedirect.com/science/article/pii/S0021999111003457
17. Liu, Q., et al.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. Concurrency Comput.: Pract. Exp. **26**(7), 1453–1473 (2014). https://doi.org/10.1002/cpe.3125
18. Moreland, K., et al.: VTK-m: accelerating the visualization toolkit for massively threaded architectures. IEEE Comput. Graph. Appl. **36**(3), 48–58 (2016)
19. Parker, S.G., Johnson, C.R.: SCIRun: a scientific programming environment for computational steering. In: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing, p. 52. ACM (1995)
20. Shende, S.S., Malony, A.D.: The tau parallel performance system. Int. J. High Perform. Comput. Appl. **20**(2), 287–311 (2006)
21. Tao, D., Di, S., Guo, H., Chen, Z., Cappello, F.: Z-checker: a framework for assessing lossy compression of scientific data. Int. J. High Perform. Comput. Appl. 1094342017737147 (2017). https://doi.org/10.1177/1094342017737147
22. Tchoua, R., et al.: ADIOS visualization schema: a first step towards improving interdisciplinary collaboration in high performance computing. In: 2013 IEEE 9th International Conference on e-Science, pp. 27–34, October 2013. https://doi.org/10.1109/eScience.2013.24
23. Whitlock, B., Favre, J., Meredith, J.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, pp. 101–109 (2011)
24. Zhang, F., et al.: In-memory staging and data-centric task placement for coupled scientific simulation workflows. Concurrency Comput.: Pract. Exp. **29**(12), e4147 (2017)