LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-737832

The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman

M. Larsen, J. Ahrens , U. Ayachit, E. Brugger, H. Childs, B. Geveci, C. Harrison

August 31, 2017

In Situ Analysis and Visualization (ISAV) 2017
Denver, CO, United States
November 12, 2017 through November 12, 2017

Disclaimer

The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman

Matthew Larsen[1], James Ahrens[2], Utkarsh Ayachit[3], Eric Brugger[1]
Hank Childs[4], Berk Geveci[3], Cyrus Harrison[1]
Lawrence Livermore National Lab[1], Los Alamos National Lab[2]
Kitware, Inc[3], University of Oregon[4]
utkarsh.ayachit,berk.geveci@kitware.com,hank@cs.uoregon.edu
brugger1,cyrush,larsen30@llnl.gov,ahrens@lanl.gov

ABSTRACT

This paper introduces ALPINE, a flyweight in situ infrastructure. The infrastructure is designed for leading-edge supercomputers, and has support for both distributed-memory and shared-memory parallelism. It can take advantage of computing power on both conventional CPU architectures and on many-core architectures such as NVIDIA GPUs or the Intel Xeon Phi. Further, it has a flexible design that supports for integration of new visualization and analysis routines and libraries. The paper describes ALPINE's interface choices and architecture, and also reports on initial experiments performed using the infrastructure.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel and high-performance simulations**; **Scientific visualization**; **Ray tracing**; **Massively parallel algorithms**; **Shared memory algorithms**;

KEYWORDS

HPC, Scientific Visualization, In Situ

1 INTRODUCTION

ALPINE is a multi-institution effort funded by the U.S. Department of Energy's Exascale Computing Project (ECP) [15]. Its purpose is to deliver in situ infrastructure for visualization and analysis to ECP application teams. ALPINE's strategy is to support two existing in situ runtimes: VisIt's [7] LibSim [20] and ParaView's [2] Catalyst [5]. However, ALPINE has also developed a new in situ runtime, which is called "ALPINE Ascent".

We believe ALPINE advances the state of the art in three distinct ways:

- **Support for modern supercomputing architectures.** ALPINE was designed with modern supercomputing architectures in mind. It follows a hybrid parallel strategy, meaning it has support for both distributed-memory parallelism across nodes and shared-memory parallelism within a node. The shared-memory parallel support comes through usage of VTK-m [16], which encourages algorithm development using hardware-agnostic building blocks. These building blocks are replaced at compile time with efficient hardware-specific implementations, enabling portable performance over multiple architectures. ALPINE's distributed-memory parallel support can come from either DIY [17] or MPI [8]. ALPINE achieves this hybrid parallelism through use of a new library, called VTK-h ('h' for hybrid parallelism), that combines VTK-m and DIY/MPI. VTK-h is introduced later in this paper.
- **Flyweight infrastructure.** For ALPINE, the flyweight goal is realized in three ways: (1) an interface that can easily be adopted by stakeholders, (2) minimal dependencies on other software packages and small encumbrance on the binary size of the simulation code, and (3) minimal overheads incurred during processing, specifically with respect to copying data and memory usage.
- **Interoperability with software.** Although VTK-m plays a special role in the ALPINE project, the new Ascent runtime was designed to support additional libraries. Specifically Ascent makes use of a data flow library called "Flow" to organize execution. Flow is agnostic to the data models and libraries used in filters, and therefore can enable other libraries (such as R [19]) to be used within Ascent. Of course, it would be up to those libraries to provide support for parallelism, and additional work would be needed to bridge between data models (for example, VTK-m to R or vice-versa).

The paper is organized as follows: Section 3 describes ALPINE's interface concepts, Section 4 describes the main components of ALPINE's infrastructure, and Section 5 describes some initial results.

2 RELATED WORK

In situ processing has become increasingly popular in recent years. Bauer et al. surveyed key research and infrastructures and we direct interested readers to their survey [6]. Many

ISAV'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5139-3/17/11…$15.00
https://doi.org/10.1145/3144769.3144778

of the design decisions made by ALPINE are similar to LibSim [20] and Catalyst [5]: linking into the simulation binary, alternating execution between simulation and visualization, and sharing the same memory space. This contrasts with an approach like ADIOS [13], which takes a more loosely-coupled approach.

SENSEI [4] is a generic in situ interface that can be characterized as a "write once, use everywhere" approach that supports Catalyst, Libsim, and ADIOS back-ends. The ALPINE in situ infrastructure differs in that we have an execution model and our own visualization routines. Additionally, our execution model allows us to perform hybrid in situ, where we can perform on-node visualization then send the results off-node (e.g., ADIOS).

The ALPINE in situ infrastructure is heavily based on Strawman [12], with Strawman effectively serving as an early prototype for ALPINE. The key differences between ALPINE and Strawman are (1) ALPINE is intended for end users while Strawman was a mini-app, (2) ALPINE has extended distributed-memory support, including usage of DIY [17], (3) ALPINE uses VTK-m [16] where Strawman used EAVL [14], and (4) ALPINE has extended support for using other packages and libraries.

3 INTERFACE

The ALPINE Interface is the API used to direct ALPINE Ascent, ALPINE's new flyweight in situ runtime.

The top-level interface mirrors the Strawman interface and consists of four methods: "Open", "Publish", "Execute", and "Close". All data and parameters are passed into ALPINE using Conduit [10], a library that provides an intuitive model for describing hierarchical scientific data. Since Conduit provides C++, C, Fortran, and Python APIs for data description, exposing the ALPINE interface in these languages is trivial.

ALPINE is initialized by calling the "Open" method and passing several parameters, including an MPI communicator. Next, simulation mesh data structures are described using Conduit's Mesh Blueprint [11], a set of conventions for describing domain-decomposed mesh data ranging from uniform meshes to topologies composed of higher order elements, and the resulting Conduit Node is published through ALPINE's "Publish" method. ALPINE verifies that published data conforms to the Mesh Blueprint. If the data does not conform, a detailed description of what is missing or invalid is provided. Third, ALPINE executes sets of actions described and passed to the "Execute" method. Finally, the "Close" call informs ALPINE that no further execution is required.

```
Alpine alpine;
alpine.Open(options);
alpine.Publish(data);
alpine.Execute(actions);
alpine.Close();
```

Listing 1: Top-level ALPINE interface in C++.

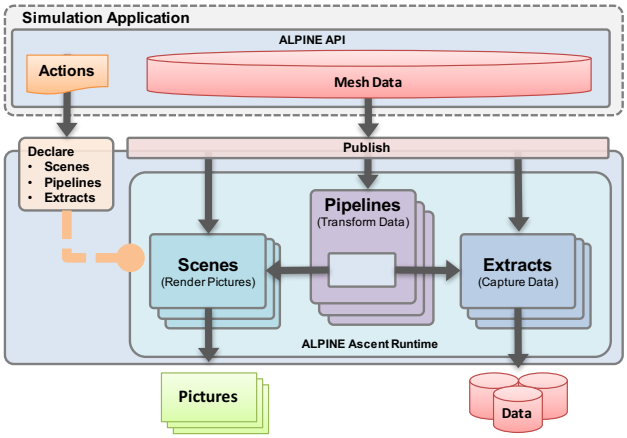

Figure 1: The ALPINE Interface allows users to declare Scenes, Extracts, and Pipelines for execution by the ALPINE Ascent runtime.

3.1 Actions

In ALPINE, we replaced Strawman's simple rendering focused actions with a more expressive interface. The new interface is organized around three main use cases: making pictures, capturing data, and transforming data. Users invoke these use cases by passing actions that declare sets of Scenes, Extracts, and Pipelines:

- Scenes: make pictures
- Extracts: capture data
- Pipelines: transform data

At a high level, Scenes allow users to describe the images they want to create. Extracts allows users to describe how they want to capture data for use outside of ALPINE, and Pipelines allow users to describe the ways they want ALPINE to transform their data. Figure 1 outlines how three concepts are supported.

3.1.1 Scenes: Make Pictures. A scene encapsulates the information required to generate one or more images. The user specifies a collection of plots (e.g., volume or surface rendering) and associated parameters such as camera definitions, light positions, and annotations.

To define a scene, a user describes one or more plots. The simplest plot description requires only a plot type and scalar field name. A scene defined in this way uses the default data source, which is all of the data published by the simulation, and default settings for camera position, image resolution, lighting, and annotations. An example of a minimally defined scene is shown at the beginning of Listing 2. The definition of "scene1" indicates the user wants a volume plot of the scalar field "noise" using the default parameters. In this case, a single image of the volume rendering of the scalar field "noise" will be saved to the file system.

```
conduit::Node scenes;
// scene 1, a single volume plot
// w/ default camera and output res
```

```
scenes["scene1/plots/plot1/type"] = "volume";
scenes["scene1/plots/plot1/params/field"] = "noise";

// scene 2, a pc plot + mesh plot of pipeline 'pl1'
// w/ default camera and output res
scenes["scene2/plot1/type"] = "pseudocolor";
scenes["scene2/plot1/pipeline"] = "pl1";
scenes["scene2/plot1/params/field"] = "noise";

scenes["scene2/plot2/type"]     = "mesh";
scenes["scene2/plot2/pipeline"] = "pl2";
```

Listing 2: Example Scene Descriptions in C++

The second scene definition. "scene2", in Listing 2 indicates that a single image should be created by combining the output of two plots, a pseudocolor plot and a mesh plot. Unlike the first example, the plots that form "scene2" do not use the default data source. Instead, they use the result of a pipeline (see Section 3.1.3), a series of transformations on the published mesh data.

A single scene definition can create more than one image. The rendering parameters are contained in a list, and one image is created for each entry. Further, rendering parameters support camera definitions that output many images. This allows us to target Cinema [3], by providing camera parameters *phi* and *theta*, that produce a total of $phi * theta$ images.

3.1.2 Extracts: Capture Data. Extracts are an abstraction that enables the user to specify how they want to capture their data. In terms of ALPINE, data capture sends data outside the ALPINE infrastructure. Examples include writing out the raw simulation data to the file system, creating HDF5 files, or sending the data off node (e.g., ADIOS [13]).

```
conduit::Node extracts;
// use default pipeline (original mesh)
extracts["ex1/type"] = "hdf5";

// use the result of a pipeline
extracts["ex2/type"] = "hdf5";
extracts["ex2/pipeline"] = "pl2";
```

Listing 3: Example Extract Descriptions in C++.

Extracts, like scenes, can either consume the default data source, all published simulation data, or the result of a pipeline (see Section 3.1.3). Listing 3 shows the declaration of two extracts. The first extract saves the default data source into an HDF5 file, and the second extract saves the result of a pipeline, referenced by pipeline name, into an HDF5 file.

3.1.3 Pipelines: Transform Data. Pipelines allow users to compose filters that transform the published input data into new meshes. This is where users specify typical geometric transforms (e.g., clipping and slicing), field based transforms (e.g., threshold and contour), etc. The resulting data from each Pipeline can be used as input to Scenes or Extracts.

```
conduit::Node pipelines;
// pipeline 1
pipelines["pl1/f1/type"] = "contour";
// filter parameters
conduit::Node contour_params;
contour_params["field"] = "noise";
contour_params["iso_values"] = {0.0, 0.5};
pipelines["pl1/f1/params"] = contour_params;

// pipeline 2
pipelines["pl2/f1/type"] = "threshold";
// filter parameters
conduit::Node thresh_params;
thresh_params["field"]  = "noise";
thresh_params["min_value"] = 0.0;
thresh_params["max_value"] = 0.5;
pipelines["pl2/f1/params"] = thresh_params;

pipelines["pl2/f2/type"]   = "clip";
// filter parameters
conduit::Node clip_params;
clip_params["topology"] = "mesh";
clip_params["sphere/center/x"] = 0.0;
clip_params["sphere/center/y"] = 0.0;
clip_params["sphere/center/z"] = 0.0;
clip_params["sphere/radius"]   = .1;
pipelines["pl2/f2/params/"] = clip_params;
```

Listing 4: Example Pipeline Descriptions in C++.

Listing 4 shows the declaration of two pipelines. The first applies a contour filter to extract two isosurfaces of the scalar field "noise". The second pipeline applies a threshold filter to screen the "noise" field, and then a clip filter to extract the intersection of what remains from the threshold with a sphere.

4 INFRASTRUCTURE

VTK-h, Flow, and Ascent are the three main ALPINE in situ infrastructure software components. VTK-h is a library that will enable development of in situ algorithms that can be deployed in VisIt, ParaView, and Ascent. Flow is a simple data flow library that orchestrates the setup and execution of filter graphs. ALPINE Ascent is a new flyweight in situ runtime that implements the actions supported by the ALPINE Interface. Ascent leverages VTK-h and Flow. This section describes the roles of VTK-h, Flow, and Ascent.

4.1 VTK-h

VTK-h is a stand alone library that implements a distributed-memory layer on top of the VTK-m library [16], which focuses on shared-memory parallelism. The VTK-h library is a collection of distributed-memory algorithms, and VTK-h does not contain an execution model, such as the demand-driven data flow in VTK. The design of VTK-h is intended to facilitate the wrapping of VTK-m algorithms so that they can be included in the execution models of other visualization tools including ALPINE Ascent, ParaView, and VisIt. Consequently, VTK-h serves as a single point of development in

which algorithms can be easily deployed into any toolkit that includes the VTK-h library.

VTK-h heavily leverages VTK-m, and the basic building block of the VTK-h data model is the VTK-m data set. Strawman uses EAVL for its shared-memory parallelism, and just as ALPINE Ascent is the successor to Strawman, VTK-m is the successor to EAVL. A VTK-h data set is a collection of VTK-m data sets along with supporting methods that handle distributed-memory queries (e.g., global scalar ranges). Within VTK-h, most code will directly invoke VTK-m methods to implement algorithms, and while it is possible to directly implement new VTK-m functionality within VTK-h, that functionality is limited to distributed-memory features. For distributed-memory parallelism, VTK-h uses MPI and also includes the DIY [18] toolkit which encapsulates block-based abstractions that are common in distributed-memory problems, and VTK-h uses DIY to implement distributed-memory image compositing.

4.2 Flow

Recall from the prior section that VTK-h does not provide its own execution model. This choice simplifies the VTK-h API and makes it easy to leverage VTK-h within ParaView and VisIt's existing full featured execution models.

Since ALPINE Ascent does not leverage ParaView or VisIt's infrastructure, it needs a basic execution model to support using VTK-h algorithms to carry out the user's requested actions.

Ascent uses a simple data flow library named Flow to efficiently compose and execute VTK-h filters. ALPINE's Flow library is a C++ evolution of the Python data flow network infrastructure used in [9]. It supports declaration and execution of directed acyclic graphs (DAGs) of filters created from a menu of filter types that are registered at runtime. Filters declare a minimal interface, which includes the number of expected inputs and outputs, and a set of default parameters. Flow uses a topological sort to ensure proper filter execution order, tracks all intermediate results, and provides basic memory management capabilities.

The VTK-h algorithms needed by Ascent are wrapped as Flow Filters so they can be executed as part of DAGs composed by Ascent.

Like its Python predecessor, Flow provides support for generic inputs and outputs. Flow provides a mechanism for filters to check input data types at runtime if necessary. Because of this data-type agnostic design, the Flow library does not depend on VTK-h. This provides the flexibility to create filters which can process data in other data models and APIs. This design supports important future use cases, such as creating a filter to refine high-order MFEM [1] meshes into VTK-h data sets for rendering.

4.3 Ascent Runtime

The Ascent Runtime is the layer that sits on top of Flow and beneath the ALPINE In Situ Interface. Ascent's responsibility is to translate a set of actions (e.g., Listings 2, 3, and 4) passed to the ALPINE "Execute" method into a Flow graph. Ascent loops through hierarchy of actions contained in a Conduit Node, and creates a series of Flow filters (i.e., graph nodes) and connects the Flow filters together (i.e., edges). Figure 2 shows the graph representation Ascent creates given the actions described by Listings 2, 3, and 4. When the 'execute' action is processed, Ascent executes the graph.

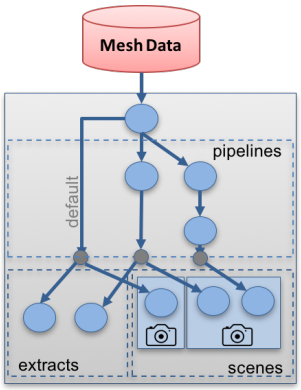

Figure 2: A Flow graph created from the actions described by Listings 2, 3, and 4.

	Sim	Vis	Percentage Vis
CPU	32.5 s	1.5 s	4.4%
GPU	32.0 s	1.3 s	3.7%

Table 1: Average time in seconds spent running a simulation and executing visualization per cycle as described in Section 5.

5 RESULTS

We evaluated the performance of the ALPINE infrastructure using the included Kripke physics proxy-application that models deterministic neutron transport. Running on 48 nodes equipped with 16-core Intel Xeon CPUs and NVIDIA K40m GPUs, we ran Kripke with a total problem size of 100M zones. Each cycle, we clipped the mesh in half, created a pseudocolor plot of the clipped mesh, and combined the result with a volume rendering of the mesh. It is worth noting that this rendering is not be possible in Strawman, since Strawman does not support multiple plots and does not contain any filters. The average time per cycle spent performing simulation and visualization is shown in Table 1, and the resulting image is shown in Figure 3. Visualization results in less than 5% of total runtime.

6 CONCLUSION

This paper introduced the ALPINE in situ interface and ALPINE's infrastructure components VTK-h, Flow, and Ascent. The new interface and these components enable simulation users to easily execute in situ visualization and analysis algorithms on modern supercomputing architectures. The

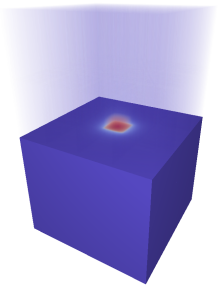

Figure 3: Image produced using the operations described in Section 5.

ALPINE interface and Ascent runtime are evolutions of the Strawman in situ visualization mini-app. ALPINE is aimed at production, and it includes significant additions to support a wider range of use cases than Strawman. We demonstrated this with a basic example of transforming data and rendering multiple plots within a Kripke simulation. For future work, we will continue to develop VTK-h and ALPINE capabilities, and we will work with ECP application teams to deploy in situ visualization and analysis algorithms to simulation users via ALPINE.

7 ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC (LLNL-CONF-737832). Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow other to do so, for United States Government purposes.

REFERENCES

[1] 2017. MFEM: Modular finite element methods. (2017). http://mfem.org

[2] James Ahrens, Berk Geveci, and Charles Law. 2005. Paraview: An end-user tool for large data visualization. *The Visualization Handbook* 717 (2005).

[3] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. 2014. An Image-based Approach to Extreme Scale in Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, Piscataway, NJ, USA, 424–434. https://doi.org/10.1109/SC.2014.40

[4] Utkarsh Ayachit et al. 2016. The SENSEI Generic in Situ Interface. In *Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV '16)*. 40–44.

[5] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. 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*. ACM, 25–29.

[6] Andrew C. Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, and E. Wes Bethel. 2016. *In Situ* Methods, Infrastructures, and Applications on High Performance Computing Platforms, a State-of-the-art (STAR) Report. *Computer Graphics Forum, Proceedings of Eurovis 2016* 35, 3 (June 2016). LBNL-1005709.

[7] Hank Childs et al. 2012. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*. CRC Press/Francis–Taylor Group, 357–372.

[8] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing* 22, 6 (1996), 789–828.

[9] C. Harrison, P. Navràtil, M. Moussalem, M. Jiang, and H. Childs. 2012. Efficient Dynamic Derived Field Generation on Many-Core Architectures Using Python. In *Proceedings of the 2012 Workshop on Python for High Performance and Scientific Computing (PyHPC 2012)*. 583–592. https://doi.org/10.1109/SC.Companion.2012.82

[10] Lawrence Livermore National Laboratory. 2017. Conduit: Simplified Data Exchange for HPC Simulations. (2017). https://software.llnl.gov/conduit/

[11] Lawrence Livermore National Laboratory. 2017. Conduit: Simplified Data Exchange for HPC Simulations - Conduit Blueprint. (2017). https://software.llnl.gov/conduit/blueprint.html

[12] Matthew Larsen, Eric Brugger, Hank Childs, Jim Eliot, Kevin Griffin, and Cyrus Harrison. 2015. 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 (ISAV2015)*. ACM, New York, NY, USA, 30–35. https://doi.org/10.1145/2828612.2828625

[13] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. 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)*. ACM, New York, NY, USA, 15–24. https://doi.org/10.1145/1383529.1383533

[14] Jeremy S. Meredith, Sean Ahern, Dave Pugmire, and Robert Sisneros. 2012. EAVL: The Extreme-scale Analysis and Visualization Library. In *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association.

[15] Paul Messina. 2017. The Exascale Computing Project. *Computing in Science & Engineering* 19, 3 (2017), 63–67.

[16] 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. 2016. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A)* 36, 3 (May/June 2016), 48–58.

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

[18] Tom Peterka, Robert Ross, Wesley Kendall, Attila Gyulassy, Valerio Pascucci, Han-Wei Shen, Teng-Yok Lee, and Abon Chaudhuri. 2011. Scalable Parallel Building Blocks for Custom Data Analysis. In *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11*. Providence, RI.

[19] R Core Team. 2000. R language definition. *Vienna, Austria: R foundation for statistical computing* (2000).

[20] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel in Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)*. 101–109.