libIS: A Lightweight Library for Flexible In Transit Visualization

Will Usher*[1,2], Silvio Rizzi[3], Ingo Wald[2,4], Jefferson Amstutz[2], Joseph Insley[3], Venkatram Vishwanath[3], Nicola Ferrier[3], Michael E. Papka[3], and Valerio Pascucci[1]
[1]SCI Institute, University of Utah, [2]Intel Corporation
[3]Argonne National Laboratory, [4]Now with Nvidia
*will@sci.utah.edu

ABSTRACT

As simulations grow in scale, the need for *in situ* analysis methods to handle the large data produced grows correspondingly. One desirable approach to *in situ* visualization is *in transit* visualization. By decoupling the simulation and visualization code, *in transit* approaches alleviate common difficulties with regard to the scalability of the analysis, ease of integration, usability, and impact on the simulation. We present libIS, a lightweight, flexible library which lowers the bar for using *in transit* visualization. Our library works on the concept of abstract regions of space containing data, which are transferred from the simulation to the visualization clients upon request, using a client-server model. We also provide a SENSEI analysis adaptor, which allows for transparent deployment of *in transit* visualization. We demonstrate the flexibility of our approach on batch analysis and interactive visualization use cases on different HPC resources.

CCS CONCEPTS

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

KEYWORDS

In Transit Visualization, Scientific Visualization, SENSEI

ACM Reference Format:
Will Usher, Silvio Rizzi, Ingo Wald, Jefferson Amstutz, Joseph Insley, Venkatram Vishwanath, Nicola Ferrier, Michael E. Papka, and Valerio Pascucci. 2018. libIS: A Lightweight Library for Flexible In Transit Visualization. In *ISAV: In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV '18), November 12, 2018, Dallas, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3281464.3281466

1 INTRODUCTION

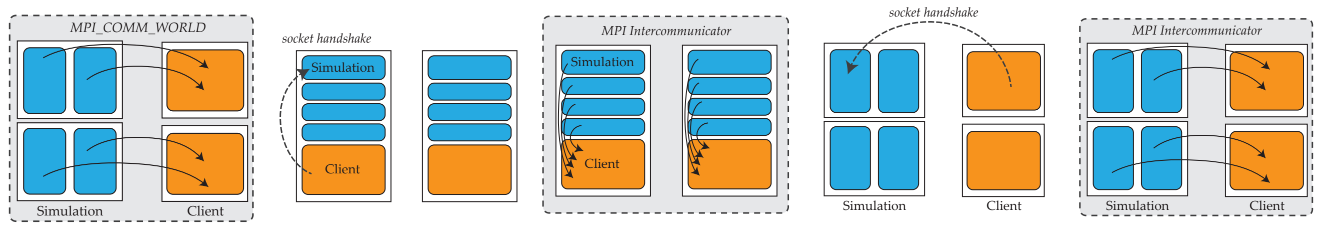
The increasing gap between FLOPs and I/O has motivated the development of *in situ* visualization [3], which has been identified as a key technology to enable science at exascale [1]. A range of *in situ* visualization and analysis approaches have been proposed, from simulation-tailored methods [5, 12, 17, 19, 23–26], to general purpose frameworks, such as ParaView Catalyst [4, 11], VisIt LibSim [8, 22], ADIOS [16], and GLEAN [20]. The SENSEI [2] project addresses the portability and reusability challenges introduced by this growth in frameworks, by enabling simulations to implement a single *in situ* integration which can then work with different visualization backends.

Two key axes by which an *in situ* integration can be classified are its *proximity* and *access* [7]. Proximity describes how close the simulation and visualization are run, e.g., in the same process or a separate one, potentially on another node. Access refers to how the visualization gains access to the simulation data, e.g., directly via sharing pointers, or indirectly through a copy of the data. *In situ* methods can be roughly categorized as either "tightly-coupled" (same process, direct access), or "loosely-coupled" (separate process, indirect access). Loosely-coupled approaches are also frequently referred to as *in transit*, which is the terminology we adopt in this paper.

While a tightly-coupled approach brings clear benefits by eliminating data copies and not requiring coordination between separate processes, *in transit* approaches can offer specific desirable advantages at scale [13]. By decoupling the simulation and visualization, *in transit* methods allow for greater flexibility in when, where, and at what scale, the visualization code is run (e.g., Bennett et al. [5]), ease integration effort and usability, and reduce the impact of the visualization on the simulation. Furthermore, with *in transit* visualization it is possible to run the visualization sporadically as a separate process or job, reducing resource requirements. These features are especially desirable for exascale simulations, as any scalability limitations of the visualization code will not impact the simulation, nor will faults in the visualization crash the simulation.

However, a set of algorithmic and practical challenges remain to deploying *in transit* visualization in practice. The bulk of work on general *in situ* infrastructures has targeted tightly-coupled *in situ* [11, 22], or has implemented *in transit* by re-purposing I/O APIs [16, 20] and performing data aggregation with a general data communication and query system (e.g., DataSpaces [10]). While the latter approach is desirable if the simulation is already using the I/O API being repurposed, doing so for a new integration with the express goal of *in transit* visualization may not be, requiring additional implementation effort and complexity.

The contribution of this paper is libIS, a flexible, lightweight library for *in transit* visualization. Our library alleviates common challenges with *in transit* visualization by managing data aggregation and coordination with the simulation. To enable low overhead data aggregation from M simulation ranks to N visualization ranks, we introduce a generic representation of the distributed simulation domains. Visualization code run with libIS can be run on either the same nodes as the simulation, or a distinct set of nodes, or even within the same MPI communicator, when using MPI MPMD. Moreover, the visualization can be run sporadically, connecting and disconnecting seamlessly from the simulation as desired to provide



(a) MPI-multilaunch mode. **(b) Shared node socket-handshake mode.** **(c) Separate node socket-handshake mode.**

Figure 1: LibIS supports a variety of runtime configurations, making it applicable to a wide range of *in situ* use cases. The socket-handshake modes (b, c) can disconnect and reconnect as desired, allowing for sporadic execution of the visualization code.

on-demand, flexible visualization. To ease the integration of libIS into existing code, we provide a SENSEI data adaptor. Our adaptor transfers data to a set of visualization clients, which can then run any Catalyst, LibSim, etc., pipelines which would have previously been run tightly-coupled with SENSEI.

2 RELATED WORK

There has been a substantial amount of work in the last decade focusing on *in situ* analysis and visualization, covering a wide range of approaches for integrating these tasks into simulations. For a detailed discussion of the current space of *in situ* visualization we refer to the comprehensive surveys by Bauer et al. [3] and Childs et al. [7], and summarize the work most related to our own below.

From the simulation's perspective, I/O and data transfer for *in transit* visualization share many similarities. As such, one approach to easily integrate *in transit* visualization is to integrate it within the I/O library used by the simulation, e.g., ADIOS [16] or GLEAN [20]. These approaches often introduce a distributed data transfer and restructuring technique into the I/O library, e.g., PreDatA [27], DataSpaces [10], FlexIO [28] or FlexPath [9]. By leveraging the metadata provided to the I/O library, these methods can aggregate and restructure the data to match the parallelism of the visualization process.

Although these approaches provide a large amount of flexibility in how and where the visualization routines can be run, this comes at the cost of increased library complexity. Furthermore, the restructuring of the data incurs additional runtime cost [19], and may require additional effort to support new mesh types. Moreover, if the simulation is not already using the I/O library that the *in transit* system is integrated into, migrating the simulation to use a different I/O library can be a significant effort. As a result, new integrations of *in transit* visualization for prototyping or evaluation can require unnecessary additional effort. Recent work by Larsen et al. [14, 15] has investigated the development of lightweight *in situ* infrastructures, providing easy to use, low overhead, tightly-coupled *in situ*. With libIS we provide a similar simplification for *in transit* visualization.

3 THE LIB-IS LIBRARY

LibIS is specifically designed for asynchronous, *in transit* visualization, where the simulation and visualization run decoupled from each other, and communicate over MPI. The library provides a lightweight method to communicate data between a distributed simulation, acting as a server, and a distributed visualization client. The simulation and visualization can be run on the same nodes, separate nodes, or within the same MPI launch command (Figure 1).

LibIS is split into a simulation-side and client-side library. The simulation-side library, coupled to the simulation either through our SENSEI interface or directly, exposes the simulation as a data-server, while the client-side library queries this server for new timesteps.

3.1 Connecting the Simulation and Client

LibIS provides two methods for connecting the simulation and client, either over an existing MPI communicator or via a socket-handshake. The first method allows users to specify a previously created communicator to be used for communication. For example, `MPI_COMM_WORLD` can be used when launching multiple programs simultaneously with MPI (Figure 1a). When using the socket-handshake method, the simulation always listens on a background thread for a new client, which opens an MPI port and sends this information to the simulation over a TCP socket. The simulation and client then set up an MPI intercommunicator using this MPI port (Figures 1b and 1c).

3.2 Querying Data

After the communicator between the simulation and client has been setup, rank 0 of the simulation posts a `MPI_Irecv` to receive the next command the client wants to execute, either query data or disconnect. The state of this receive is checked each time the simulation calls `libISProcess` (each timestep). When a new command is received, it is broadcast to the other simulation ranks and processed collectively.

Given a run with M simulation ranks and N client ranks, where $M \geq N$, each client will receive data from $\frac{M}{N}$ simulation ranks, with any remainder ranks assigned evenly among the clients. Each simulation rank sends its data independently to its client using MPI point-to-point communication, allowing for better communication locality. In contrast to more complex approaches which restructure the data to match the parallelism of the clients (e.g., ADIOS, FlexIO), libIS keeps the original simulation data distribution, returning to each client a list of $\frac{M}{N}$ regions. By avoiding more expensive data redistribution, libIS incurs little additional transfer cost, and requires minimal information about the data being transferred. Although libIS requires $M \geq N$, this is typically the case in practice.

3.3 Simulation Library

The simulation-side library exposes a C API, allowing it to be easily integrated into simulations written in a broad range of languages. The API is used to pass libIS information about the simulation state, including the world, local and ghost bounds (if applicable), along with the rank's field and particle data. This is done by creating and filling out a `libISSimState` using the following API calls. The configured simulation state consists of a list of pointers to the simulation data, and some metadata describing it, providing a zero-copy interface.

```
void libISSetLocalBounds(libISSimState *state,
  const libISBox3f box); // World/Ghost identical
void libISSetField(libISSimState *state,
  const char *name, const uint64_t dims[3],
  const libISDType type, const void *data);
void libISSetParticles(libISSimState *state,
  const uint64_t nLocal, const uint64_t nGhost,
  const uint64_t stride, const void *data);
void libISProcess(const libISSimState *state);
```

The current API only supports regular 3D grid fields and array-of-structures particle layouts; however, the mapping from simulation ranks to client ranks requires no spatial information or data redistribution, making it straightforward to add support for additional mesh types and particle layouts.

The final call, `libISProcess`, is called each timestep, and passes the filled out simulation state to libIS to send to any clients which have requested data. If the simulation has clients requesting data, it is sent directly from the passed pointers, otherwise control is returned back to the simulation.

3.4 Client Library

The client-side library provides a C++ API to query data from the simulation. In contrast to the simulation-side, where `libISProcess` is non-blocking, the client's query method will block until it receives data for the new timestep. Having a blocking query simplifies the client implementation in most cases, where there is nothing to do besides wait until the data has arrived. Interactive applications can call query on a background thread to prevent blocking the application (e.g., Section 4.2).

After the query has completed, the client will receive the simulation data and metadata from each of its assigned $\frac{M}{N}$ simulation ranks. This is returned as a vector of `SimState` structures.

```
struct SimState {
  libISBox3f world, local, ghost;
  int simRank;
  std::unordered_map<std::string, Field> fields;
  Particles particles;
};
std::vector<SimState> query();
```

The fields member is the list of field data sent by the simulation, indexed by the field name. The particles member contains the array-of-structures layout particle data, if any.

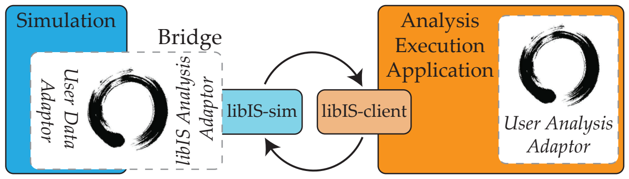


Figure 2: The architecture of our *in transit* visualization execution system for SENSEI.

3.5 SENSEI Analysis Adaptor

To ease integration effort for simulations or visualization code already using SENSEI [2], we also provide a SENSEI analysis adaptor and *in transit* analysis execution application. A SENSEI *in situ* system consists of a *bridge*, *data adaptor* and *analysis adaptor*. The first two are tailored to the simulation, and responsible for converting its data representation to the VTK representation used by SENSEI. The VTK representation is then passed to an analysis adaptor, which runs the desired *in situ* visualization. To transparently provide *in transit* visualization to SENSEI users, we provide a libIS *analysis adaptor* and an *in transit* analysis execution application which runs the user's analysis adaptors (Figure 2).

Our libIS analysis adaptor takes the VTK data from SENSEI, constructs the equivalent `libISSimState`, and calls `libISProcess` to send the data to our *in transit* analysis application. The *in transit* analysis application uses libIS to query data from the simulation, constructs the equivalent VTK representation, and passes it on to the user's analysis adaptors. The conversion back to the VTK representation from libIS's uses VTK's zero-copy arrays, avoiding additional data copies.

While running the SENSEI analysis *in transit* with libIS in a $1:1$ configuration is straightforward, doing so for an $M > N$ configuration is less so. SENSEI's existing data model expects one region per-process, i.e., a tightly-coupled *in situ* use case, or a data restructured *in transit* use case, requiring some special treatment for $M > N$ configurations. For example, an analysis adaptor which computes a global reduction over the data, such as the min or max, or a histogram, may not produce the correct result if each analysis rank ran a `for` loop over its assigned regions. To avoid requiring users to modify their analysis code, we pass the data to the user's SENSEI analysis code as a `vtkMultiBlockDataset`, giving the appearance of a single simulation rank with $\frac{M}{N}$ blocks of data.

4 RESULTS

We evaluate the performance and scalability of libIS using LAMMPS as our simulation code. We demonstrate our SENSEI analysis adaptor on the analysis algorithms and infrastructures provided by the SENSEI configurable analysis adaptor (Section 4.1). To compare the different configurations libIS supports, we implement an interactive *in transit* viewer (Section 4.2), and examine the simulation and visualization performance, and data transfer bandwidth in each configuration (Section 4.3). We run our benchmarks on *Theta* at Argonne National Laboratory, and *Stampede2* at the Texas Advanced Computing Center. *Theta* and *Stampede2* use the Intel® Xeon Phi™ Knight's Landing (KNL) processors. *Stampede2* has an additional partition with Intel® Xeon® Skylake (SKX) processors; however, our evaluations were done only on the KNLs.

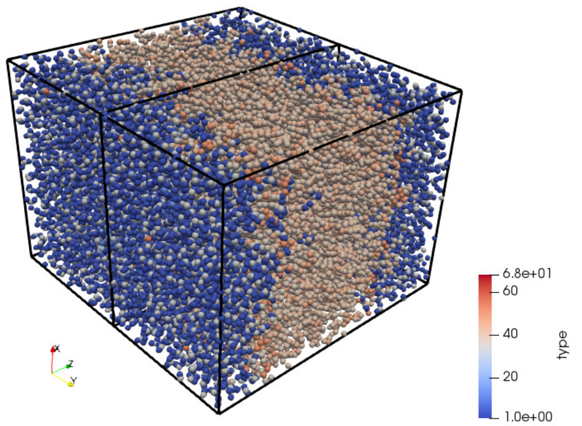


Figure 3: Interactive visualization of a LAMMPS simulation with two ranks in ParaView. The data is passed *in transit* through libIS and our SENSEI interface, saved to disk using the VTK I/O backend in SENSEI, and visualized in a post-process with ParaView.

4.1 SENSEI Integration

We integrate our SENSEI interface into LAMMPS by taking advantage of existing mechanisms in LAMMPS for coupling it with other simulation codes [18]. We provide a driver application which acts as a thin wrapper over LAMMPS, along with a bridge and data adaptor for SENSEI. The driver takes the user's input file and LAMMPS commands as before, and inserts an additional callback to be run by LAMMPS each timestep. This callback passes the data to our SENSEI data adaptor, which converts the particles to a `vtkPolyData`, with additional attributes (e.g., atom type and id) treated as 1D fields.

The VTK data is then passed to our libIS analysis adaptor, which sends the data to our analysis execution application. Our analysis application runs SENSEI's configurable analysis, which can connect to *in situ* algorithms in SENSEI, and multiple *in situ* infrastructures, including Catalyst, LibSim, and ADIOS. We validate our implementation with the SENSEI histogram algorithm, and verify that the results are correct for multiple MPI ranks. We also demonstrate the VTK I/O capabilities of SENSEI, and write out the simulation data, which we visualize in ParaView (Figure 3).

4.2 Interactive Visualization

An advantage of *in transit* visualization is that the visualization and simulation can execute in parallel, allowing for interactive *in situ* visualization, without blocking the simulation. This is in contrast to tightly-coupled interactive visualization approaches, which must pause the simulation. Furthermore, by using libIS's connect and disconnect support, such interactive viewers can be used sporadically, e.g., to check in on a long running simulation, or begin monitoring after some event, without requiring additional nodes be set aside for the entire run. To evaluate interactive visualization with libIS we implemented a client-server based renderer for LAMMPS simulations using OSPRay [21] for rendering.

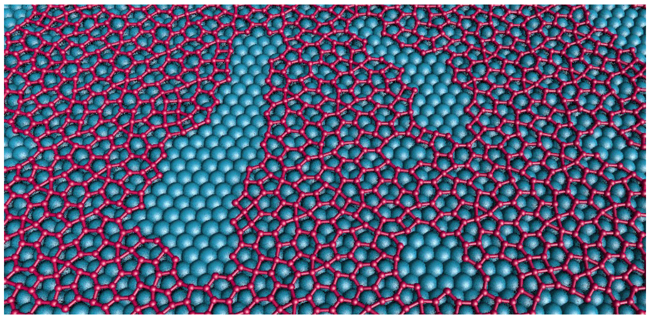


Figure 4: Interactive *in situ* visualization of a 172k atom simulation of silicene formation [6] with 128 LAMMPS ranks sending to 16 OSPRay renderer ranks, all executed on Theta in the `mpi-multi` configuration. When taking four ambient occlusion samples per-pixel, our viewer averages 7FPS at 1024×1024.

The render server runs on the HPC resource, and uses libIS to query data, and OSPRay's data-distributed API to render it. The render server continues querying data as the simulation runs, enabling the user to watch the simulation state evolve over time. Along with rendering the data, the server also supports running some local VTK pipelines to process the data, e.g., to compute bonds between atoms. The client is run on the user's desktop, and connects to the render server over a socket. The client sends the user's camera position, and receives back rendered images as JPGs. An example image from the renderer is shown in Figure 4.

4.3 Performance of Different Configurations

We evaluate the weak scaling of libIS and our interactive viewer by replicating the LAMMPS Lennard-Jones benchmark problem to keep a constant 2 million atoms per-node, with each renderer rank receiving from 16 simulation ranks. The benchmarks are rendered at a 1024×1024 resolution as the camera is rotated around the data set. We find that libIS and the OSPRay rendering client provide good weak scaling for both data-transfer bandwidth (Figure 5), and rendering performance (Figure 6). It is worth noting that the shared and separate node configurations do not have much effect on either the bandwidth or render time; however, the MPI multilaunch mode on Theta yields better performance at some node counts. While we would expect the MPI MPMD (mpi-multi) to perform similar to the separate node case, we could not run this configuration on Theta to compare, due to the machine not supporting the necessary MPI APIs.

We find that libIS has little impact on the simulation. In the separate node and mpi-multi configurations (which use a separate set of nodes for rendering) we find that LAMMPS takes an average of 6.6% longer to compute each timestep. When rendering on the same nodes as the simulation we find a larger impact due to oversubscribing the nodes, with LAMMPS taking 16.1% longer per-timestep.

5 CONCLUSION

We have presented libIS, a lightweight, flexible library which lowers the barrier to *in transit* visualization. By combining our library and our provided SENSEI analysis adaptor and execution application,

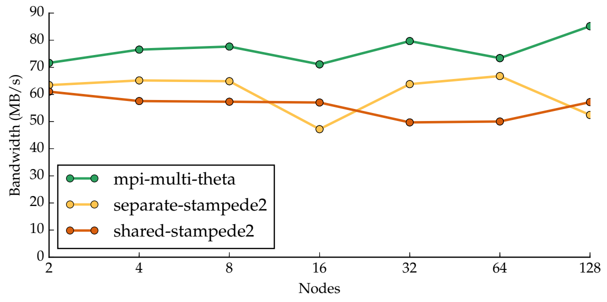


Figure 5: Peak bandwidth per-client weak scaling of different configurations. Clients communicate independently with their simulation ranks, allowing for good weak scaling.

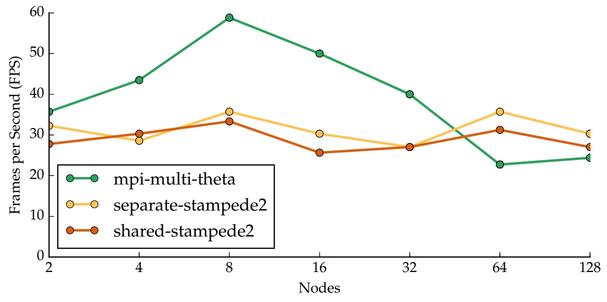


Figure 6: Rendering performance weak scaling of different configurations, without ambient occlusion. Our OSPRay-based renderer provides good weak scaling, and high frame rates.

existing tightly-coupled *in situ* workflows can be transparently converted to run *in transit*. Running the analysis *in transit* incurs little impact on the simulation, and offers greater flexibility in how the visualization can be run. In addition to batch analysis, we also demonstrated libIS's applicability to interactive visualization as the simulation evolves, which, due to the decoupled execution, can be done without blocking the simulation. We release both libIS and our SENSEI adaptor and application open source to allow developers to easily integrate them into their simulations (Appendix A).

In future work, it would be interesting to investigate restructuring the data to match the visualization process's parallelism. Although this comes at a runtime cost, restructuring has the potential to reduce bandwidth and memory use by removing duplicated ghost regions, and simplify the implementation of some analysis tasks. To provide better support for simulations with a long timestep it is also interesting to consider buffering the previous timestep in memory, or on a burst buffer, to make data immediately available to clients when they connect.

ACKNOWLEDGMENTS

We would like to thank Mathew Cherukara, Badri Narayanan, Henry Chanc and Subramanian Sankaranarayanan at Argonne National Laboratory for sharing their simulation data and providing insightful discussion.

This work is supported in part by the Intel Parallel Computing Centers Program, NSF: CGV Award: 1314896, NSF:IIP Award: 1602127, NSF:ACI Award: 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC Award: 1842042. This work was supported by and used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357, through the project "Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery." The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported in this paper.

REFERENCES

[1] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Ken Joy, Quincey Koziol, Gerald Lofstead, Jeremy S. Meredith, Kenneth Moreland, George Ostrouchov, Michael Papka, Venkatram Vishwanath, Matthew Wolf, Nicholas Wright, and Kensheng Wu. 2011. *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization.*

[2] Utkarsh Ayachit, Brad Whitlock, Matthew Wolf, Burlen Loring, Berk Geveci, David Lonie, and E. Wes Bethel. 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).*

[3] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. 2016. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum* (2016).

[4] Andrew C. Bauer, Berk Geveci, and Will Schroeder. 2015. *The Catalyst User's Guide v2.0.* Kitware Inc.

[5] Janine C. Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, and Valerio Pascucci. 2012. 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.*

[6] Mathew J. Cherukara, Badri Narayanan, Henry Chan, and Subramanian K. R. S. Sankaranarayanan. 2017. Silicene Growth Through Island Migration and Coalescence. *Nanoscale* (2017).

[7] H. Childs, S. D. Ahern, J. Ahrens, A. C. Bauer, J. Bennett, E. W. Bethel, P.-T. Bremer, E. Brugger, J. Cottam, M. Dorier, S. Dutta, J. Favre, T. Fogal, S. Frey, C. Garth, B. Geveci, W. F. Godoy, C. D. Hansen, C. Harrison, B. Hentschel, J. Insley, C. R. Johnson, S. Klasky, A. Knoll, J. Kress, M. Larsen, J. Lofstead, K.-L. Ma, P. Malakar, J. Meredith, K. Moreland, P. Navrátil, P. O'Leary, M. Parashar, V. Pascucci, J. Patchett, T. Peterka, S. Petruzza, N. Podhorszki, D. Pugmire, M. Rasquin, S. Rizzi, D. H. Rogers, S. Sane, F. Sauer, R. Sisneros, H.-W. Shen, W. Usher, R. Vickery, V. Vishwanath, I. Wald, R. Wang, G. H. Weber, B. Whitlock, M. Wolf, H. Yu, and S. B. Ziegeler. 2018. The In Situ Terminology Project. *(Under Submission)* (2018).

[8] H. Childs, K.-L. Ma, H. Yu, B. Whitlock, J. Meredith, J. Favre, S. Klasky, N. Podhorszki, K. Schwan, M. Wolf, M. Parashar, and F. Zhang. 2012. In Situ Processing. In *High Performance Visualization: Enabling Extreme-Scale Scientific Insight.*

[9] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. 2014-05. Flexpath: Type-Based Publish/Subscribe System for Large-Scale Science Analytics.

[10] Ciprian Docan, Manish Parashar, and Scott Klasky. 2012. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. *Cluster Computing* (2012).

[11] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K.E. Jansen. 2011. The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In *Symposium on Large Data Analysis and Visualization (LDAV).*

[12] Hanqi Guo, Tom Peterka, and Andreas Glatz. 2017. In situ magnetic flux vortex visualization in time-dependent Ginzburg-Landau superconductor simulations. In *Pacific Visualization Symposium (PacificVis).*

[13] James Kress, Scott Klasky, Norbert Podhorszki, Jong Choi, Hank Childs, and David Pugmire. 2015. Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV 2015).*

[14] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. 2017. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17).*

[15] 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)*.
[16] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. 2009. Adaptable, Metadata Rich IO Methods for Portable High Performance IO. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium On.*
[17] Silvio Rizzi, Mark Hereld, Joseph Insley, Michael E. Papka, Thomas Uram, and Venkatram Vishwanath. 2015. Large-Scale Co-Visualization for LAMMPS Using Vl3. In *2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*.
[18] Sandia National Laboratories. Accessed 2018. Coupling LAMMPS to other codes.
[19] Will Usher, Ingo Wald, Aaron Knoll, Michael Papka, and Valerio Pascucci. 2016. In Situ Exploration of Particle Simulations with CPU Ray Tracing. *Supercomputing Frontiers and Innovations* (2016).
[20] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E. Papka. 2011. Topology-aware Data Movement and Staging for I/O Acceleration on Blue Gene/P Supercomputing Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.
[21] Ingo Wald, Greg P. Johnson, Jefferson Amstutz, Carson Brownlee, Aaron Knoll, Jim Jeffers, Johannes Günther, and Paul Navrátil. 2017. OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2017).
[22] 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 '11)*.
[23] Wathsala Widanagamaachchi, Karl Hammond, Li-Ta Lo, Brian Wirth, Francesca Samsel, Christopher Sewell, James Ahrens, and Valerio Pascucci. 2015. Visualization and Analysis of Large-Scale Atomistic Simulations of Plasma-Surface Interactions. In *Eurographics Conference on Visualization (EuroVis) - Short Papers*.
[24] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. 2011. In-situ Sampling of a Large-Scale Particle Simulation for Interactive Visualization and Analysis. *Computer Graphics Forum* (2011).
[25] Jonathan Woodring, Mark Petersen, Andre Schmeiber, John Patchett, James Ahrens, and Hans Hagen. 2016. In Situ Eddy Analysis in a High-Resolution Ocean Climate Model. *IEEE Transactions on Visualization and Computer Graphics* (2016).
[26] Hongfeng Yu, Chaoli Wang, Ray W. Grout, Jacqueline H. Chen, and Kwan-Liu Ma. 2010. In Situ Visualization for Large-scale Combustion Simulations. *IEEE Computer Graphics and Applications* (2010).
[27] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. 2010. PreDatA–Preparatory Data Analytics on Peta-Scale Machines. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*.
[28] Fang Zheng, Hongbo Zou, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Jai Dayal, Tuan-Anh Nguyen, Jianting Cao, Hasan Abbasi, Scott Klasky, Norbert Podhorszki, and Hongfeng Yu. 2013. FlexIO: I/O Middleware for Location-Flexible Scientific Data Analytics.

A ARTIFACT DESCRIPTION APPENDIX

To allow for reproducing our results, and make libIS easily available to users, we released both libIS and our various test applications and benchmarks open-source on GitHub. The libIS library is available on at https://github.com/ospray/libIS and our interactive viewer application can be found at https://github.com/ospray/ospray_senpai. The SENSEI adaptor used in the experiments is being merged into the main SENSEI repo, the development branch can be found at https://gitlab.kitware.com/sensei/sensei/tree/lammps.

In our evaluation we encountered additional difficulty on Theta, when trying to connect our viewer client to the render server on the compute nodes. Due to the system's network configuration, it is not possible to SSH tunnel to or from the compute nodes directly, requiring instead to first tunnel to the login node, then the MOM node (which runs the job scripts). From the MOM node we then run `socat` to bridge the SSH tunnel to a TCP connection on the compute node. We have encapsulated this process in a bash script, which we make available on GitHub to aid our fellow researchers: https://github.com/Twinklebear/theta-tunnel.

At the time of submission our viewer used an internal branch of OSPRay with improvements to the distributed renderer, which added support for local ambient occlusion with ghost zones, and significantly improved performance. This branch has now been merged into OSPRay and is available on the public development branch on GitHub: https://github.com/ospray/ospray/tree/devel.