Inshimtu – A Lightweight In Situ Visualization "Shim"

James Kress[1][0000−0002−9706−6182], Glendon Holst[1], Hari Prasad Dasari[1][0000−0003−0628−4481], Shehzad Afzal[1][0000−0002−9712−3618], Ibrahim Hoteit[1][0000−0002−3751−4393], and Thomas Theußl[1][0000−0001−7907−7382]

King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia {james.kress, glendon.holst, hari.dasari, shehzad.afzal, ibrahim.hoteit, thomas.theussl}@kaust.edu.sa

Abstract. In situ visualization and analysis is a valuable yet under utilized commodity for the simulation community. There is hesitance or even resistance to adopting new methodologies due to the uncertainties that in situ holds for new users. There is a perceived implementation cost, maintenance cost, risk to simulation fault tolerance, potential lack of scalability, a new resource cost for running in situ processes, and more. The list of reasons why in situ is overlooked is long. We are attempting to break down this barrier by introducing Inshimtu. Inshimtu is an in situ "shim" library that enables users to try in situ before they buy into a full implementation. It does this by working with existing simulation output files, requiring no changes to simulation code. The core visualization component of Inshimtu is ParaView Catalyst, allowing it to take advantage of both interactive and non-interactive visualization pipelines that scale. We envision Inshimtu as stepping stone to show users the value of in situ and motivate them to move to one of the many existing fully-featured in situ libraries available in the community. We demonstrate the functionality of Inshimtu with a scientific workflow on the Shaheen II supercomputer.

Inshimtu is available for download at: https://github.com/kaust-vislab/Inshimtu-basic.

1 Introduction

With the launch of the worlds first exascale computer, the benefits that in situ visualization are poised to provide to the community at large are only going to grow. With each new generation of supercomputer the gap between compute and storage capacity continues to grow [3, 13]. In order to combat this trend, simulation teams often turn down the frequency of their writes to disk, further opening the possibility of missing important features, while also causing video transitions over time to become jittery. As a whole, the in situ visualization community has created and maintains a number of different in situ visualization packages that are capable of not only speeding up simulations time to solution, but also will

allow them to take advantage of higher temporal and spatial resolutions. However, the uptake of in situ technologies by simulation teams is still low, with users concerned over things such as resilience, data integrity, integration time, and code maintenance [21]. The question we must ask then is how aid simulation teams to overcome these issues whether perceived or actual?

In response to this question we present Inshimtu, an in situ shim, that allows simulation codes to experience some of the benefits of in situ with no changes required to their simulations. The idea behind this library is a low barrier try before you buy approach, where simulations can see, strategize about, and understand the different types of in situ and how they could potentially be beneficial, all with very little effort. This ease of use is accomplished through the use of file based in situ, meaning Inshimtu reads directly from the simulation output, and then all of the visualization and analysis is handled by our integration with ParaView Catalyst. The visualization pipelines and scripts used with Inshimtu are exactly the same as those used by the fully simulation-embedded version of Catalyst. So in the end, if Catalyst is chosen as the way forward for a full in situ integration by the simulation team, their existing scripts can be reused. In practice the only modification to existing simulation setup required is one additional line in the batch file to start Inshimtu in the background before the simulation. Inshimtu is thus a stepping stone from basic to complete in-situ visualization, with it's primary goal to be a low barrier to entry teaching tool to educate and inform simulation developers about the benefits they are missing by ignoring in situ techniques.

In this paper we describe Inshimtu and an associated use case on our Shaheen II supercomputer. In Section 3 we describe the design and implementation, as well as the underlying technologies used by Inshimtu. In Section 4 we present examples of Inshimtu being used for visualization and analysis tasks. We also discuss the easy instrumentation and configuration process which makes Inshimtu have a low barrier to entry. Finally in Section 5 we present a summary and provide some thoughts on future directions.

2 Related Work

In situ visualization is a fragmented space, with many different tools and frameworks available, each with their own dedicated use cases and implementation strategies [24]. Generally, a simulation team would need to pick either an in-line [8] or in-transit [21, 26] visualization tool and then redesign their code and I/O strategy around this new framework. This approach can lead to very powerful and performant visualization [14] that can not only save time vs. pos-hoc visualization but also cost [2, 7, 22, 23, 28].

Some of the most current and pervasive tools include: LibSim [31] is a library that allows simulations to use the full set of features of the VisIt [12] visualization tool. ParaView Catalyst [6] offers a similar in situ functionality for the ParaView [4] visualization tool. ADIOS2 [20] is an I/O middleware library that exposes both in-line and in-transit paradigms to a simulation through a

POSIX-like API. Its in-transit capabilities are provided by a number of different data transport methods, including DataSpaces [16], DIMES [33], and Flex-Path [15]. Damaris/Viz [17] provides both in-line and in-transit visualization using the Damaris I/O middleware. Ascent [25] is a fly-weight in situ infrastructure that supports both distributed-memory and shared-memory parallelism. SENSEI [5, 9, 10] is a generic data interface that allows transparent use of the LibSim, Catalyst, and ADIOS in situ frameworks. UDJ [18] is a communication library for transporting distributed data between parallel high performance computing applications and has demonstrated a proof of concept using an extension of Inshimtu, which is based on streaming data vs. file synchronization. Freeprocessing [19] is an in situ interposition library that works in either a synchronous or asynchronous mode. Initialize Compute Analyze Render Update Steer (ICARUS) [30] is a ParaView plug-in for in situ visualization and computational steering using a shared memory mapped HDF5 file for data access.

However, the simulation communities uptake of in situ technologies has been slow. There are a long list of reasons typically given why in situ is not used, but it generally boils down to the fact that visualization has traditionally been performed as a post-processing task, where simulation outputs are read from disk by a visualization and analysis tool. This completely un-couples the simulation from the visualization and doesn't distract the simulation team from their true goal, of getting the perfect simulation. As newer and more simulation friendly in situ integration methods are developed (such as visualization as a service [27] and Fides [29]) the resistance for adoption by simulation stake holders will likely decrease.

In the meantime however, to lower the barrier for simulations to experience and benefit from in situ, we present Inshimtu. Inshimtu allows for a low-barrier to entry with no changes needed to existing simulation codes. Simulation codes can try out pseudo-in situ easily, and see the benefits that a full in situ integration could provide. There are numerous flavors of in situ visualization, and Inshimtu falls into one of them, as categorized by an effort led by Childs et al [11] who created a system by which an in situ systems could be described by six different axes, each with various corresponding sub-categories. Inshimtu is:

- Integration Type: Application Aware (dedicated API)
- Proximity: Off Node, Same Computing Resource
- Access: Indirect
- Division of Execution: Space Division
- Operation Controls: Both - Automatic and Human-in-the-Loop
- Output Type: Varies based on Catalyst script

While Inshimtu isn't, and doesn't purport to be, a long term solution for a simulation codes in situ integration, it can provide a much needed entry point where users can learn about and see the benefits of in situ, and then move on to a full-fledged in situ tool after they have been convinced. This step is something that has been missing in the community up until this point, with the onus of moving to in situ being left to simulation developers, which is one reason why adoption has been so low up until this point.

3 Design and Implementation of Inshimtu

Inshimtu is so named because it is our vision that this code is an in situ shim that can be used to augment existing simulations with in situ capabilities. The goal of Inshimtu is to provide a stepping stone for users coming from a post hoc processing world to try in situ with no development required. A further benefit of Inshimtu to simulation developers is that they do not have to link their codes with any external dependencies in order to test Inshimtu, simplifying the process to try in situ. Inshimtu builds on existing technologies for data visualization and is not designed to replace a full instrumentation of a simulation code with one of the many existing in situ libraries, it is simply a low barrier way for codes that have never tried in situ and don't know what it can offer or where to start, to experience the benefits in situ provides. Simulations write their output to temporary files where Inshimtu reads, processes, visualizes, creates subsets, and can even delete the original output files, saving only what it interesting or useful. The temporary nature of these files means that we can write out higher spatial and/or temporal resolutions than the filesystem might otherwise support; while, the output of the visualization pipelines produce compressed simulation artifacts to save to persistent storage. This process is illustrated in Fig. 1. Of course, the use of Inshimtu comes with a small cost. Inshimtu is not real in-situ, the traditional advantages of in-situ, like speeding up simulation execution time by avoiding costly I/O, can obviously not be exploited. Below, we first describe the existing technologies used in Inshimtu, followed by its overall design and implementation.

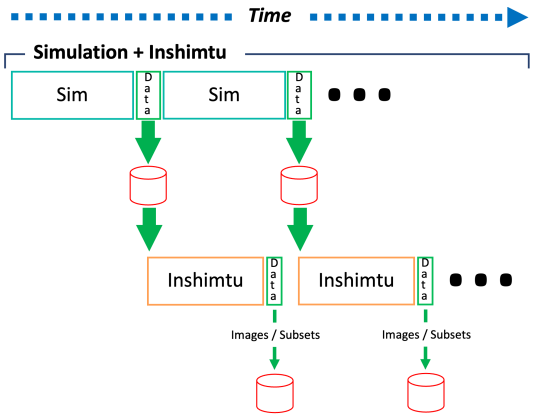

Fig. 1: A simulation plus Inshimtu pipeline over time. This pipeline denotes the simulation outputting data to disk where Inshimtu will then process that simulation output using a ParaView Catalyst pipeline. The output of this processing can be images, summary statistics, or even subsets of the simulation data.

3.1 Existing Technology

Inshimtu uses three primary existing technologies: 1. VTK 2. ParaView and 3. Catalyst.

One of the main design goals was an easy integration into existing visualization tools. We chose the open source ParaView and Catalyst for this task. We leverage the full workflow of these tools for designing in-situ pipelines.

VTK - VTK is used as the underlying data model. A number of different importers have been implemented in Inshimtu that directly make use of existing VTK readers. Using VTK means that the data can be directly passed into Catalyst for processing and visualization.

Catalyst - Catalyst is used for its ability to access the full power of ParaView. By using Catalyst scripts users are able to perform any visualization, subset, or statistics operator that they would be able to use in the full ParaView GUI. Catalyst1 is used in the current implementation of Inshimtu.

ParaView - The main function of ParaView for Inshimtu is to enable live interactive visualization. Inshimtu can connect to ParaView using Catalyst allowing users to interactively explore their data. This is particularly useful for users when they are designing their Catalyst scripts and visualization operations for large runs.

3.2 Inshimtu Design

Inshimtu is composed of four primary components: 1. core 2. sentinels 3. utils and 4. processing.

core - This component comprises the basic Inshimtu infrastructure (MPI, coordinators, initial argument parsing).

sentinels - This component is responsible for setting up file system watchers to watch for new files to process, as well as watching for the *done* signal that comes at the end of a batch job.

utils - This is a helper component that consists of *help* and *logging* classes, allowing for easier debugging when developing new components.

processing - This component is the main workhorse within Inshimtu. It contains the Catalyst adapter, sets up pipelines, and contains the various importer classes. Currently Inshimtu uses Catalyst1. There are four current implemented importers: 1. RawNetCDF 2. XMLImage 3. XMLPImage and 4. XMLRectilinear. Implementing new importers for different simulation codes is straightforward when making use of existing VTK importers. Thus, Inshimtu can easily be extended to accommodate reading new simulation output.

3.3 Using Inshimtu

Enabling Inshimtu amounts to adding a line to the batch file before running a simulation code. No changes to the simulation code are required. The simulation code is not aware of Inshimtu, and no coupling or synchronization exists between the simulation and Inshimtu. Once the simulation is actively running and saving its outputs to disk, Inshimtu takes those outputs from disk, processes them with a Catalyst script (users can even connect live with ParaView if desired), saves the outputs (images, extracts, summary statistics, etc.), and then watches and waits for new outputs to appear. Once the simulation terminates, the batch script updates a *"*.done"* file which Inshimtu watches in order to know when all processing is complete. One important feature of Inshimtu is that it can be enabled to remove simulation output files after it processes them, thus only keeping its outputs on disk, drastically saving storage space during a long simulation.

There are two ways to launch and configure Inshimtu. First, Inshimtu can parse a JSON file with the necessary arguments (see Fig. 2). This JSON file (see Fig. 3) can either pass a link to a catalyst script, or the script can be embedded into the JSON file. In addition to specific catalyst commands, this JSON file can be used to specify specific pre- or post-processing commands that are run before or after the catalyst functions. The benefit of this is that data can be decompressed by Inshimtu before use if needed, moved, modified, then processed by Catalyst, and then the original data and the output from Catalyst can be manipulated further if needed. Additionally, this script can tell Inshimtu to only process specific variables, which directory to watch for new simulation files, as well as pass control information to tell Inshimtu when to start and stop. Second, all of the required arguments can be specified on the command line (see Fig. 4).

```
1 srun ./Inshimtu -c ../testing/configs/png_watchDir_QVAPOR.json
```

Fig. 2: Launching Inshimtu using the JSON configuration file detailed in Fig. 3.

We believe that the simple interface for using Inshimtu, coupled with the use of well established existing visualization technologies, creates a convenient way to promote in situ to the simulation community. It is our goal that this code be used to educate simulation developers and users to the value of in situ, and guide them in creating true in situ integrations in their codes.

4 Inshimtu Use Cases

In this section we describe two different visualization case studies that demonstrate the functionality of Inshimtu as described in Section 3. The first case study

```
{ "input": {
    "watch": {
      "directory_path": "testing",
      "files_regex": "\\w*(.pvti)"
    }
  },
  "pipeline": {
    "scripts": [
      "../testing/pipelines/pngQVAPOR.py"
    ],
    "variables": [
      "QVAPOR"
    ]
  },
  "control": {
    "done_watchfile": "testing.done",
    "initial_connection_wait_secs": 0,
    "catalyst_inporter_nodes": [
      "0"
    ],
    "delete_processed_input_files": false }}
```

Fig. 3: Inshimtu JSON code example that watches a directory for *.pvti files and then processes them with a Catalyst Python script.

```
srun ./Inshimtu -w testing
                -d testing.done
                -s ../testing/pipelines/pngQVAPOR.py
                -f ''\w*(.pvti)''
                -v QVAPOR
```

Fig. 4: Launching Inshimtu using explicit command line arguments to match the pipeline described in Fig. 3.

in Section 4.1 uses Inshimtu to perform basic post processing tasks, demonstrating the Catalyst pipelines and basic Inshimtu functionality. The second case study in Section 4.2 uses WRF, the Weather Research and Forecasting Model [32] run on the Shaheen II supercomputer. Finally, we discuss the results and instrumentation process.

Inshimtu Release The following two subsections make use of Inshimtu and reference scripts and examples available in our GitHub repository: `https://github.com/kaust-vislab/Inshimtu-basic`.

4.1 Post processing with Inshimtu

Inshimtu is designed to be run during an active simulation, but since it is an in situ shim, and designed to work from files, it can easily be used to post process data as well. In fact, this is the perfect way to test catalyst scripts, different Inshimtu settings, and easily process existing simulation output. An example of how to do this is available in our repository, and uses the script shown in Fig. 5

To setup this example, first build Inshimtu according to the *README.md* instructions. Once Inshimtu is built *cd* into the build directory, in order to prepare to run the *png_enumerated_QVAPOR.json* example. This example will process all existing files specified in the example script, and exit when all files have been processed. It uses the Catalyst script *pngQVAPOR.py* to transfer data to ParaView, if watching live using ParaView, and creates a **.png* image as output. To run this example use the following steps:

1. Create a directory called *testing* where data files that are to be processed by Inshimtu will be copied

```
{ "input": {
    "initial_files": [
      "testing/dataoutfile_QVAPOR_00.pvti",
      "testing/dataoutfile_QVAPOR_01.pvti",
      "testing/dataoutfile_QVAPOR_02.pvti",
      "testing/dataoutfile_QVAPOR_03.pvti",
      "testing/dataoutfile_QVAPOR_04.pvti",
      "testing/dataoutfile_QVAPOR_05.pvti",
      "testing/dataoutfile_QVAPOR_06.pvti",
      "testing/dataoutfile_QVAPOR_07.pvti",
      "testing/dataoutfile_QVAPOR_08.pvti",
      "testing/dataoutfile_QVAPOR_09.pvti",
      "testing/dataoutfile_QVAPOR_10.pvti",
      "testing/dataoutfile_QVAPOR_11.pvti",
      "testing/dataoutfile_QVAPOR_12.pvti",
      "testing/dataoutfile_QVAPOR_13.pvti",
      "testing/dataoutfile_QVAPOR_14.pvti"
    ]
  },
  "pipeline": {
    "scripts": [
      "../testing/pipelines/pngQVAPOR.py"
    ],
    "variables": [
      "QVAPOR"
    ]
  },
  "control": {
    "initial_connection_wait_secs": 10,
    "catalyst_inporter_nodes": [
      "0"
    ]
  }
}
```

Fig. 5: Inshimtu JSON code example that processes an enumerated list of existing data files with a Catalyst Python script.

2. Copy the example data files to this directory: *cp -r ../testing/data/wrf/pvti/* testing*
3. To view the data live in ParaView enable the Catalyst connection in ParaView
   - Select *Catalyst / Connect...* from the ParaView menu
   - Click *OK* in Catalyst Server Port dialog to accept connections from Inshimtu
   - Click *OK* in Ready for Catalyst Connections dialog
   - Select *Catalyst / Pause Simulation* from ParaView menu
4. Run Inshimtu:
   ./Inshimtu -c ../testing/configs/png_enumerated_QVAPOR.json
5. View the results in ParaView as well as the images saved to disk

There are several more examples in the *testing/configs* directory that you can try, each one being a derivation of one of Inshimtu's features, or using a different data type. View these files to explore other features of Inshimtu.

4.2 WRF + Inshimtu

For this use case we used the WRF simulation code run on Shaheen II at KAUST. WRF was chosen as an example code as there is a large user community at KAUST and it can produce vast quantities of data when looking at fine temporal resolutions. We ran wrf configured to simulate an area approximately 50 KM in a square centered around Jeddah Saudi Arabia and to create an extreme rain event. The premise behind this visualization is that the scientists were interested in creating a movie of various variables each time the simulation was run, with the catch being, that the phenomena of interest only occurred when simulating at a fine temporal resolution; a great use case for in situ.

Inshimtu was run on a separate compute cluster called Ibex, which is primarily dedicated to running various analysis workflows. Ibex and Shaheen have access to a common filesystem, so Inshimtu was able to read the files as they were produced by WRF. This visualization used a very similar setup to that of the *png_enumerated_QVAPOR.json* example and the *pngQVAPOR.py* catalyst

script. We just added more complex interactions and more variables. As we were using Catalyst we could also interactively explore the data live in ParaView by having the Ibex Catalyst script connect back to our local desktop. Fig. 6 gives an example visualization of the area of interest around Jeddah showing four primary variables.

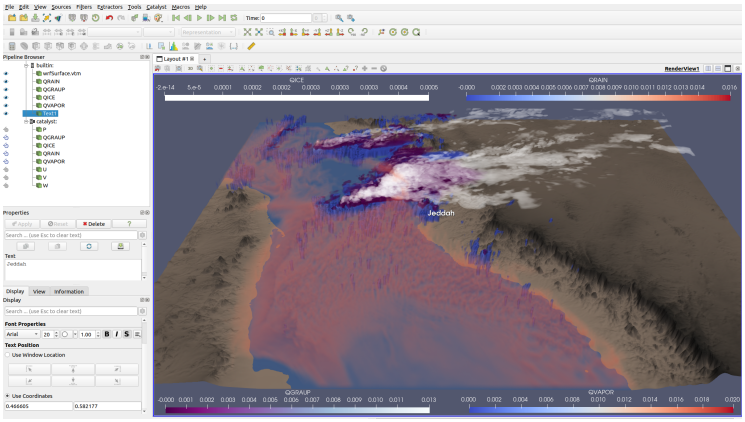

Fig. 6: Example visualization from Catalyst using Inshimtu of one of timesteps in the wrf simulation.

These two use cases are designed to show the functionality of Inshimtu, and the usefulness of being able to enable an in situ shim for simulations that do not yet have built-in in situ capabilities. While Inshimtu will not save time or cost of running a simulation, it allows for repeatable scriptable visualizations that can be interactive and demonstrate the function and power of in situ to a wide audience.

5 Conclusions and Future Work

The visualization community is investing heavily into creating and expanding various in situ visualization libraries and frameworks. These efforts are becoming well established and are ready for integration into simulation codes. However, community adoption of in situ is low, and it is difficult to quickly demonstrate the benefits of in situ for each and every simulation code. Therefore, we presented Inshimtu as a stepping stone towards full in situ visualization workflows. Inshimtu is a low barrier to entry in situ "shim" that works without modifying existing simulation codes, allowing for users to try before they buy into a full in situ integration. We also demonstrated Inshimtu usage on desktop machines as well as a use case on the Shaheen II supercomputer.

In the future, we would like to extend Inshimtu to work with more file types and demonstrate the benefits of using an in situ shim approach vs. a traditional post hoc approach at larger scale. We would also like to update the code base to use the newest iteration of Catalyst. We are also continuing work on integrating a data streaming library into a second version of Inshimtu that will allow simulation codes to bypass the filesystem, which is a current limitation. Finally, we hope to demonstrate the benefits of in situ to other codes run at KAUST and around the world using Inshimtu, helping to promote broader in situ adoption.

Acknowledgments This work was supported by King Abdullah University of Science and Technology (KAUST). This research made use of the resources of the Visualization and Supercomputing Laboratories at KAUST.

References

1. Freprocessing. `https://www.github.com/tfogal/freeprocessing`. Accessed: 2016-11-27.
2. Vignesh Adhinarayanan, Wu-chun Feng, David Rogers, James Ahrens, and Scott Pakin. Characterizing and modeling power and energy for extreme-scale in-situ visualization. In *IEEE Parallel and Distributed Processing Symposium (IPDPS)*, pages 978–987, 2017.
3. Sean Ahern et al. Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization, July 2011.
4. James Ahrens, Berk Geveci, and Charles Law. ParaView: An End-User Tool for Large-Data Visualization. In *The Visualization Handbook*, pages 717 – 731. 2005.
5. U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel. The SENSEI Generic In Situ Interface. In *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 40–44, Nov 2016.
6. Utkarsh Ayachit et al. ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 25–29, 2015.
7. Utkarsh Ayachit et al. Performance Analysis, Design Considerations, and Applications of Extreme-scale *In Situ* Infrastructures. In *ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*, November 2016.
8. Andrew C. Bauer et al. *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.
9. E Wes Bethel, Burlen Loring, Utkarsh Ayachit, David Camp, Earl PN Duque, Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress, Patrick O'Leary, et al. The sensei generic in situ interface: Tool and processing portability at scale. In *In Situ Visualization for Computational Science*, pages 281–306. Springer, 2022.
10. E Wes Bethel, Burlen Loring, Utkarsh Ayachit, Earl PN Duque, Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress, Patrick O'Leary, Dave Pugmire, et al. Proximity portability and in transit, m-to-n data partitioning and movement in sensei. In *In Situ Visualization for Computational Science*, pages 439–460. Springer, 2022.

11. Hank Childs, Sean D. Ahern, James Ahrens, Andrew C. Bauer, Janine Bennett, E. Wes Bethel, Peer-Timo Bremer, Eric Brugger, Joseph Cottam, Matthieu Dorier, Soumya Dutta, Jean M. Favre, Thomas Fogal, Steffen Frey, Christoph Garth, Berk Geveci, William F. Godoy, Charles D. Hansen, Cyrus Harrison, Bernd Hentschel, Joseph Insley, Chris R. Johnson, Scott Klasky, Aaron Knoll, James Kress, Matthew Larsen, Jay Lofstead, Kwan-Liu Ma, Preeti Malakar, Jeremy Meredith, Kenneth Moreland, Paul Navrátil, Patrick O'Leary, Manish Parashar, Valerio Pascucci, John Patchett, Tom Peterka, Steve Petruzza, Norbert Podhorszki, David Pugmire, Michel Rasquin, Silvio Rizzi, David H. Rogers, Sudhanshu Sane, Franz Sauer, Robert Sisneros, Han-Wei Shen, Will Usher, Rhonda Vickery, Venkatram Vishwanath, Ingo Wald, Ruonan Wang, Gunther H. Weber, Brad Whitlock, Matthew Wolf, Hongfeng Yu, and Sean B. Ziegeler. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 34(6):676–691, 2020.
12. Hank Childs et al. A contract-based system for large data visualization. In *Proceedings of IEEE Visualization 2005*, pages 190–198, 2005.
13. Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Comput. Graph. Appl.*, 30(3):22–31, May 2010.
14. Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Visualization at extreme scale concurrency. In *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, Boca Raton, FL, 2012.
15. Jai Dayal et al. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID)*, 2014.
16. Ciprian Docan et al. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.
17. M. Dorier et al. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 67–75, Oct 2013.
18. Aniello Esposito and Glendon Holst. In situ visualization of wrf data using universal data junction. In Heike Jagode, Hartwig Anzt, Hatem Ltaief, and Piotr Luszczek, editors, *High Performance Computing*, pages 475–483, Cham, 2021. Springer International Publishing.
19. Thomas Fogal, Fabian Proch, Alexander Schiewe, Olaf Hasemann, Andreas Kempf, and Jens Krüger. Freeprocessing: Transparent in situ visualization via data interception. In *Eurographics Symposium on Parallel Graphics and Visualization: EG PGV:[proceedings]/sponsored by Eurographics Association in cooperation with ACM SIGGRAPH. Eurographics Symposium on Parallel Graphics and Visualization*, volume 2014, page 49. NIH Public Access, 2014.
20. William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.
21. James Kress et al. Loosely coupled in situ visualization: A perspective on why it's here to stay. In *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 1–6, November 2015.

22. James Kress, Matthew Larsen, Jong Choi, Mark Kim, Matthew Wolf, Norbert Podhorszki, Scott Klasky, Hank Childs, and David Pugmire. Comparing the efficiency of in situ visualization paradigms at scale. In *High Performance Computing: 34th International Conference, ISC High Performance 2019, Frankfurt/Main, Germany, June 16–20, 2019, Proceedings 34*, pages 99–117. Springer, 2019.
23. James Kress, Matthew Larsen, Jong Choi, Mark Kim, Matthew Wolf, Norbert Podhorszki, Scott Klasky, Hank Childs, and David Pugmire. Opportunities for cost savings with in-transit visualization. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35*, pages 146–165. Springer, 2020.
24. James Michael Kress. In-line vs. in-transit in situ: Which technique to use at scale?, 2020.
25. Matthew Larsen et al. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Workshop on In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 42–46, 2017.
26. Kenneth Moreland, Ron Oldfield, Pat Marion, Sebastien Jourdain, Norbert Podhorszki, Venkatram Vishwanath, Nathan Fabian, Ciprian Docan, Manish Parashar, Mark Hereld, et al. Examples of in transit visualization. In *Proceedings of the 2nd international workshop on Petascale data analytics: challenges and opportunities*, pages 1–6. ACM, 2011.
27. David Pugmire, James Kress, Jieyang Chen, Hank Childs, Jong Choi, Dmitry Ganyushin, Berk Geveci, Mark Kim, Scott Klasky, Xin Liang, et al. Visualization as a service for scientific data. In *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI: 17th Smoky Mountains Computational Sciences and Engineering Conference, SMC 2020, Oak Ridge, TN, USA, August 26-28, 2020, Revised Selected Papers 17*, pages 157–174. Springer, 2020.
28. David Pugmire, James Kress, Jong Choi, Scott Klasky, Tahsin Kurc, Randy Michael Churchill, Matthew Wolf, Greg Eisenhower, Hank Childs, Kesheng Wu, et al. Visualization and analysis for near-real-time decision making in distributed workflows. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1007–1013. IEEE, 2016.
29. David Pugmire, Caitlin Ross, Nicholas Thompson, James Kress, Chuck Atkins, Scott Klasky, and Berk Geveci. Fides: a general purpose data model library for streaming data. In *High Performance Computing: ISC High Performance Digital 2021 International Workshops, Frankfurt am Main, Germany, June 24–July 2, 2021, Revised Selected Papers 36*, pages 495–507. Springer, 2021.
30. Marzia Rivi, Luigi Calori, Giuseppa Muscianisi, and Vladimir Slavnic. In-situ visualization: State-of-the-art and some use cases. *PRACE White Paper; PRACE: Brussels, Belgium*, 2012.
31. Brad Whitlock, Jean Favre, and Jeremy Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *In Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 101–109, 2011.
32. WRF – The Weather Research and Forecasting Model, v. 3.7.1, 2015.
33. Fan Zhang, Tong Jin, Qian Sun, Melissa Romanus, Hoang Bui, Scott Klasky, and Manish Parashar. In-memory staging and data-centric task placement for coupled scientific simulation workflows. *Concurrency and Computation: Practice and Experience*, 29(12), 2017.