Bridging Gaps in Simulation Analysis through a General Purpose, Bidirectional Steering Interface with Ascent

Andres Sewell[*†], Dimitrios K. Fytanidis[†], Victor A. Mateevitsi[†△], Cyrus Harrison[‡], Nicole Marsaglia[‡], Thomas Marrinan[‖†], Silvio Rizzi[†], Joseph A. Insley[†§], Michael E. Papka[†△], Steve Petruzza[*]
[*]*Utah State University,* [†]*Argonne National Laboratory,* [‡]*Lawrence Livermore National Laboratory,*
[△]*University of Illinois Chicago,* [‖]*University of St. Thomas,* [§]*Northern Illinois University*

***Abstract*—This paper presents a novel, general-purpose interface for adding interactive human-in-the-loop steering controls to existing simulation codes. The design is agnostic to any specific in situ analysis and visualization library, though our reference implementation is based on Ascent – a common in situ visualization and analysis library for large-scale simulations. Traditional in situ analysis and visualization workflows are typically automated through trigger mechanisms that execute as simulations reach certain predefined states (e.g. every *N* timesteps, simulation parameters become unstable, etc.). Although such automated in situ tasks suffice for many real-world applications, we demonstrate that a complementary interactive interface can significantly boost scientific productivity. We show through two use cases how our approach enables scientists to pause simulations and make interactive adjustments between timesteps. This method eliminates cold restart overhead, accelerating setup, troubleshooting, and exploration.**

***Index Terms*—Human-in-the-loop, in situ, interactive steering, Ascent, simulation.**

I. Introduction

Scientific simulations serve as crucial instruments for modeling real-world phenomena in a controlled and reproducible manner, enabling scientists to answer questions that might be impractical or prohibitively expensive to investigate through direct physical experiments [21]. As high-performance computing (HPC) systems evolve, scientific simulations grow in scale and complexity. Exascale computing [15] has enabled more detailed, dynamic, and high-resolution simulations, unlocking new scientific discoveries. These advancements have increased data sizes, causing I/O and storage challenges [9]. Issues like I/O bottlenecks highlight the need for modern in situ analysis and visualization techniques. These methods provide scientists with a standardized and efficient way to extract intermediate results from their simulations without requiring the output of raw simulation data [5].

Computational steering, a form of in situ processing, allows scientists to adjust simulation parameters on the fly, enhancing their productivity by reducing the delay between user intervention and the observation of outcomes [20]. Intervention includes changing parameters, adjusting mesh resolution, or restoring a checkpoint. In situ techniques, including computational steering, play a vital role in enabling scientists to discover answers to research questions more swiftly and efficiently.

Historically, implementing interactive computational steering has necessitated the development of custom steering infrastructure on a per-simulation basis, or employing specialized frameworks such as SCIRun [16]. However, the emergence of general-purpose in situ libraries, characterized by "time division and on node proximity" [6], has facilitated the integration of powerful analysis capabilities into existing simulation codes, minimizing the reliance on bespoke solutions. Despite these advancements, most in situ steering relies on triggers which automatically execute based on the simulation state. We feel that there remains a significant gap in facilitating human-in-the-loop simulation interactivity and steering for situations where domain knowledge is critical for determining appropriate intervention strategies.

To our knowledge, there are currently no in situ libraries that provide a comprehensive, general-purpose steering interface to interactively control existing simulation codes. This paper marks our attempt to meet this requirement, presenting:

1) An implementation of our interactive bidirectional steering mechanism based on the Ascent [11] in situ framework.
2) The creation of terminal-based and Jupyter notebook-based steering interfaces.
3) Two fully implemented use cases of interactive steering, each based on a different simulation, neither of which natively supports steering without our mechanism.

All the features described in this paper are also currently integrated in Ascent and readily available to use.

II. Background

The rise in popularity of in situ methods has led to the development of specialized software libraries designed for "time division and on node proximity" [6] to make these capabilities readily accessible to simulation developers. Noteworthy libraries, including VisIt/Libsim [4], ParaView/Catalyst [2], SENSEI [3], and Ascent [11] [1], aim to alleviate the complexities of building efficient, heterogeneous, and vendor-agnostic in situ infrastructures.

Libsim and Catalyst are Application Programming Interfaces (APIs) designed to interface with VisIt and ParaView respectively, each allowing users to define and execute in situ analysis and visualization tasks via scripts. Both APIs can also interface with their respective Graphical User Interfaces (GUIs) for interactive analysis, allowing users to configure render pipelines manually in a GUI and then export them as scripts to use for subsequent simulation runs. Catalyst supports bidirectional data communication, crucial for interactive steering.

SENSEI is an intermediary between simulations and various in situ endpoints, enabling different in situ back-ends to be swapped in at runtime. When Catalyst is used as an endpoint, SENSEI can facilitate bidirectional data communication through the Catalyst API. By contrast, Ascent is standalone and primarily relies on YAML or JSON files for task definition, with the option to provide it with external Python scripts for added flexibility. Although Ascent does not directly integrate with GUI applications like VisIt or ParaView, it provides a Python interface which can be used for data analysis within a Jupyter Notebook environment.

Bidirectional Steering. The distinction between computational steering and bidirectional steering is subtle. Computational steering occurs when the simulation state changes as a result of external intervention, whether it originates directly from a user or as the result of a trigger. Bidirectional steering is computational steering informed by in situ analysis and visualization. In other words, it is the feedback loop through which a user performs in situ tasks on simulation data and makes steering decisions. In this scenario, each communication direction (the user and the simulation, with a steering interface acting as a bridge) influences the other component.

Libsim, Catalyst, and, by extension, SENSEI facilitate bidirectional communication limited to passing simulation data around. This allows for limited simulation steering via their respective GUIs (VisIt for Libsim, ParaView for Catalyst) [18]. Catalyst stands out for its simplicity in integrating interactive bidirectional steering into existing simulation codes. Until recently, Catalyst was bound to the Paraview user interface, but Catalyst 2 [13] now provides an API which external applications could integrate and use, such as ADIOS 2 [14]. In principle, this new interface could be used to perform bidirectional interactions, however it requires all components of the workflow to interface with the given API. Our proposed mechanism instead, is more flexible and allows to register, list and invoke callbacks that are part of the simulation, in situ framework or third party applications, as well as executing system commands (e.g., Linux utilities) and scripts as part of the in situ workflow.

A. Use Cases For Bidirectional Steering

To demonstrate the capabilities of our bidirectional steering mechanism we focus on some real-world scenarios, such as: (i) correcting unproductive simulations that might otherwise require a complete restart due to issues like modeling errors, software bugs, or resource limitations; (ii) streamlining the process of finding optimal initial conditions; (iii) investigating intriguing simulation states to explore phenomena of interest.

III. Implementation

This section presents an overview of the existing features of Ascent that motivated its selection for our reference implementation. We then discuss our design/implementation decisions.

A. Existing Ascent Capabilities

The Ascent framework supports various data models and extraction methods, and features a comprehensive actions API, a sophisticated trigger system, and an interactive data analysis through Jupyter Notebook integration [10]. Actions within Ascent, which include rendering specifications, data transformations, camera settings, plots, and other configurations, are determined before the simulation runs and can be defined in the code or specified in external YAML files. Notably, before this work, this could only be used to analyze the current timestep data, and not influence the simulation.

A noteworthy aspect of Ascent is its query and trigger mechanism, enabling the execution of specific in situ actions based on the simulation's state. This functionality can range from simple triggers, such as actions occurring every N timesteps, to more complex conditions, like triggering visualizations when data entropy exceeds a certain level. Additionally, Ascent's data extraction features allow for the output of timestep data in various forms: saving to disk as HDF5 files, processing via external Python scripts, forwarding to a visualization cluster with ADIOS2, or displaying in real-time in a Jupyter Notebook instance [8]. The maturity of the framework and the features described above, make Ascent the perfect framework to extend for our goals.

B. Bidirectional Interface Integration

The integration of Ascent into a new simulation codebase is generally straightforward. It involves initializing Ascent, setting how often it receives data, defining in situ tasks, and finalizing Ascent when the simulation ends. Data is passed by reference using zero-copy methods provided by Conduit. To enable bidirectional steering capabilities, we wanted to provide the ability for users to register and invoke arbitrary functions or scripts through a steering interface. Hence, our implementation adds support for application callback registration and extended the trigger system in Ascent to use those.

Simulation/Application Callbacks. Since developers and users know best what interactivity their simulations need, we delegate the implementation of specific details to them, avoiding the inclusion of specific steering functionalities within Ascent. This consideration is particularly relevant for simulations that already possess internal functions capable of performing desired actions, but are normally inaccessible to the in situ library. Aiming for a versatile solution, we devised a system where developers can register functions with Ascent.

Our interface provides the ability to invoke functions on the simulation/application side. Such functions, may either wrap existing simulation capabilities, or be written from scratch to provide custom functionality. These functions use void or

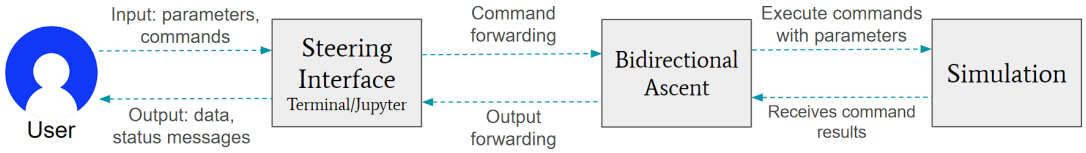


Fig. 1. Diagram illustrating the user's input of commands and parameters via the steering interface, which Ascent processes and executes. The subsequent output is returned to the user. Based on the in situ analysis, the user makes steering decisions that change the simulation state. This can be repeated as many times as desired.

boolean callback signatures, defining how Ascent registers and invokes them. They can either wrap existing simulation capabilities or be written from scratch. Once these functions are defined, they must be registered with Ascent. Registering function callbacks involves providing Ascent with a function reference and a user-friendly name to associate with the command.

Ascent has been modified to include a mapping of callback names to function pointers. We also introduced static methods for registering these callbacks with Ascent and the necessary infrastructure to invoke these callbacks by name. Two simulation callback signature types are supported: one accepts two Conduit Node references for input and output parameters, returning void, while the other accepts no arguments and returns a boolean. Conduit Nodes allow a broad range of parameters to be passed without requiring Ascent to understand their semantics. The callback system is designed to abstract MPI communication details, ensuring that end-users do not need to manually handle MPI communicators when invoking steering commands. However, developers can still access the MPI communicator, providing greater flexibility for those needing custom MPI logic within their callbacks.

Other in situ libraries could theoretically allow the registration of the same callbacks based on the function signatures defined here, providing a flexible mechanism for callback integration. Adopting a mechanism for callback registration and integration similar to what we implemented would enable compatibility across different in situ frameworks.

```
action: "add_triggers"
triggers:
  t1:
    params:
      callback: "isStable"
      actions_file :"stable_actions.yml"
  t2:
    params:
      callback: "isUnstable"
      actions_file :"unstable_actions.yml"
```

Listing 1. Ascent actions which combine the existing trigger infrastructure with custom callbacks to create more complex control flow.

Trigger mechanism extension. To implement our interactive steering interface, a trigger mechanism is set up to signal Ascent when to enter steering mode, pausing the simulation and allowing for real-time user interaction with the assurance that the simulation state won't continue changing. This setup enables users to engage with the simulation through an interactive steering interface, offering direct control over in situ tasks and the ability to issue commands.

Leveraging Ascent's trigger infrastructure to utilize simulation callbacks as conditions for initiating custom in situ actions allows users to craft nuanced control flows. An example is in Listing 1, in which two triggers are defined using two separate callbacks as conditions. Ascent will perform different sets of actions depending on whether the simulation is stable. Additionally, for scenarios where users wish to modify the simulation interactively, they can develop and register complex void callbacks that process input parameters and produce outputs. This method of callback support empowers simulation developers to design an interaction API best suited for their codebase, embodying the flexibility central to our general-purpose approach. For example, this allows to simply use external tools into steering workflows. One possible application for this feature is to set up external notifications about the simulation's state. For instance, if a trigger condition identifies a noteworthy event, it could be configured to automatically send an email notification to the user

C. Interactive User Steering Interfaces

We have developed two user steering interfaces: a terminal-based interface and a Jupyter-notebook interface. Once the simulation triggers the designated condition and enters a paused state, the user is prompted to issue commands using either interface. While the simulation is paused, all computational resources are held in place, ready for the simulation to resume without the need for reallocation or reinitialization. This interface-agnostic approach gives users the flexibility to choose between the straightforwardness of a terminal, the typical experience offered by a Jupyter-notebook, or to develop something entirely custom to their needs.

Our **terminal-based steering interface** is dependency-free, using an event loop to parse user inputs into executable commands. A list feature displays the simulation callbacks registered with Ascent, informing users of available simulation commands. An internal Conduit node facilitates argument passing for execution with simulation commands, manipulated via a *param* command for assigning values to parameters. The interface ensures that commands are executed with the current parameter set, while parameter resets are also possible. To conclude steering and resume simulation, users can use an exit command.

The **Jupyter-notebook steering interface** requires users to establish an SSH tunnel to the Ascent instance's compute node. Upon accessing the notebook, users should select the "Ascent-Jupyter-Bridge" kernel, enabling Ascent to establish a live connection with the simulation. The initial step involves executing the "%connect" command, which bridges the user to the simulation. At this point, Ascent functions like a remote server processing Python code. While standard Jupyter-

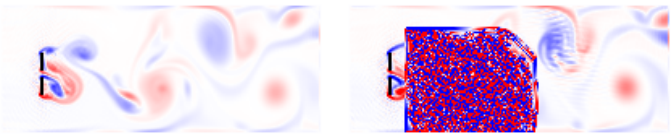


Fig. 2. LBM-CFD simulation without (left) and with (right) numerical instability.

notebook features remain accessible (subject to certain constraints), our enhancement allows for the direct invocation of simulation callbacks registered with Ascent. After concluding interactions within the interface, the "%disconnect" command relinquishes control, allowing the simulation to proceed with its execution. The Jupyter-notebook environment also provides access to Ascent's Python library and users' Python environment allowing for more advanced interactive data analysis.

IV. Use cases and evaluation

We implemented two use cases to showcase some of the potential in our general-purpose steering interface. Both simulations did not natively provide steering controls beforehand.

A. Rescue use case: Lattice Boltzmann CFD

For this demonstration, we utilized a 2D Lattice Boltzmann CFD (LBM-CFD) simulation [7] to address the rescue use case. This evaluation was performed using 20 nodes on the Polaris supercomputer at Argonne National Laboratory [17]. This use case is important to show the ease of use and the flexibility of the proposed mechanism.

Numerical instability, a common challenge in various simulations, can manifest in LBM-CFD simulations under certain conditions. Instability occurs when the delta time for each simulated timestep is too large, causing particles in the fluid to move more than one grid cell per step, leading to cumulative errors and eventual instability. Stability is maintained by keeping density values above zero. Rescuing the simulation involves reverting to a stable checkpoint, increasing timesteps, and recalculating the state.

```
params = conduit.Node()
params["timesteps"] = 3200
output = conduit.Node()
execute_callback("setTimeSteps", params, output)
```

Listing 2. Users can manually input arguments for executing callbacks, facilitating a comprehensive command structure.

This simulation includes a built-in function for stability checking, which we encapsulated within a boolean callback, registered this callback with Ascent, and defined corresponding actions. After each timestep, Ascent checks this callback; if stable, Ascent proceeds with actions that render the grid data. If instability is detected (see Figure2), Ascent triggers actions designated for unstable states, including sending an email (i.e., using the system *mail* command), initiating a Jupyter Notebook server, and awaiting user connection to the steering interface, with the email conveniently linking directly to the Jupyter server.

The user connects to the live simulation through Ascent's bridge kernel and interacts with timestep data and callbacks. To stabilize the simulation, the user invokes two void callbacks: revert to the last stable checkpoint and increase the number of timesteps (as detailed in Listing 2). The simulation then resumes with its state altered based on user input, confirmed by Ascent's return to stable visualizations.

This instance of successfully mitigating simulation instability through direct user intervention illustrates the potential and versatility of our proposed interface, suggesting broader applicability in solving real-world challenges in collaboration with simulation developers and users.

B. Streamlining the process of finding optimal parameters: Direct Forcing Immersed Boundary Method (DF-IBM) CFD

This use case is grounded in a CFD simulation of a real-world terrestrial vegetation canopy. Vegetation is instrumental in mitigating the urban heat island effect and enhancing outdoor comfort (e.g., [12]). However, challenges persist in modeling vegetation effects on momentum, turbulence, and heat/moisture fluxes. A key challenge is bridging the scale gap between centimeter-scale tree branches and meter-scale street modeling. The goal is to find a range of suitable resolutions such that the simulation performance is balanced with high quality results.

The direct-forcing immersed boundary method (DF-IBM), proposed by Yang et al. [22], was adopted to simulate real-world tree vegetation using sample Lidar data available from the US Geological Survey [19]. This method allows for including sub-grid resolution geometrical features of trees, such as branches and stems. The implementation, based on the scalable Nek5000 solver, is suitable for both Large Eddy Simulation (LES) and Reynolds-Averaged Navier-Stokes (RANS) simulations, utilizing Lidar data from real trees and vegetation canopies. The runs were performed using 50 nodes, 6,400 total cores, on Argonne's LCRC (Laboratory Computing Resource Center) Improv supercomputer.

It requires using a Gaussian kernel to force each point from the point cloud, which is then projected to the Gauss–Lobatto–Legendre (GLL) points. However, there is a trade-off between the number of points used to describe the tree geometry and the additional computational cost required for the DF-IBM method's forcing, especially for subgrid features. Numerical instability can occur across different simulations depending on the number of points, grid resolution, geometry orientation, and imposed flow velocity at the inlet.

In a typical workflow, before any production simulation run or parametric analysis occurs, optimizing the computational setup of a case involves extensive testing with varying particle numbers used to represent the geometry of the tree, selecting an optimal time step (dt) based on numerical stability and computational efficiency for each run. Therefore, there is a need to balance the resolution (number of points) used to describe the geometry of the tree with the accuracy of the resulting geometry (see Figure 3).

Using a non-interactive procedure, this process would require running several test simulations. While these simulations could potentially be run simultaneously, users typically choose

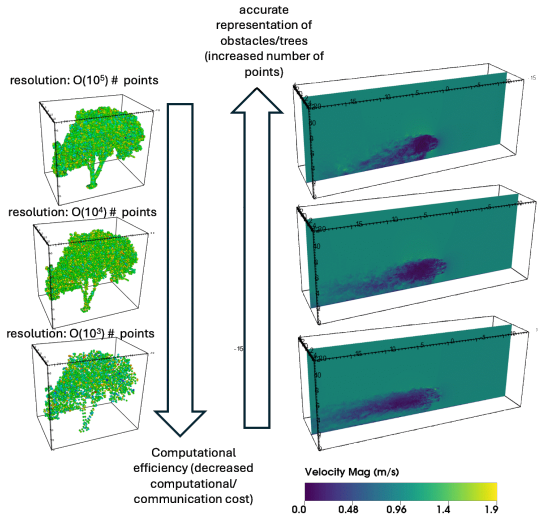
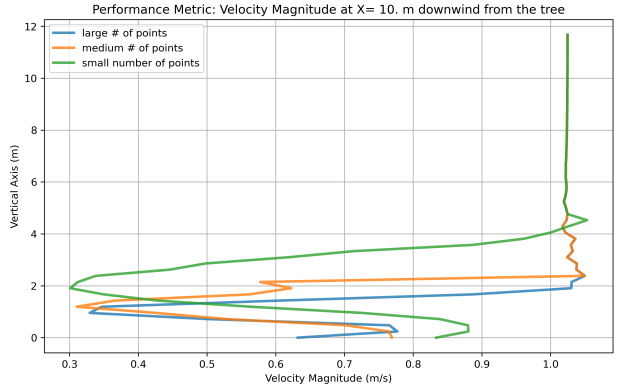


Fig. 3. Optimization of computational cost versus accurate representation of obstacles described using the Immersed Boundary Method (IBM). On the left: different resolution of points is shown. On the right: the corresponding results for the instantaneous velocity field from an LES simulation are shown.

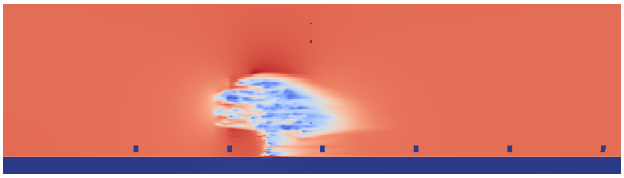
to perform them sequentially for convenience. For example, if a simulation is already producing unacceptably inaccurate results at the current resolution, the user would know to not further reduce the resolution. Additionally, while these runs could be scripted to execute automatically, the analysis and setup of the simulations still requires manual efforts (i.e., minutes or hours in complex cases). Finally, there is a time overhead due to domain partitioning that occurs at the beginning of each new simulation run.

As an alternative, our proposed general-purpose steering interface was used to streamline this process. Rather than cold-restarting the simulation for each resolution that we wanted to test, we instead devised two callbacks: one which swaps the current Lidar dataset with one of a higher or lower resolution, and another which forces NekIBM to load and use the newly swapped particle data. The simulation ran without restarting, adjusting case resolution every 6,000 timesteps via the steering interface. The average velocity at a downwind location on the leeward side of the tree was plotted in situ, reviewed by a scientist, and used as a metric to evaluate the accuracy of the reduced case compared to the fully resolved target case. In this use case the human-in-the-loop is essential as both quantitative as well as qualitative metrics are required to take a decision e.g., check the size of the wake behind the obstacles (not only velocity but also turbulent quantities like TKE, etc.). The thresholds for such qualitative analysis are not written in stone and require scientific judgment. To further decrease the computational cost, the simulations were not reinitialized when the point number was updated. Instead, the solution from a case with a higher number of points was used as the initial condition for the next simulation, saving time.

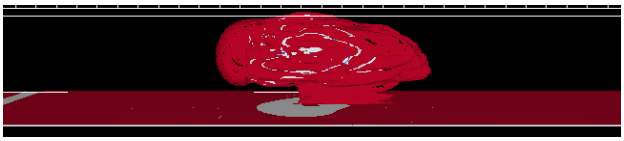
This procedure was repeated until the scientist found an optimal number of points, balancing reduced computational cost with the representativeness of the target case results. In addition to the metric plot, 3D visualization of the flow field was also performed in situ, in order to give the scientist an



(a)



(b)



(c)

Fig. 4. (a) Comparison of the performance between the high, medium and small number of points cases. The small case deviates significantly from the target case (large number of points). (b): 3D visualization of the velocity field (m/s) over a slice in the computational domain. (c): isocontour of the velocity field (in m/s).

opportunity to inspect the simulation results not only over a single line (where the velocity profile is plotted) but also everywhere in the domain (see Figure 4).

From a computational cost perspective, we observe that the proposed mechanism does not add any sensitive overhead to the runtime. On the contrary, it is saving the re-initialization cost (2.5 minutes over a runtime of 57.5 minutes) when interactively swapping the data at runtime versus running a new simulation from scratch each time the user decides to change the number of particles. A scientist working on this case might perform this operation 4-6 times to find the right resolution which makes the proposed mechanism very valuable to reduce overhead from simulation restart while maintaining an interactive experience.

V. Conclusion

We introduced a general purpose interactive bidirectional steering mechanism which allows for easy real-time, human-in-the-loop interventions. Our reference implementation extends Ascent's in situ capabilities by enabling interactive invocation of simulation callbacks and system commands through terminal or Jupyter notebook interfaces. Our use cases demonstrated the flexibility, ease of use, and benefits for human-in-the-loop interactive in situ analysis and visualization workflows.

ACKNOWLEDGMENTS

This work was supported by and used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy Office of Science User Facility supported under Contract DE-AC02- 06CH11357. This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative. This work was partly funded by NSF award 2221812. We gratefully acknowledge the computing resources provided on Improv, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. 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.

REFERENCES

[1] J. Ahrens, M. Arienti, U. Ayachit, J. Bennett, H. Carr, R. Binyahib, P.-T. Bremer, E. Brugger, R. Bujack, J. Chen, H. Childs, S. Dutta, A. Essiari, B. Geveci, C. Harrison, S. Hazarika, M. H. Fulp, P. Hristov, X. Huang, J. Insley, Y. Kawakami, C. Keilers, J. Kress, M. Larsen, D. Lipsa, M. Majumder, N. Marsaglia, V. Mateevitsi, V. Pascucci, J. Patchett, S. Patel, S. Petruzza, D. Pugmire, S. Rizzi, D. Rogers, O. Ruebel, J. Salinas, S. Sane, S. Shudler, A. Stewart, K. Tsai, T. Turton, W. Usher, Z. Wang, G. Weber, C. Wetterer-Nelson, J. Woodring, and A. Yenpure. The ecp alpine project: In situ and post hoc visualization infrastructure and analysis capabilities for exascale. *International Journal of High Performance Computing Applications*, June 2024.

[2] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '15, p. 25–29. Association for Computing Machinery, New York, NY, USA, 2015. doi: 10.1145/2828612.2828624

[3] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel. The sensei generic in situ interface. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pp. 40–44, 2016. doi: 10.1109/ISAV.2016.013

[4] J. M. B. Whitlock, J.M. Favre. Parallel in situ coupling of a simulation with a fully featured visualization system. In *Eurographics Symposium on Parallel Graphics and Visualization*, EGPGV '11, 2011.

[5] 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. In situ methods, infrastructures, and applications on high performance computing platforms. *Computer Graphics Forum*, 35(3):577–597, 2016. doi: 10.1111/cgf.12930

[6] 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. M. 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. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 34(6):676–691, 2020. doi: 10.1177/1094342020935991

[7] T. Franczak, A. Nkansah, T. Marrinan, and M. Papka. A path from serial execution to hybrid parallelization for learning HPC. In *Proceedings of the Workshop on Education for High-Performance Computing*, EduHPC '17, 2017.

[8] S. Ibrahim, T. Stitt, M. Larsen, and C. Harrison. Interactive in situ visualization and analysis using ascent and jupyter. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '19, p. 44–48. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3364228.3364232

[9] M. Isakov, E. d. Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy. Hpc i/o throughput bottleneck analysis with explainable local models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–13, 2020. doi: 10.1109/SC41405.2020.00037

[10] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, eds., *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87 – 90. IOS Press, 2016.

[11] M. Larsen, E. Brugger, H. Childs, and C. Harrison. Ascent: A Flyweight In Situ Library for Exascale Simulations. In *In Situ Visualization For Computational Science*, pp. 255 – 279. Mathematics and Visualization book series from Springer Publishing, Cham, Switzerland, May 2022.

[12] E. Massaro, R. Schifanella, M. Piccardo, L. Caporaso, H. Taubenböck, A. Cescatti, and G. Duveiller. Spatially-optimized urban greening for reduction of population exposure to land surface temperature extremes. *Nature Communications*, 14(1):2903, 2023.

[13] F. Mazen, L. Givord, and C. Gueunet. Catalyst-adios2: In transit analysis for numerical simulations using catalyst 2 api. In *International Conference on High Performance Computing*, pp. 269–276. Springer, 2023.

[14] F. Mazen, L. Gombert, L. Givord, and C. Gueunet. In situ in transit hybrid analysis with catalyst-adios2. *arXiv preprint arXiv:2406.18112*, 2024.

[15] P. Messina. The Exascale Computing Project. *Computing in Science & Engineering*, 19(3):63–67, May 2017. Conference Name: Computing in Science & Engineering. doi: 10.1109/MCSE.2017.57

[16] S. G. Parker, D. M. Weinstein, and C. R. Johnson. *The SCIRun Computational Steering Software System*, pp. 5–44. Birkhäuser Boston, Boston, MA, 1997. doi: 10.1007/978-1-4612-1986-6_1

[17] Polaris — Argonne Leadership Computing Facility — alcf.anl.gov. https://www.alcf.anl.gov/systems/polaris. [Accessed 09-03-2024].

[18] Steering with catalyst and sensei - sensei 4.1.0-97-g00b3a5f documentation, 2018.

[19] U. G. Survey. Usgs tree lidar data. https://www.usgs.gov/media/images/tree-lidar.

[20] R. van Liere, J. D. Mulder, and J. J. van Wijk. Computational steering. *Future Generation Computer Systems*, 12(5):441–450, 1997. HPCN96. doi: 10.1016/S0167-739X(96)00029-5

[21] E. Winsberg. Computer Simulations in Science. In E. N. Zalta and U. Nodelman, eds., *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 ed., 2022.

[22] Y. Yang and S. Balachandar. A scalable parallel algorithm for direct-forcing immersed boundary method for multiphase flow simulation on spectral elements. *The Journal of Supercomputing*, 77:2897–2927, 2021.