Automated In Situ Computational Steering Using Ascent's Capable Yes-No Machine

M. Lawson, C. Harrison, B. Brugger, A. Skinner, M. Larsen

August 30, 2021

ISAV 2021: In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization
st louis, MO, United States
November 15, 2021 through November 15, 2021

Automated In Situ Computational Steering Using Ascent's Capable Yes-No Machine

Margaret Lawson
mlawson@sadia.gov
Univ. of Illinois at Urbana-Champaign
USA

Cyrus Harrison
cyrush@llnl.gov
Lawrence Livermore Nat. Lab
USA

Eric Brugger
brugger1@llnl.gov
Lawrence Livermore Nat. Lab
USA

Aaron Skinner
skinner15@llnl.gov
Lawrence Livermore Nat. Lab
USA

Matthew Larsen
larsen30@llnl.gov
Lawrence Livermore Nat. Lab
USA

ABSTRACT

The life of a multi-physics code user is complicated. Simulation crashes, efficient resource utilization, and simulation parameter choices are time consuming workflow issues that increase a user's iteration time. Simulations often don't provide general tools to support automatically adapting workflows to the diverse set of problems that multi-physics codes are capable of simulating. In situ visualization and analysis infrastructures are designed to be general. They are repositories of shared capability that support multiple simulation codes. Normally, the connection between simulation and in situ analysis is unidirectional (e.g., render an image or querying the mesh). In this work, we close the loop between an in situ infrastructure and a simulation, and we explore opportunities to leverage in situ triggers for automatic computational steering at runtime. We demonstrate using Ascent's in situ trigger interface as a capable yes-no machine for controlling simulation choices.

KEYWORDS

scientific visualization, triggers, computational steering, in situ

ACM Reference Format:
Margaret Lawson, Cyrus Harrison, Eric Brugger, Aaron Skinner, and Matthew Larsen. 2021. Automated In Situ Computational Steering Using Ascent's Capable Yes-No Machine. In *ISAV'21 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV'21), November 12, 2021, St. Loius, MO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Multi-physics codes can simulate a diverse set of problems, but with these broad capabilities, users of multi-physics codes face a complex set of workflow problems. Both compute and human resources are often constrained, and maximizing both is paramount to conducting effective research in the simulation sciences.

As implied by the name, multi-physics codes can use multiple types of physics models (e.g., hydrodynamics, turbulence, and heat transfer) that affect the evolution of a problem. Each model has an associated computational cost that can vary by orders of magnitude. However, not all physics is needed all the time, and turning on physics only when needed can significantly reduce the overall time-to-solution. If the exact moment where the switch needs to occur is not known a priori, then the switch of the physics requires human interaction, which further constrains human resources.

Additional workflow problems arise when simulations stop converging or crash. Simulations can slow to a crawl when the time steps become tiny or even crash when cells in the mesh become inverted. Recovering from such occurrences almost always requires a human to identify the cause of the issue and provide a solution (e.g., trying a new problem or addressing issues with the computational mesh).

Simulation users address these scenarios with code-specific timers, code-specific heuristics, and human-in-the-loop interventions, often in the form of saving files for inspection with post-hoc visualization tools. Simplifying user workflows, by developing automatic solutions that help simulations adapt during runtime, can save both compute and human resources. Computational steering research focuses on solutions that adapt simulations at runtime, but prior research has focused largely on systems for interactive and even collaborative decision making. Interactive (human-in-the-loop) decision making will always be a key component of scientific modeling workflows. However, there are cases where interactive solutions cause tension between human resources and compute resources. Examples include: an allocation on a large supercomputer waiting idle for a user decision, or ensembles of thousands of simulations that users cannot reasonably direct interactively. Developing automatic solutions for a wider set of scenarios would simplify workflows, saving both compute and human time.

In many ways, challenges with automatically adapting multi-physics code workflows directly parallel those for in situ visualization and data analysis. Multi-physics codes can run a diverse set of problems, which is why complicated workflows exist, and similarly, in situ visualization and data analysis tools support a diverse set of simulations and analysis needs. To address these challenges, the visualization community has explored triggers to automate adaptive visualization tasks. A trigger is a statement that expresses the conditional: if X occurs then do Y. Previous trigger research has focused on both X and Y being some visualization or data analysis task (e.g., render an image or create an isosurface). In this work, we explore use cases where X is performed by the in situ visualization infrastructure and Y controls simulation parameters. By leveraging in situ triggers as part of a simulation's decision making process (for physics or algorithm choices), we can expand the scenarios where automation improves workflows. Even when human judgment is still needed, we can improve the workflow by using in situ tools to automatically capture relevant data and present useful context to the user for decision making.

In this paper, we show how in situ tools can be used for automatic computational steering. Our contribution is an approach for automating computational steering workflows by leveraging the general capabilities of Ascent, an in situ visualization tool. As part of this work, we enhanced Ascent's expressions language, used for triggers, and connect the language to a simulation, creating a yes-no interface that extends the simulation's ability to adapt at runtime, without human intervention. We demonstrate the effectiveness of our approach by examining use cases where automated decision-making can streamline simulation workflows.

2 RELATED WORK

2.1 Computational Steering

In recent years, a large number of research efforts have concentrated on creating an infrastructure to enable HPC scientists to dynamically steer their applications, a process typically referred to as "computational steering" [3, 5–10, 15, 18, 21–29, 31–33]. A good overview of these efforts can be found in [19, 20]. This work is largely focused on allowing scientists to interactively steer their applications through online visualization and computational steering controls [26]. This process is referred to more generally as putting the "human-in-the-loop" for HPC applications [19]. Computational steering has a number of use cases including allowing users to: explore different "what-if" scenarios [30, 33] or cause and effect relationships [20, 23]; visually identify complex patterns that are difficult to identify programmatically [19]; change a simulation's timestep [28] to help capture transient events or provide more accurate solutions during critical moments in the simulation; trigger remeshing or adjust boundary conditions via a GUI [5, 21]; explore intermediate simulation states to gain a better understanding of model behavior [20, 30]; stop an application that is no longer producing useful results or has experienced some sort of degeneration [30]; and improve performance through techniques such as load balancing [20]. Our work focuses on automating steering decisions, instead of providing infrastructure supporting "human-in-the-loop" decisions.

2.2 Triggers

Automating in situ visualization and analysis actions is a key in situ challenge and an active research area. A priori in situ visualization workflows must compete with post-hoc workflows that provide much more flexibility. One approach to close this gap is to use *triggers*, which allow users to describe conditions under which to dynamically adapt visualization actions (if X occurs then do Y). Triggers can be domain-specific or a general capability provided by in situ tools. [4] is a notable example of a domain-specific trigger used to execute expensive analysis only when needed in a combustion simulation. The Ascent [16] library provides a general trigger capability. [17] describes Ascent's trigger system and other foundational related work on triggers. Other recent work includes [34], which outlines an in situ workflow system based on a reactive domain-specific language. Triggers are also enabling other approaches to mitigate a priori constraints by using I/O more efficiently to save important subsets of data for post hoc exploration [14]. In this work, we extend Ascent's infrastructure, connecting its expression language to its trigger system. We also connect Ascent in a feedback loop with a simulation code. The trigger conditions are expressed using Ascent's actions API and are read by the simulation code as a yes/no result. This allows us to explore cases where the human can be taken out of the loop through use of in situ triggers.

3 OVERVIEW

This work uses Ascent [16] coupled with the MARBL [1] multi-physics code to demonstrate automatic steering of example simulation workflows. Ascent is a many-core capable flyweight in situ visualization and analysis infrastructure. In addition to scalable rendering and mesh transformations, Ascent provides an expression language and trigger system that allows users to describe conditions under which to adapt visualization actions. In this work, we close the loop between Ascent and the simulation in order to provide the simulation with direct access to the general analysis inside Ascent, and we demonstrate that, with this link, we can enhance a simulation codes ability to automatically adapt.

3.1 Trigger Expressions

The trigger system in Ascent has evolved over time. When first conceived, triggers were a series of filters with hard-coded functionality [17], which lacked composability. To address this limitation, we created a domain-specific language (DSL) [12] in Ascent that supports data aggregation and summarization methods (e.g., histograms, scalar reductions, and math expressions). The DSL allows users to compose functions to create complex expressions, and triggers are naturally supported as a comparison operator that results in *true* or *false*.

The expression language can either consume the mesh given by the simulation, or it can consume the results of transformations applied by Ascent pipelines. For example, a user could calculate an isosurface, find a per cell surface area, and use a sum aggregation to find the total surface area of the isosurface. Further, the expression results are cached as the simulation executes, which can be accessed through functions in the DSL. As part of this work, we added the capability to calculate history gradients of any sub-expression and select ranges of values from the expression cache. Building on the previous example, a user could monitor an isosurface's rate of expansion over time, using one of the built-in history gradient functions. Finally, Ascent provides access to the expression cache which enables an integrated simulation to take action based on what is calculated by Ascent.

3.2 Integration

MARBL is an arbitrary Lagrangian-Eulerian (ALE) multi-physics code. MARBL has an existing set of simple timers that allow users to trigger actions (e.g., checkpoints, visualization, or changes in physics) specified inside the simulation's Lua input interface. Timers can fire at a specific time (or cycle) or can fire at points uniformly spaced in time (or cycles). Ascent is integrated into the input specification allowing users to call a curated set of functions, including returning the results of a Boolean expression. Through this interface users can leverage Ascent to develop criteria to modify simulation parameters.

3.3 Use Cases

MARBL users face several workflow problems, and our use cases fall into two categories:

- Resource Maximization: both human and compute resources
- ALE Challenges: controlling and debugging ALE simulations

3.3.1 Resource Maximization. MARBL is a multi-physics code, and there are a number of types of physics that can be turned on or off depending on the problem. Since some of the physics can be much faster (by an order of magnitude), it is important to only turn on what is needed to minimize compute resources, thus maximizing the number of simulations a user can run. In some cases, expensive physics is needed in only a part of the problem and having costly physics on the entire run is prohibitively expensive. If the exact time of switch is known prior to the run, then a simple timer can be used to switch the physics. However, it is often the case that only some derived characteristic is known (e.g., turn on the expensive physics when the temperature exceeds some threshold). In these cases, a human must monitor the simulation and intervene, which creates a tension between human resources and compute resources.

3.3.2 ALE Challenges. Arbitrary Lagrangian-Eulerian (ALE) simulations [2, 11] use coupled Lagrangian and Eulerian remap/remesh phases to evolve hydrodynamic states. In the Lagrangian phase, the mesh moves with the material flow, and as the simulation progresses, the mesh can become distorted, even inverting computational cells. If the mesh becomes inverted, then the simulation results become numerically invalid. To prevent this from happening, an occasional Eulerian phase is used to remap or remesh the solution to a less distorted mesh. Exactly when to remap can be an art rather than a science, and in MARBL, remap is controlled using simple timers, and even with remap, a simulation is not guaranteed to finish. Users must monitor a simulation to either find better times to trigger the remap or use checkpoints to debug the source of the problematic mesh cells.

Since this process is typically an iterative human-in-the-loop workflow, again creating a tension between human resources and compute resources, automating ALE choices is a recent area of research. [13] demonstrated using neutral networks to connect mesh quality metrics to ALE decisions made by expert modelers. They then used those neural networks to automate ALE decisions testing several scenarios. Triggers can be used to describe conditions that signal a simulation is degrading. When such a trigger occurs, we can automatically collect relevant context (including mesh quality metrics) to present to users or we can automatically adjust the simulation by calling a remap operation. Our goal in this work is not to demonstrate a new state of the art ALE metric, but to show that ALE choices can leverage Ascent's trigger infrastructure, and thus future solutions can use the extended capabilities Ascent offers, or new methods could be deployed in Ascent for use in multiple simulation codes.

4 DEMONSTRATIONS

In this section, we demonstrate how Ascent's expressions can an provide users with a way to characterize conditions in which we want to adapt two example simulations, in ways that maximize resources and address ALE challenges. The demonstrations are:

- Controlling Physics Models
- Debugging ALE Crashes
- Controlling ALE Remaps

4.1 Controlling Physics Models

The Sedov blast problem is a standard hydrodynamics problem with an analytical solution. The problem deposits a region of high energy at the center of the mesh, and a shockwave propagates outward. MARBL includes a version of the Sedov blast problem that adds a magnetohydrodynamics model (MHD), and the additional implicit solve adds additional cost on top of the explicit solve used for pure hydrodynamics. Normally, the shockwave propagates symmetrically, but the addition of the MHD forces push back and breaks the symmetry.

To demonstrate Ascent's ability to feed information back to the simulation and automatically change physics, we created an expression that monitors the progress of the shockwave. The simulation starts with all physics turned on, allowing the wave to develop asymmetries. When the shockwave reaches a specific distance, we turn off the MHD, saving the cost of the implicit solve, then we let the asymmetric shock evolve using pure hydrodynamics.

We compared the runtime of a small problem using 72 MPI ranks on two nodes. The runtime of the problem with MHD on for the full run was 575 seconds. Using the trigger to turn off MHD approximately 40% of the way through the problem, resulted in a runtime of 455 seconds. Switching physics enabled a savings of over 20% of total runtime. If we ran many simulations as part of an ensemble, then we could run 5 simulations using triggers in the same time it would take to run 4 without them. Further, there are physics models that are far more expensive than MHD, which suggests there is significant opportunity for savings in other types of problems. In this case, Ascent's DSL gave us an expressive means to describe a complex feature (the outgoing shock) and make a decision based on the location of that feature, which is something a static timer cannot achieve.

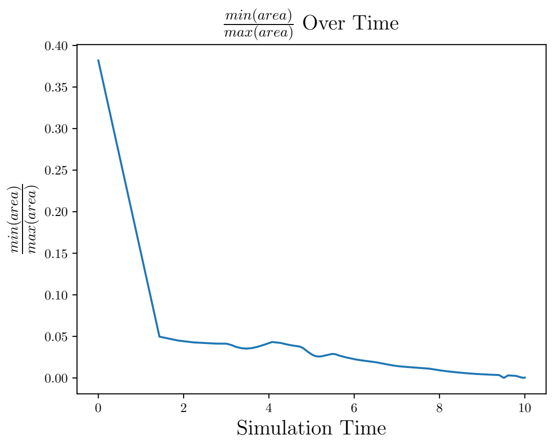

Figure 1: A plot of the ratio of the minimum cell area versus the maximum cell area over time during a 2D run.

4.2 Debugging ALE Crashes

The triple point problem is another standard shock test problem that uses two materials with three distinct states. The three regions interact and form a vortex that twists the computational mesh, testing a simulation's ability to handle mesh motion. If the mesh twists too much without a re-meshing, then the simulation will crash or slow to a halt (if the time steps become too small).

Debugging mesh tangling is an important part of an ALE user's workflow. Typically, a user will use visualization software to examine checkpoints before the simulation crashes to debug the cause of the crash (e.g., degenerate cells or large differences in cell volumes). Debugging can be a tedious manual process that consumes human resources, as it is an iterative process (debug-test-crash) until the simulation completes. Using expressions and triggers can be a valuable tool to assist users in the debugging process.

Ascent expressions can summarize features of the computational mesh as the simulation executes. Figure 1 shows a plot of the ratio of minimum and maximum cell areas during 2D run. The plot clearly indicates that the mesh quality is degrading over time, and without action, the simulation might not complete. Ascent also captures the spatial location of minimum and maximum values which can be used for identifying problematic areas of the mesh. Additionally, Ascent can calculate histograms of values and spatial distributions. Feeding this information back into the simulation provides a rich toolbox for users to create automated workflows that are tailored to specific simulations (e.g., dumping out checkpoints when the ratio dips below some threshold and correcting ALE issues by triggering remaps).

4.3 Controlling ALE Remaps

In this demonstration, we show that we can use Ascent's expression system to trigger a remap of the mesh, using use the same triple problem as in Section 4.2. We previously used mesh quality metrics as an indicator that the simulation state was degrading over time. Instead of mesh quality metrics, we examined dt through time (i.e., how far in simulation time the simulation advances per cycle). This is also a strong indicator that the simulation state is degrading.

The expression we used to trigger the remap used several quantities. First, we derived the average dt between calls to Ascent, which was every 1000 cycles. Second, we calculated the initial average dt during the first two time units of the simulation run and the sliding average of the dt during the previous 5000 cycles, using Ascent's history function (which allows us access to the time history of an expression). Next, we looked at the gradient of the $log(dt)$ to identify when dt was drastically dropping. Finally, the trigger fired, causing the simulation to remap, when the sliding average dropped below a threshold relative to the initial average and the gradient was negative. Listing 3 shows the full set of expressions used.

Figure 2 shows the results of two runs, with and without the trigger. Both runs complete 10 units of simulation time, but drastically differ in cycle count. The run without the trigger completed in 302,702 cycles, and the run with the trigger completed in 131,950 cycles, a greater than 50% decrease in the total cycles used. Using simple uniform timers is an option, but remapping can be an expensive global operation, and this demonstration shows that we can craft expressions that are far more expressive than the triggers that are currently used by the code.

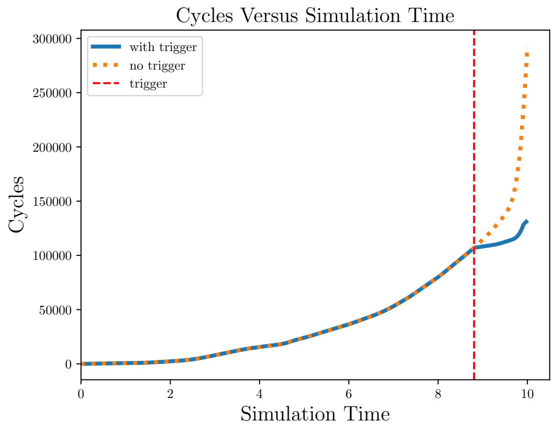

Figure 2: A plot of two simulations runs: one with no intervention (orange) and one with a trigger that causes the simulation to remap the mesh (blue). The x-axis is simulation time over 10 time units, and the y-axis is the number of simulation cycles. A steeper slop indicates that each cycle progresses the simulation more slowly to the target of 10 time units. The vertical red dotted line indicates when the trigger fired causing the remap.

5 CONCLUSION

There are a diverse set of complex issues running and managing simulation workflows. Our investments in situ visualization and analysis tools can help. These tools provide general capabilities across simulation codes that can help automatically address a variety of workflow issues. In this paper we demonstrated how we can leverage general in situ visualization and analysis to extend the steering capabilities of a multi-physics code. Using Ascent's expressions language we described a number of situations relevant to a multi-physics code user's workflow. By connecting this back to the simulation, we automatically steered the simulation's evolution and assisted in debugging ALE issues. In terms of future work, we want to explore more simulation use cases, develop additional constructs like Ascent's history gradient that help with these use cases, and connect Ascent in a similar way to more simulation codes.

REFERENCES

[1] Francis Alexander et al. 2020. Exascale Applications: Skin in the Game. *Philosophical Transactions of the Royal Society* 378, 2166 (2020).
[2] Robert W. Anderson, V. Dobrev, T. Kolev, R. Rieben, and V. Tomov. 2018. High-Order Multi-Material ALE Hydrodynamics. *SIAM J. Sci. Comput.* 40 (2018).
[3] Atanas Atanasov, Hans-Joachim Bungartz, Jérôme Frisch, Miriam Mehl, Ralf-Peter Mundani, Ernst Rank, and Christoph van Treeck. 2010. Computational steering of complex flow simulations. In *High Performance Computing in Science and Engineering, Garching/Munich 2009.* Springer, 63–74.
[4] Janine C. Bennett, Ankit Bhagatwala, Jacqueline H. Chen, C. Seshadhri, Ali Pinar, and Maher Salloum. 2015. Trigger detection for adaptive scientific workflows using percentile sampling. *CoRR* abs/1506.08258 (2015). arXiv:1506.08258 http://arxiv.org/abs/1506.08258
[5] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. 2011. Parallel computational steering and analysis for hpc applications using a paraview interface and the hdf5 dsm virtual file driver. In *Eurographics Symposium on Parallel Graphics and Visualization.* Eurographics Association, 91–100.
[6] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. 2012. Parallel computational steering for hpc applications using hdf5 files in distributed shared memory. *IEEE Transactions on Visualization and Computer Graphics* 18, 6 (2012), 852–864.
[7] Jonathan Chin, Peter V Coveney, and Jens Harting. 2004. The TeraGyroid project–collaborative steering and visualization in an HPC Grid for modelling complex fluids. In *UK All-hands e-Science Conference.*
[8] Bob K Danani and Bruce D D'Amora. 2015. Computational steering for high performance computing: applications on Blue Gene/Q system. In *Proceedings of the Symposium on High Performance Computing.* 202–209.
[9] Geoffrey C Fox, Devarshi Ghoshal, Shantenu Jha, Andre Luckow, and Lavanya Ramakrishnan. 2017. Streaming computational science: Applications, technology and resource management for hpc.
[10] GA Geist, James Arthur Kohl, and Philip M Papadopoulos. 1997. Cumulvs: Providing fault toler. ance, visualization, and steer ing of parallel applications. *The International Journal of Supercomputer Applications and High Performance Computing* 11, 3 (1997), 224–235.
[11] C.W Hirt, A.A Amsden, and J.L Cook. 1974. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *J. Comput. Phys.* 14, 3 (1974), 227–253. https://doi.org/10.1016/0021-9991(74)90051-5
[12] Seif Ibrahim, Cyrus Harrison, and Matthew Larsen. 2020. JIT's Complicated: A Comprehensive System For Derived Field Generation. In *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (Atlanta, GA, USA) *(ISAV'20).* Association for Computing Machinery, New York, NY, USA, 27–31. https://doi.org/10.1145/3426462.3426467
[13] Ming Jiang, Brian Gallagher, Noah Mandell, Alister Maguire, Keith Henderson, and George Weinert. 2019. A deep learning framework for mesh relaxation in arbitrary Lagrangian-Eulerian simulations. In *Applications of Machine Learning*, Michael E. Zelinski, Tarek M. Taha, Jonathan Howe, Abdul A. S. Awwal, and Khan M. Iftekharuddin (Eds.), Vol. 11139. International Society for Optics and Photonics, SPIE, 168 – 182. https://doi.org/10.1117/12.2529731
[14] Yuya Kawakami, Nicole Marsaglia, Matthew Larsen, and Hank Childs. 2020. Benchmarking In Situ Triggers Via Reconstruction Error. In *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (Atlanta, GA, USA) *(ISAV'20).* Association for Computing Machinery, New York, NY, USA, 38–43. https://doi.org/10.1145/3426462.3426469
[15] Samuel Kortas and Mohsin Ahmed Shaikh. 2020. Towards an HPC Service Oriented Hybrid Cloud Architecture Designed for Interactive Workflows. In *2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC).* IEEE, 36–46.
[16] 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 the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization* (Denver, CO, USA) *(ISAV'17).* Association for Computing Machinery, New York, NY, USA, 42–46. https://doi.org/10.1145/3144769.3144778
[17] Matthew Larsen, Amy Woods, Nicole Marsaglia, Ayan Biswas, Soumya Dutta, Cyrus Harrison, and Hank Childs. 2018. A Flexible System for in Situ Triggers. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (Dallas, Texas, USA) *(ISAV '18).* Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3281464.3281468
[18] Jan Linxweiler, Manfred Krafczyk, and Jonas Tölke. 2010. Highly interactive computational steering for coupled 3D flow problems utilizing multiple GPUs. *Computing and visualization in science* 13, 7 (2010), 299–314.
[19] Marta Mattoso, Jonas Dias, Kary ACS Ocana, Eduardo Ogasawara, Flavio Costa, Felipe Horta, Vitor Silva, and Daniel

De Oliveira. 2015. Dynamic steering of HPC scientific workflows: A survey. *Future Generation Computer Systems* 46 (2015), 100–113.

[20] Jurriaan D Mulder, Jarke J Van Wijk, and Robert Van Liere. 1999. A survey of computational steering environments. *Future generation computer systems* 15, 1 (1999), 119–129.

[21] Florian Niebling, Andreas Kopecki, and Martin Becker. 2010. Collaborative steering and post-processing of simulations on hpc resources: Everyone, anytime, anywhere. In *Proceedings of the 15th International Conference on Web 3D Technology*. 101–108.

[22] John R Ossyra, Ada Sedova, Matthew B Baker, and Jeremy C Smith. 2019. Highly Interactive, Steered Scientific Workflows on HPC Systems: Optimizing Design Solutions. In *International Conference on High Performance Computing*. Springer, 514–527.

[23] Steven G Parker and Christopher R Johnson. 1995. SCIRun: a scientific programming environment for computational steering. In *Supercomputing'95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. IEEE, 52–52.

[24] Sabine Rathmayer and Michael Lenke. 1997. A tool for on-line visualization and interactive steering of parallel hpc applications. In *Proceedings 11th International Parallel Processing Symposium*. IEEE, 181–186.

[25] RealityGrid [n.d.]. *RealityGrid*. http://www.rcs.manchester.ac.uk/research/realitygrid

[26] Morris Riedel, Th Eickermann, Sonja Habbinga, Wolfgang Frings, Paul Gibbon, Daniel Mallmann, Felix Wolf, Achim Streit, Th Lippert, Felix Wolf, et al. 2007. Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures. In *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. IEEE, 483–490.

[27] Morris Riedel, Wolfgang Frings, Thomas Eickermann, Sonja Habbinga, Paul Gibbon, Achim Streit, Felix Wolf, and Thomas Lippert. 2010. Collaborative interactivity in parallel hpc applications. In *Remote Instrumentation and Virtual Laboratories*. Springer, 249–262.

[28] Allen Sanderson, Alan Humphrey, John Schmidt, and Robert Sisneros. 2018. Coupling the Uintah framework and the VisIt toolkit for parallel in situ data analysis and visualization and computational steering. In *International Conference on High Performance Computing*. Springer, 201–214.

[29] Ashwin Shashidharan, Ranga Raju Vatsavai, Abhinav Ashish, and Ross K Meentemeyer. 2017. tFUTURES: Computational steering for geosimulations. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 1–10.

[30] Pavel Vasev. 2020. Analyzing an Ideas Used in Modern HPC Computation Steering. In *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*. IEEE, 1–4.

[31] Jeffrey Vetter and Karsten Schwan. 1997. High performance computational steering of physical simulations. In *Proceedings 11th International Parallel Processing Symposium*. IEEE, 128–132.

[32] Andreas Wierse, Ulrich Lang, and Roland Rühle. 1993. A system architecture for data-oriented visualization. In *Workshop on Database Issues for Data Visualization*. Springer, 148–159.

[33] Helen Wright, Robin H Crompton, Sanjay Kharche, and Petra Wenisch. 2010. Steering and visualization: Enabling technologies for computational science. *Future Generation Computer Systems* 26, 3 (2010), 506–513.

[34] Qi Wu, Tyson Neuroth, Oleg Igouchkine, Kwan-Liu Ma, Aditya Konduri, and Jacqueline H. Chen. 2020. Diva: A Declarative and Reactive Language for In-Situ Visualization. *CoRR* abs/2001.11604 (2020). arXiv:2001.11604 https://arxiv.org/abs/2001.11604

A ARTIFACT DESCRIPTION

In this section, we list the software that was used and the expressions that were used in the demonstrations.

A.1 Software

We use the Ascent in situ visualization library which is an open source project located on github:

- https://github.com/Alpine-DAV/ascent.

While MARBL is not open source, Ascent include proxy applications that can run similar problems, although the proxy applications are not mulit-physics codes. Ascent's proxy application documentation can be found here:

- https://ascent.readthedocs.io/en/latest/ExampleIntegrations.html

Links to specific problems can be found here:

- Lulesh: Sedov Problem.
- Laghos: Triple Point.

A.2 Expressions

In this section, we list the full Ascent actions used in Section 4 for reproducibility.

Listing 1: The Ascent actions used in Section 4.1.

```
-
  action: "add_queries"
  queries:
    q1:
      params:
        expression: "max(field('density'))"
        name: "max_den"
    q11:
      params:
        expression: "max_den.position"
        name: "pos"
    q12:
      params:
        expression: "vector(pos.x, pos.y, pos.z)"
        name: "pvec"
    q13:
      params:
        expression: "magnitude(pvec)"
        name: "dist"
    q14:
      params:
        expression: "max_den.value > 2.0"
        name: "max_v"
    q2:
      params:
        expression: "max_v and (dist > 0.28)"
        name: "trigger"
```

Listing 2: The Ascent actions used in Section 4.2

```
-
  action: "add_pipelines"
  pipelines:
    pl1:
      f1:
        type: "mesh_quality"
        params:
          metric: area
          topology: main
-
  action: "add_queries"
  queries:
    time:
      params:
        expression: "time()"
        name: time
    max_area:
      pipeline: pl1
      params:
        expression: "max(field('area'))"
        name: max_area
    min_area:
      pipeline: pl1
      params:
        expression: "min(field('area'))"
        name: min_area
    area_ratio:
      pipeline: pl1
      params:
        expression: "min_area.value/max_area.value"
        name: area_ratio
```

Listing 3: The Ascent actions used in Section 4.3.

```
-
  action: "add_queries"
  queries:
    time:
      params:
        expression: "time()"
        name: time
    cycle:
      params:
        expression: "cycle()"
        name: cycle
    dt:
      params:
        expression: "(time - history(time,relative_index=1)) /(cycle - history(cycle, relative_index=1))"
        name: dt
    log_dt:
      params:
        expression: "log(dt)"
        name: log_dt
    start_ave:
      params:
        expression: "avg(replace(history_range(dt,first_absolute_time=0.0, last_absolute_time=2.5),nan(),0.0008))"
        name: start_ave
    now_ave:
      params:
        expression: "avg(replace(history_range(dt,first_absolute_time=time-0.5, last_absolute_time=time), nan(), 0.0008))"
        name: now_ave
    grad:
      params:
        expression: "gradient_range(log_dt,first_relative_index=1, last_relative_index=5)"
        name: grad
    grad_ave:
      params:
        expression: "avg(grad)"
        name: grad_ave
    trigger:
      params:
        expression: "(avg(grad) < -1.0) and (start_ave * 0.09 > now_ave)"
```