Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

doi:10.3233/APC200080

When Parallel Performance Measurement and Analysis Meets In Situ Analytics and Visualization

Allen D. MALONY [a], Matt LARSEN [b] Kevin HUCK [a] Chad WOOD [a]
Sudhanshu SANE [c] Hank CHILDS [c]

[a] *Oregon Advanced Computing Institute for Science and Society (OACISS)*
[b] *Lawrence Livermore National Laboratory*
[c] *Department of Computer and Information Science University of Oregon*

Abstract.
Large scale parallel applications have evolved beyond the tipping point where there are compelling reasons to analyze, visualize and otherwise process output data from scientific simulations *in situ* rather than writing data to filesystems for post-processing. This modern approach to in situ integration is served by recently developed technologies such as *Ascent*, which is purpose-built to transparently integrate runtime analysis and visualization into many different types of scientific domains. The TAU Performance System (*TAU*) is a comprehensive suite of tools that have been developed to measure the performance of large scale parallel libraries and applications. TAU is widely-adopted and available on leading-edge HPC platforms, but has traditionally relied on post-processing steps to visualize and understand application performance. In this paper, we describe the integration of Ascent and TAU for two complementary purposes: Analyzing Ascent performance as it serves the visualization needs of scientific applications, and visualizing TAU performance data at runtime. We demonstrate the immediate benefits of this in situ integration, reducing the time to insight while presenting performance data in a perspective familiar to the application scientist. In the future, the integration of TAU's performance observations will enable Ascent to reconfigure its behavior at runtime in order to consistently stay within user-defined performance constraints while processing visualizations for complex and dynamic HPC applications.

Keywords. HPC, performance measurement, runtime visualization.

1. Introduction

Parallel applications developed for large-scale, high-performance computing (HPC) continue to increase in sophistication and complexity. To a great extent, this is driven by the advances in computational modeling of scientific and engineering phenomena that will demand the next-generation hardware technologies fueling the HPC evolution. The ability of applications to harness the greater computational resources of HPC systems will deliver results of finer precision, higher resolution, and more significant predictive power. The challenge, of course, is to develop applications that can maximize the performance potential of present and future HPC platforms. This represents the flip side of the sophistication and complexity problem. Applications will need advanced parallel programming, language, runtime system, and communication interface technologies to produce programs that can utilize the heterogeneous many-core processors, multi-level memory architecture, and fast interconnect hardware effectively and do so in a performance portable manner.

The dual nature of what defines HPC application success — advanced scientific outcomes and highly-efficient execution — is reflected in tools created to further enhance that success, again in the context of HPC sophistication and complexity. For example, analysis and visualization tools are central to the understanding of science and engineering simulation results. The last 30 years has seen a steady progression from tools generating analysis and visualization products *post hoc* to those running *in situ* with the application [1]. The reasons are consequential of simulation fidelity and HPC scale, making it increasingly intractable to save and process huge modeling data offline [2]. In a similar manner, the importance of parallel performance measurement, analysis, and visualization tools is central to understanding and tuning applications on HPC machines. Contemporaneous to the transition of *in situ* analysis and visualization, runtime performance data introspection, analytics, and feedback are becoming more relevant in performance systems. Again, the reasons are due to HPC idiosyncrasies, including larger performance data size, more factors affecting performance variation, and non-deterministic performance dynamics as a result of asynchronous execution, all making *post mortem* performance methods less productive.

Like *brothers from a different mother*, we consider in this paper the opportunities for the inter-operation of a parallel performance system with an *in situ* analysis and visualization framework. Interestingly, the shared HPC heritage positions these tools today in a place that begs for their integration and supports it. We will demonstrate the merits of the endeavor by focusing on two leading efforts: the *Ascent in situ* project [3] and the *TAU Performance System*® [4]. Ascent is being developed by a multi-institution effort funded by the U.S. Department of Energy (DOE) Exascale Computing Project (ECP) [5] to deliver in situ analysis and visualization technology ECP application teams. TAU provides portable, robust performance measurement and analysis of HPC applications and systems.

There are three perspectives that we will investigate, the first two of which we will describe in this paper. One perspective looks at the use of TAU to instrument, measure, and analyze the Ascent infrastructure. TAU is particularly suited to observing the execution of large-scale software [6] and can directly be applied to characterize the performance of Ascent components. Ascent's performance will correlate with the application-specific analytics and visualization requirements for which it is being used. Based on the performance analysis, Ascent developers will be able to understand inefficiencies and optimize performance for specific usage scenarios.

Another perspective involves the use of Ascent for application performance analytics and visualization. Here our interest is to gather and process performance data online (from TAU's measurement of the application measurement and systems-level information) and utilize Ascent's infrastructure to analyze and visualize the data *in situ*. Specifically, we program Ascent's runtime with filters to interface with TAU performance measurements, application-specific values, and system information during periods when Ascent is invoked.

It is reasonable to assume that TAU and Ascent will co-exist in HPC platforms and applications. Thus, an outcome in pursuing the two perspectives above is to validate the cross-leveraging of Ascent and TAU technologies. A third, intriguing perspective comes from more tightly-coupled integration of TAU and Ascent whereby they are being used cooperatively in online tuning and adaptive control of the application. We envision this taking several forms. For instance, suppose that the user constrains *in situ* analytics and visualization to take no more than 10% of the application's execution time. TAU could be used to measure the performance of both the application and Ascent, thereby informing the Ascent infrastructure when corrective action is necessary.

Another possibility is to develop joint analytics that take into account a combination of performance data, application variables, and other execution state to guide policies concerning how the application should advance. Innovations taking place in both Ascent and TAU for supporting application *triggers* [7], *feedback* mechanisms, and autonomic management make this especially salient for integration purposes. Furthermore, there are strong motivations to extend Ascent's and TAU's operation to scientific workflows where *in situ* concerns of computational productivity and performance efficiency involve interactions between multiple simulation modules and workflow components.

Our plan is to evaluate these perspectives with benchmark applications taken from the ECP proxy applications project. These include two of the applications that are part of the Ascent test programs, LULESH and Cloverleaf3D. We ran our experiments on large-scale HPC machines at DOE national laboratories. The main research objectives are to investigate effective methods and explore development strategies for the integration of two state-of-the-art runtime infrastructures for HPC, principally Ascent for *in situ* analytics and visualization and TAU for parallel performance measurement and analysis.

2. Applied Technologies

2.1. Ascent

Ascent [3] is a library for in situ visualization and analysis. Simulation geometry and results are passed to Ascent at runtime in order to generate periodic analysis results without the need to write much larger simulation data to disk for postmortem analysis. It differs from other in situ libraries in its focus on "flyweight processing," meaning small API, small binary size, small execution overhead, small memory footprint, and few dependencies on other libraries. Ascent supports zero-copy in situ (meaning that it can share memory with a simulation code), and supports parallelism both within a node and across nodes. Its parallelism within a node comes from incorporating the VTK-m project [8], which focuses on delivering portable performance across many-core architectures for visualization and analysis algorithms. It has been demonstrated good performance on 16,384 cores of the Oak Ridge Titan machine [9], 16,384 GPUs on Lawrence Livermore's Summit machine, and 2,048 GPUs on Oak Ridge's Summit.

2.2. TAU and PerfStubs

The TAU Performance System [4] is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C/C++, Java, and Python. TAU is capable of gathering performance information through system-interrupt-based sampling and/or instrumentation of functions, methods, basic blocks, and statements. The instrumentation can be inserted in the source code automatically with a TAU specific compiler wrapper based on the Program Database Toolkit (PDT) [10], dynamically using DyninstAPI [11], at runtime in the Java Virtual Machine or Python runtime, or manually using the instrumentation API (application programming interface). TAU measurements represent first-person, per-OS (operating system) thread measurements for all processes in a distributed application, such as an MPI application. TAU measurements are collected as profile summaries and/or a full event trace.

While application developers are willing to instrument their source code for performance measurement or correctness testing, they are frequently reluctant to add a build dependency for their library or application. The TAU library has many useful features, however can be complex to configure for a given system, and has several configuration options that are mutually exclusive and may require multiple configurations and builds for a given performance experiment. Also, a library such as Ascent is meant to be integrated into larger simulation applications and a complex configuration/build process for "optional" features will prevent adoption of these technologies. Finally, many applications already include some instrumentation to provide rudimentary performance measurement for the purposes of reporting at the end of execution. For these reasons, we have developed a simple instrumentation interface library called *PerfStubs* that attempts to resolve API symbols at link time (using weak symbol overrides) or at runtime (using the dynamic library loader). PerfStubs itself is a small library (one source file) with no additional build dependencies and can quickly be installed on a system.

If the PerfStubs symbols are not defined in the application symbol table at runtime, the instrumentation API will check to see if the function pointer is defined (not-null) and if not, return – an acceptable amount of negligible overhead. If the symbols are resolved at the program startup process, function pointers are assigned the resolved addresses and the PerfStubs API will "feed" any performance measurement tool that implements the tool interface. TAU includes the `tau_exec` script that will preload the TAU shared object libraries and provide the symbol implementations needed by the *perfstubs* interface. Other measurement libraries (e.g. APEX, Score-P, Caliper) could also implement the simple API and be used with the interface. Because the instrumentation interface is pre-processor macro-based, it can be entirely removed at compile time if the *PerfStubs* API is not desired. In fact, the Ascent library already has implemented its own macro-based instrumentation, and the PerfStubs API was easily integrated into that code, as well as into specific places in the Ascent code base, as described in Section 3.

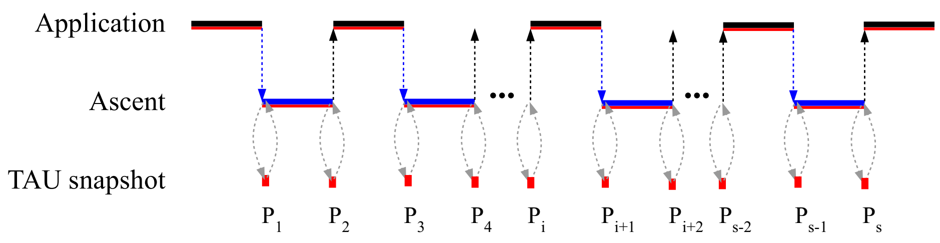

Figure 1. The application (black line) and Ascent (blue line) execute synchronously, with TAU performance measurements (red line) possibly enabled. The transition points between them is an opportunity to link in TAU performance data analysis, including the passing of information to Ascent in a friendly manner for further processing. Profile snapshots are an example of online performance data processing.

2.3. Integration Model

The Ascent operational design provides the basis for the strategy we pursued for TAU integration. Specifically, Ascent is invoked synchronously by the application at certain places during its execution. Ascent then operates while the application is halted. Upon completion of its work, Ascent returns to the application and it continues. This process repeats until the application finishes. Adding TAU to the mix is straightforward. First, in general, we are interested in performance measurement of both the application and Ascent. This is enabled through TAU's instrumentation and measurement mechanisms. Second, gaining access to performance data at runtime is possible with new TAU's plugin architecture. The application/Ascent transition points present an opportunity to look at the current measurements, run analytics, and pass results to Ascent for further processing. In essence, the transitions are used as triggers for TAU analysis.

Our integration design is demonstrated in Figure 1. Shown is a sequence of phases of application execution (black line) followed by Ascent execution (blue line). The dashed arrows indicate the transition points. The red line indicates TAU performance measurement taking place during both the application and Ascent. Dashed gray lines further highlight calls to the TAU plugin (red box) at the beginning and end of Ascent processing. For example, the plugin could be capturing a snapshot of the present performance data state, designated as P_i. These snapshots could be stored for post-mortem analysis and/or processed online.

Profile snapshots can be used to isolate the application's performance from Ascent's performance. From Figure 1, we can use P_i, P_{i+1}, and P_{i+2} to compute the performance for the application phase by "subtracting" P_i from P_{i+1} for every event and metric measured. Similarly, we can compute the performance for the Ascent phase by subtracting P_{i+1} from P_{i+2}. This is similar to the procedure we implemented in the examples described in Section 3. If the TAU plugin stored the computed performance for each phase, it is further possible to compare between phases to detect certain features or changes in performance behavior that might reflect application dynamics.

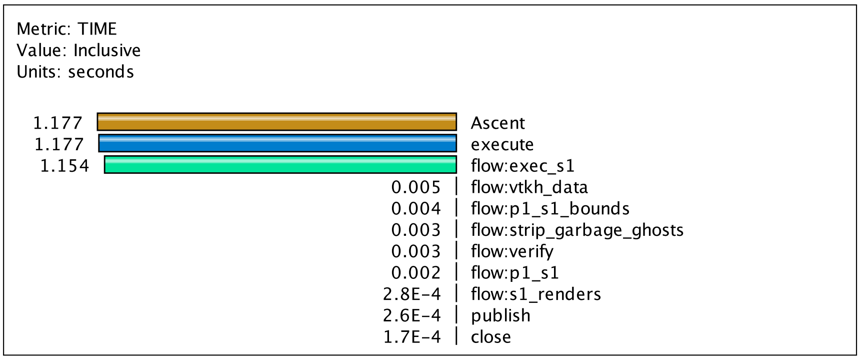

Figure 2. TAU profile data from a representative run of Lulesh integrated with Ascent. MPI and Lulesh timers have been filtered out for space considerations. Only data from rank 0 is shown.

3. Application Examples

3.1. Measuring Ascent with TAU

The first goal of the Ascent-TAU integration is to measure the Ascent library for the purpose of performance evaluation and eventually, guided execution based on performance characteristics (future work). As described in Section 2.2, the Ascent library includes its own instrumentation macros. It was straightforward to integrate `perfstubs_start()` and `perfstubs_stop()` API calls into these macro definitions that are frequently used when Ascent is integrated into a simulation. In addition, primary entry points to the Ascent library were instrumented, such as the `Ascent()` object constructor, `open()`, `close()`, `publish()` and `execute()` operations as well as the `Flow` operation pipeline executed by the `execute()` function. A TAU static phase [12] was also defined around the region of code where the simulation pauses execution in order for Ascent to render simulation output. Figure 2 shows an example TAU profile measurement of the Ascent library integrated with the Lulesh application.

3.2. Visualizing TAU data with Ascent

The second goal of the integration is to use the Ascent library to visualize performance data from the application. This can be achieved using different perspectives. For example, the application performance data can be rendered as a collection of stacked bar charts, representing the per-process performance profile from each of the MPI ranks. Figure 3 (left) shows the performance data from MPI ranks, represented as stacked bar charts. Each color represents a different timed region of the application, showing only the top 5 contributors (the rest are aggregated).

However, a much more interesting perspective is shown when the performance data is projected in the scientific domain. Figure 3 (right) shows the respective time spent in the main computational loop for all sub-domains at the end of the last iteration. What had started as a regular grid has been distorted due to the nature of the Lagrangian computation. Interestingly, the MPI rank computing

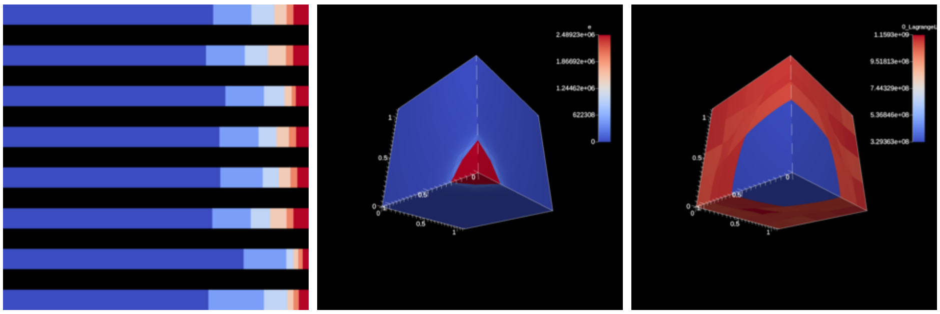

Figure 3. The figure on the left shows output from Lulesh 2.0.3, running with 8 ranks. Each stacked bar represents the performance profile of each rank at the end of the 10th iteration. The other two figures show simulation output from Lulesh 2.0.3 running with 64 MPI ranks, after iteration 4264. The middle figure shows the energy value at the end of the simulation, the right figure shows the relative time spent in the main computation loop of the simulation for each process in the domain, where each process is assigned one of the 4x4x4 (distorted) subdomains.

the region with the largest energy level also spends the least amount of time in the computation.

3.3. Performance Comparison

To further demonstrate the Ascent-TAU integration, we use an in situ algorithm used for flow analysis and visualization. Lagrangian analysis is an in situ data reduction operator used to capture the behavior of time-varying computational fluid dynamics (CFD) simulations. Lagrangian analysis involves the placement of particles and the calculation of particle trajectories across the entire spatial domain. Particle trajectories are calculated using vector fields generated by the simulation code.

In our study, we consider two Lagrangian analysis techniques which offer different workload characteristics. The first Lagrangian analysis technique, referred to as $Lagrangian_{MPI}$, is communication-based and requires all processes to synchronize every cycle. This method involves exchanging particles between nodes as they cross spatial boundaries during the calculation of particle trajectories. The second Lagrangian analysis technique, referred to as $Lagrangian_{BTO}$, is a communication-free method. This algorithm chooses to discard particles that exit the local node's spatial domain.

Our experiments use a hydrodynamics proxy application Cloverleaf3D and are run on Summit[1] at Oak Ridge National Laboratory. In each test, we use 48 MPI tasks across 8 nodes, with each MPI task using a single GPU for particle advection. $Lagrangian_{MPI}$ uses MPI to exchange particles between ranks every cycle. For each technique we considered two workloads for number of particles used: 1.56M and 12.48M. The grid resolution of the Cloverleaf3D simulation is set to 232^3 and we execute 50 cycles each of 0.01 step size. In each case, we save the particle trajectory locations after 10 cycles, i.e., 5 rounds of I/O over 50 cycles.

[1]For Summit technical specs, see https://www.olcf.ornl.gov/summit/

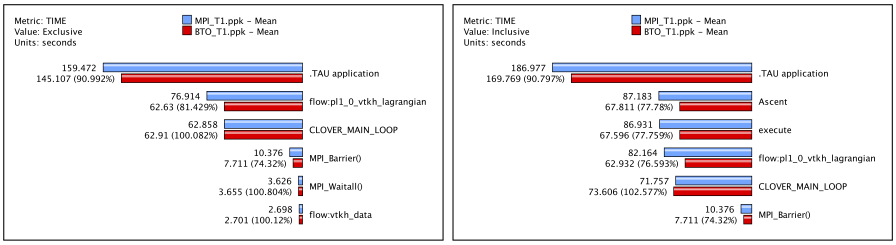

Figure 4. Exclusive and Inclusive time comparisons between MPI and BTO methods. The analysis shows that the BTO method (red) is faster because it generates less synchronization at `MPI_Barrier` and is less computationally expensive in the `pl1_0_vtkh_lagrangian` flow step in analysis.

Lagrangian analysis uses particle advection capabilities from the VTK-m library and is available for use via Ascent.

Figure 4 shows a performance comparison between the two methods when simulating the larger number of particles used (12.48M). As can be seen in both the exclusive (time not including sub-routines/-timers) and inclusive (time including sub-routines/-timers) measurements, the $Lagrangian_{BTO}$ method is less computationally expensive, and therefore less time consuming. the $Lagrangian_{BTO}$ method is also less I/O intensive, and a majority of the difference is summarized by the comparison of the time spent in the `flow:pl1_0_vtkh_lagrangian` step of the processing pipeline.

4. Related Work

Typically, performance data is visualized and represented in the physical and/or logical context of the hardware and/or software resources used in the simulation. Data is organized by processes and threads, and visualized with respect to nodes, network topologies and CPU architectures. Scalasca is a powerful performance system that has extended support in its Cube 3D analysis [13] to show how performance data is distributed across a parallel execution using a computational topology base on a cube topology. TAU provides similar capabilities by mapping performance data to network coordinates captured as metadata [14]. Husain and Gimenez's work on Mitos [15] and MemAxes [16] use memory hierarchy and architecture metadata to provide the context for performance measurements. BoxFish [17] also demonstrated the value of visualizing projections of performance data from multi-dimensional coordinate systems, providing a hierarchical data model for combining visualizations and interacting with data.

Huck et al. [18] integrated performance data with simulation output in order to project the performance data into the scientific domain. However, that technique required post-processing of both the performance and simulation data and did not allow for *in situ* processing. Using the Scalable Observation System [19], performance data was aggregated over SOS and queries were executed to extract performance data and generate VTK output files [20]. Using a similar approach, fusion simulation performance data was aggregated and exported to VTK files [21]. The authors of those papers were forced to re-define the physical domains in order to map the performance data and/or map the performance data to physical and/or logical coordinates of the allocation. In contrast, the work described in this paper has the ability to reuse the scientific domain defined for visualizing the simulation data. Weber [22] has also visualized performance data at runtime, albeit in a similar perspective to post-mortem trace visualization. Sanderson [23] is the closest work related to this paper, in that they visualized performance data at runtime in the scientific domain.

5. Conclusion and Future Work

In this paper, we have presented the integration between the Ascent in situ visualization and analysis library and the TAU Performance System. We instrumented the Ascent library with an instrumentation coupling library to understand its performance characteristics with TAU, and used the Ascent library to visualize TAU performance data during runtime of proxy applications. We used the TAU instrumentation to compare two Lagrangian analysis implementations on the Summit system. In terms of future work, we believe our approach is very relevant to nascent cost modeling efforts in the scientific visualization community. Among these are works to optimize algorithms [24,25], as well as fit in situ algorithms within time and power budgets [26,27,28]. In each of these efforts, the researchers studied performance a priori, and then used the findings to direct their algorithms. This limits the relevance of their approaches to the cases where they can perform performance studies, digest results, and calibrate their algorithms. With performance measurements, this process could be automated, meaning that researchers could develop algorithms that adapt during runtime and with no a priori performance studies.

References

[1] A. Bauer et al. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms, a State-of-the-Art (STAR) Report. In *Eurographics Conference on Visualization (EuroVis)*, June 2016.

[2] U. Ayachit et al. Performance Analysis, Design Considerations, and Applications of Extreme-scale In Situ Infrastructures. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 79:1–79:12, November 2016.

[3] M. Larsen et al. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV)*, pages 42–46, November 2017.

[4] S. Shende and A. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[5] P. Messina. The Exascale Computing Project. *Computing in Science and Engineering*, 19(3):63—67, 2017.

[6] A. Malony et al. Methods and Strategies for Parallel Performance Measurement and Analysis: Experiences with TAU and HPCToolkit. In *Performance Tuning of Scientific Applications*. CRC Press, 2010.

[7] M. Larsen et al. A Flexible System for In Situ Triggers. In *In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV)*, pages 1–6, November 2018.

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

[9] J. Kress et al. Comparing the Efficiency of In Situ Visualization Paradigms at Scale. In *ISC High Performance*, pages 99–117, Frankfurt, Germany, June 2019.

[10] K. Lindlan et al. A tool framework for static and dynamic analysis of object-oriented software with templates. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, Washington, DC, USA, 2000. IEEE Computer Society.

[11] W. R. Williams et al. Dyninst and mrnet: Foundational infrastructure for parallel tools. In *Tools for High Performance Computing 2015*, pages 1–16. Springer, 2016.

[12] A. D. Malony et al. Phase-based parallel performance profiling. In *PARCO*, pages 203–210, 2005.

[13] D. Lorenz et al. Extending scalasca's analysis features. In *Tools for High Performance Computing 2012*, pages 115–126. Springer Berlin Heidelberg, 2013.

[14] W. Spear et al. An approach to creating performance visualizations in a parallel profile analysis tool. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7156 of *Lecture Notes in Computer Science*, pages 156–165. Springer Berlin Heidelberg, 2012.

[15] B. Husain et al. Relating memory performance data to application domain data using an integration api. In *Proceedings of the 2nd Workshop on Visual Performance Analysis*, page 5. ACM, 2015.

[16] A. A. Gimenez et al. Memaxes: Visualization and analytics for characterizing complex memory performance behaviors. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[17] K. Isaacs et al. Exploring performance data with boxfish. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1380–1381. IEEE, 2012.

[18] K. A. Huck et al. Linking performance data into scientific visualization tools. In *2014 First Workshop on Visual Performance Analysis*, pages 50–57, Nov 2014.

[19] C. Wood et al. A scalable observation system for introspection and in situ analytics. In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools*, pages 42–49. IEEE Press, 2016.

[20] C. Wood et al. Projecting performance data over simulation geometry using sosflow and alpine. In *Programming and Performance Visualization Tools*, pages 201–218, Cham, 2019. Springer International Publishing.

[21] J. Y. Choi et al. Coupling exascale multiphysics applications: Methods and lessons learned. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 442–452, Oct 2018.

[22] M. Weber et al. Online performance analysis with the vampir tool set. In *International Workshop on Parallel Tools for High Performance Computing*, pages 129–143. Springer, 2017.

[23] A. Sanderson et al. Coupling the uintah framework and the visit toolkit for parallel in situ data analysis and visualization and computational steering. In *High Performance Computing*, pages 201–214, Cham, 2018. Springer International Publishing.

[24] V. Bruder et al. Prediction-based load balancing and resolution tuning for interactive volume raycasting. *Visual Informatics*, 2017.

[25] M. Larsen et al. Performance Modeling of In Situ Rendering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*, pages 24:1–24:12, Salt Lake City, UT, November 2016.

[26] M. Dorier et al. Adaptive performance-constrained in situ visualization of atmospheric simulations. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 269–278, Sept 2016.

[27] M. Dorier et al. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 67–75, Oct 2013.

[28] S. Labasan et al. PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 1–10, Barcelona, Spain, June 2017.