



Hank Childs · Janine C. Bennett ·
Christoph Garth *Editors*

In Situ Visualization for Computational Science

Mathematics and Visualization

Series Editors

Hans-Christian Hege, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB),
Berlin, Germany

David Hoffman, Department of Mathematics, Stanford University, Stanford, CA,
USA

Christopher R. Johnson, Scientific Computing and Imaging Institute, Salt Lake
City, UT, USA

Konrad Polthier, AG Mathematical Geometry Processing, Freie Universität Berlin,
Berlin, Germany

The series *Mathematics and Visualization* is intended to further the fruitful relationship between mathematics and visualization. It covers applications of visualization techniques in mathematics, as well as mathematical theory and methods that are used for visualization. In particular, it emphasizes visualization in geometry, topology, and dynamical systems; geometric algorithms; visualization algorithms; visualization environments; computer aided geometric design; computational geometry; image processing; information visualization; and scientific visualization. Three types of books will appear in the series: research monographs, graduate textbooks, and conference proceedings.

More information about this series at <https://link.springer.com/bookseries/4562>

Hank Childs · Janine C. Bennett · Christoph Garth
Editors

In Situ Visualization for Computational Science



Springer

Editors

Hank Childs
Computer and Information Science
University of Oregon
Eugene, OR, USA

Janine C. Bennett
Extreme-Scale Data Science and Analytics
Sandia National Laboratories
Livermore, CA, USA

Christoph Garth
Scientific Visualization Lab
University of Kaiserslautern
Kaiserslautern, Germany

ISSN 1612-3786
Mathematics and Visualization
ISBN 978-3-030-81626-1
<https://doi.org/10.1007/978-3-030-81627-8>

ISSN 2197-666X (electronic)
ISBN 978-3-030-81627-8 (eBook)

Mathematics Subject Classification: 00A66

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*We dedicate this book to Dr. Lucy Nowell,
who played a tremendous role in organizing
and inspiring our community.*

Preface

This book is devoted to *in situ* visualization for computational science. The definition of the latter topic, *computational science*, is widely understood: using computers to simulate scientific phenomena. The Latin phrase *in situ*, however, can create some confusion. When applied to visualization, the definition of *in situ* is often stretched a bit further than its literal translations, which include “on site,” “in position,” and “in place.” For this book, we use a simple definition for *in situ* visualization: visualizing data as it is generated. This definition certainly covers the case where visualization routines run the moment data is generated and using the same compute resources, but it also covers cases where data is sent to other compute resources for visualization.

While *in situ* visualization is not a new topic, with early research works dating back to the 1990s [6–9, 11], the last decade has seen an explosion of interest. This interest primarily comes in the context of high-performance computing and stems from supercomputing trends. Specifically, computational power is increasing much faster than storage capacity and transfer bandwidth, which disrupts the traditional “post hoc” workflow where simulations save their data to disk for later exploration by dedicated visualization programs.

Unfortunately, understanding the best approaches for *in situ* processing can be difficult. It is a highly interdisciplinary field, spanning computational science, high-performance computing, and scientific visualization communities. Further, it comprises a broad set of topics including software systems, code coupling (i.e., integration) between simulation and visualization codes, data models, execution models, and visualization and analysis algorithms and tools.

The Purpose Of This Book—Why Is It Needed and Who Is It For? Many other efforts have also considered *in situ* visualization, exploring motivation, challenges, best practices, and the latest research advancements. These include funding agency reports [1, 5, 12], position papers [4, 10], a Dagstuhl seminar [3], a survey paper [2], hundreds of research papers, and two annual workshops (The Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization, or “ISAV,” and the Workshop on In Situ Visualization, or “WOIV”).

So what problem is this book trying to solve? We feel there is a spectrum of needs when it comes to understanding *in situ*, and there is no resource for the “middle”

of the spectrum. At one extreme, it is easy to gain a high-level understanding of motivations and challenges via the excellent reports and position papers that exist today. At the other extreme, researchers invested in becoming *in situ* experts can read the hundreds of papers devoted to the topic. But there is a gap in the “middle” of the spectrum—informing beyond the high-level understanding, but not requiring years of study. This book is our attempt to fill this gap. Further, we feel this “middle” audience is diverse, and so we have aimed for a book that is accessible to a wide audience. Specifically, we hope that it will be useful to computational scientists, to high-performance computing scientists, and to (non-*in situ*) visualization scientists alike.

The History of This Book The idea for this book began at a Dagstuhl seminar, “*In Situ Visualization for Computational Science*,” in July 2018. This seminar brought together researchers and practitioners across the three aforementioned communities (computational science, high-performance computing, and scientific visualization) from Europe, the Americas, and Asia, to share research findings, to identify lines of open research, and to determine a medium-term research agenda that addresses the most pressing problems. In addition to identifying many technical areas in need of study, some participants also identified the need for a resource that would go deeper than a workshop report in describing the state of the art for *in situ* processing, which served as the motivation for this project. After the seminar, the editors of this book issued an open call for contributions, soliciting chapters that, together, would provide a deeper understanding of *in situ* processing that has previously been available. Most of the contributions synthesized previous research directions, while a handful were aimed at emerging results that represent very promising new directions. Each of the chapters went through a peer review process for both content and readability, and the entire book went through an additional round of editing.

How to Read This Book There is no need to read this book sequentially, as each chapter is written as a stand-alone work. That said, to prevent redundancy between the chapters, foundational topics are summarized in an introductory chapter (Chap. [In Situ Visualization for Computational Science: Background and Foundational Topics](#)). Therefore, we recommend that readers new to *in situ* processing read this chapter before going on to other chapters. The topics covered in the introductory chapter include a historical perspective of scientific visualization on supercomputers, why computing trends are increasingly mandating *in situ* processing, an overview of the types of *in situ* systems, a discussion of challenges with *in situ* processing, and descriptions of leading approaches.

The remainder of the book is organized into four parts. Chapters in the first three parts each synthesize existing published works within a focused area, effectively providing a series of “mini-surveys.” The fourth part contains new results that have not been published previously, as well as works speculating how existing techniques can be applied within an *in situ* setting. Specifically:

- Part I: Data Reduction Techniques. This part consists of four chapters, each devoted to a different transformation/reduction idea. For these chapters, the

general idea is to use in situ processing to transform and reduce data to a smaller form, so that it can be saved to disk and be explored post hoc.

- Part II: Workflows and Scheduling. This part consists of four chapters, including discussions of how to allocate resources efficiently, how to manage workflows, and how to manage many simultaneous simulations (ensembles) with in situ processing.
- Part III: Tools. This part consists of five chapters, discussing different frameworks for in situ processing, the decisions they have made, and their successes on real-world problems.
- Part IV: Research Results and Looking Forward. This part consists of six chapters, covering diverse topics, some of which overlap thematically with Parts I and II, while others explore entirely new areas, such as in situ rendering and machine learning.

In summary, this book brings together contributions from some of the most prominent and recognized researchers and practitioners in the field of in situ visualization. Its content covers summarizations of research results, discussions of the latest research trends, and new insights into in situ visualization. In all, our goal is to provide a valuable resource to computational scientists, high-performance computing scientists, and visualization scientists for learning about in situ visualization. We also feel this book can be a valuable reference for current practitioners and researchers. Whichever part of the intended audience you fall into, we very much hope that you find value in this book.

Eugene, OR, USA
Livermore, CA, USA
Kaiserslautern, Germany
January 2021

Hank Childs
Janine C. Bennett
Christoph Garth

References

1. Ahern, S., et al.: Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization (2011)
2. Bauer, A.C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., Moreland, K., O’Leary, P., Vishwanath, V., Whitlock, B., Bethel, E.W.: In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. Computer Graphics Forum (CGF) **35**(3), 577–597 (2016)
3. Bennett, J.C., Childs, H., Garth, C., Hentschel, B.: In Situ Visualization for Computational Science (Dagstuhl Seminar 18271). Dagstuhl Reports **8**(7), 1–43 (2018).
4. Childs, H., Bennett, J., Garth, C., Hentschel, B.: In Situ Visualization for Computational Science. IEEE Comput. Graph. Appl. (CG&A) **39**(6), 76–85 (2019)
5. Deelman, E., Peterka, T., Altintas, I., Carothers, C.D., Kleese van Dam, K., Moreland, K., Parashar, M., Ramakrishnan, L., Taufer, M., Vetter, J.: The Future of Scientific Workows. Int. J. High Perform. Comput. Appl. **32**(1), 159–175 (2018)
6. Haimes, R.: PV3-A Distributed System for Large-Scale Unsteady CFD Visualization. In: 32nd Aerospace Sciences Meeting and Exhibit, p. 321 (1994)

7. Haimes, R., Barth, T.: Application of the PV3 Co-Processing Visualization Environment to 3-D Unstructured Mesh Calculations on the IBM SP2 Parallel Computer. In: Proc. CAS Workshop (1995)
8. Haimes, R., Edwards, D.E.: Visualization in a Parallel Processing Environment. In: Proceedings of the 35th AIAA Aerospace Sciences Meeting, number AIAA Paper, pp. 97–0348 (1997)
9. Ma, K.L.: Runtime Volume Visualization for Parallel CFD. In: Parallel Computational Fluid Dynamics 1995, pp. 307–314. Elsevier (1996)
10. Ma, K.L.: In Situ Visualization at Extreme Scale: Challenges and Opportunities. IEEE Comput. Graph. Appl. **29**(6), 14–19(2009)
11. Parker, S.G., Johnson, C.R.: SCIRun: A Scienti_c Programming Environment for Computational Steering. In: Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing, pp. 52–52(1995).
12. Peterka, T., Bard, D., Bennett, J.C., Bethel, E.W., Old_eld, R.A., Pouchard, L., Sweeney, C., Wolf, M.: Priority Research Directions for In Situ Data Management: Enabling Scienti_c Discovery from Diverse Data Sources. Int. J. High Perfor. Comput. Appl. **34**(4), 409–427 (2020)

Acknowledgements

The idea to pursue this book was inspired by conversations at the Dagstuhl Seminar on “In Situ Visualization for Computational Science.” As such, we are indebted to *Schloss Dagstuhl Leibniz-Zentrum für Informatik* for supporting the seminar, the Dagstuhl staff for their assistance during the workshop, and all of the seminar participants. Furthermore, we thank everyone who participated in producing this book, including all of the chapter authors, and especially the lead authors of each chapter who provided insightful reviews.



Fig. 1 Participants of Dagstuhl Seminar 18271, “In Situ Visualization for Computational Science.”

Contents

In Situ Visualization for Computational Science: Background and Foundational Topics	1
Hank Childs, Janine C. Bennett, and Christoph Garth	
Data Reduction Techniques	
Sampling for Scientific Data Analysis and Reduction	11
Ayan Biswas, Soumya Dutta, Terece L. Turton, and James Ahrens	
In Situ Wavelet Compression on Supercomputers for Post Hoc Exploration	37
Shaomeng Li, John Clyne, and Hank Childs	
In Situ Statistical Distribution-Based Data Summarization and Visual Analysis	61
Soumya Dutta, Subhashis Hazarika, and Han-Wei Shen	
Exploratory Time-Dependent Flow Visualization via In Situ Extracted Lagrangian Representations	91
Sudhanshu Sane and Hank Childs	
Workflows and Scheduling	
Unlocking Large Scale Uncertainty Quantification with In Transit Iterative Statistics	113
Alejandro Ribés, Théophile Terraz, Yvan Fournier, Bertrand Iooss, and Bruno Raffin	
Decaf: Decoupled Dataflows for In Situ Workflows	137
Orcun Yıldız, Matthieu Dreher, and Tom Peterka	
Parameter Adaptation In Situ: Design Impacts and Trade-Offs	159
Steffen Frey, Valentin Bruder, Florian Friess, Patrick Gralka, Tobias Rau, Thomas Ertl, and Guido Reina	

Resource-Aware Optimal Scheduling of In Situ Analysis	183
Preeti Malakar, Venkatram Vishwanath, Christopher Knight, Todd Munson, and Michael E. Papka	

Tools

Leveraging Production Visualization Tools In Situ	205
Kenneth Moreland, Andrew C. Bauer, Berk Geveci, Patrick O'Leary, and Brad Whitlock	

The Adaptable IO System (ADIOS)	233
David Pugmire, Norbert Podhorszki, Scott Klasky, Matthew Wolf, James Kress, Mark Kim, Nicholas Thompson, Jeremy Logan, Ruonan Wang, Kshitij Mehta, Eric Suchyta, William Godoy, Jong Choi, George Ostrouchov, Lipeng Wan, Jieyang Chen, Berk Geveci Chuck Atkins, Caitlin Ross, Greg Eisenhauer, Junmin Gu, John Wu, Axel Huebl, and Seiji Tsutsumi	

Ascent: A Flyweight In Situ Library for Exascale Simulations	255
Matthew Larsen, Eric Brugger, Hank Childs, and Cyrus Harrison	

The SENSEI Generic <i>In Situ</i> Interface: Tool and Processing Portability at Scale	281
E. Wes Bethel, Burlen Loring, Utkarsh Ayachit, David Camp, Earl P. N. Duque, Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress, Patrick O'Leary, David Pugmire, Silvio Rizzi, David Thompson, Gunther H. Weber, Brad Whitlock, Matthew Wolf, and Kesheng Wu	

In Situ Solutions with CinemaScience	307
David H. Rogers, Soumya Dutta, Divya Banesh, Terece L. Turton, Ethan Stam, and James Ahrens	

New Research Results and Looking Forward

Deep Learning-Based Upscaling for In Situ Volume Visualization	331
Sebastian Weiss, Jun Han, Chaoli Wang, and Rüdiger Westermann	

Scalable CPU Ray Tracing for In Situ Visualization Using OSPRay	353
Will Usher, Jefferson Amstutz, Johannes Günther, Aaron Knoll, Gregory P. Johnson, Carson Brownlee, Alok Hota, Bruce Cherniak, Tim Rowley, Jim Jeffers, and Valerio Pascucci	

Multivariate Functional Approximation of Scientific Data	375
Tom Peterka, Youssef Nashed, Iulian Grindeanu, Vijay Mahadevan, Raine Yeh, and David Lenz	

A Simulation-Oblivious Data Transport Model for Flexible In Transit Visualization	399
Will Usher, Hyungman Park, Myoungkyu Lee, Paul Navrátil, Donald Fussell, and Valerio Pascucci	
Distributed Multi-tenant In Situ Analysis Using Galaxy	421
Greg Abram, Paul Navrátil, David Rogers, and James Ahrens	
Proximity Portability and <i>in Transit</i>, M-to-N Data Partitioning and Movement in SENSEI	439
E. Wes Bethel, Burlen Loring, Utkarsh Ayachit, Earl P. N. Duque, Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress, Patrick O'Leary, Dave Pugmire, Silvio Rizzi, David Thompson, Will Usher, Gunther H. Weber, Brad Whitlock, Matthew Wolf, and Kesheng Wu	

In Situ Visualization for Computational Science: Background and Foundational Topics



Hank Childs, Janine C. Bennett, and Christoph Garth

Abstract This chapter complements the Preface to this book. For more discussion on how to read this book, as well as information on the book itself, its purpose, and topics covered, we refer the reader to the Preface. Instead, this chapter provides background and an overview of foundational topics for in situ visualization for computational science. Section 1 provides a historical perspective of scientific visualization on supercomputers and why computing trends are increasingly mandating in situ processing. Section 2 presents an overview of in situ system types. The chapter concludes with Sect. 3, which highlights leading challenges and solutions for in situ processing. After reading this chapter, a beginner to this field should have sufficient context to read any of the subsequent chapters in this book, in any order.

1 The Motivation for In Situ Processing

This section is divided into two. The first subsection introduces relevant background material on computational simulations and post hoc processing. The second subsection describes how trends in high-performance computing are making post hoc processing difficult on supercomputers, and how in situ processing mitigates the resulting workflow challenges.

H. Childs (✉)
University of Oregon, Eugene, OR, USA
e-mail: hank@uoregon.edu

J. C. Bennett
Sandia National Laboratories, Livermore, CA, USA
e-mail: jcbenne@sandia.gov

C. Garth
Kaiserslautern Technische Universität, Kaiserslautern, Germany
e-mail: garth@cs.uni-kl.de

1.1 *Background: Computational Simulations and Post Hoc Processing*

Up until the recent surge of interest in in situ processing, nearly all scientific visualization for computational simulations was done with “post hoc” processing. With post hoc, the computational simulation sends its data to permanent storage and the visualization program retrieves this data from storage at some unknown time in the future, i.e., the data is not visualized as it is generated.

The amount of data generated by computational simulations can be very large. Simulations often represent a spatial domain using a computational mesh, consider multiple fields on that mesh (e.g., pressure, density, temperature, velocity), and store values for each field at each location on its mesh (e.g., on each vertex). The field values on the mesh change as the simulation advances, and in some cases the mesh itself changes as well. Typically, these advancements occur via “cycles,” where a simulation solves equations and then uses the solutions to update its field data (and possibly its mesh) for a subsequent moment in time. As a result, the data generated by a simulation is effectively a collection of “time slices” (also referred to as “time steps”), one for each cycle. Note that there are two types of time in this process—execution time, i.e., how long it takes for the hardware to carry out the desired computations, and simulation time, i.e., the time elapsed in the domain being simulated since the simulation began. For example, a combustion simulation may take five seconds to carry out the computations for a given cycle (execution time), but the result of that cycle may advance the simulation forward by only a few nanoseconds (simulation time). Both the execution time and simulation time vary from simulation to simulation (and can vary from cycle to cycle within a simulation), depending on factors such as the resolution of mesh, the quality of the mesh, the nature of the equations being solved, and the hardware being used. Typically, the execution times per cycle range from seconds to minutes, while the changes in simulation times depend on the application. Coming back to the topic of data size, the total data set generated by a simulation can be as many as quintillions of values (exabytes), as it generates values for the cross product of its time slices (typically ranging from thousands to hundreds of thousands), locations on its mesh (regularly in the hundreds of millions or billions and sometimes in the trillions), and fields (rarely less than ten and sometimes one hundred or more).

Saving all of a simulation’s data to permanent storage is often not practical, due to both the time it would take to transfer the data to storage and the amount of storage it would require. The most common approach for this problem is temporal subsampling, i.e., saving only some of a simulation’s time slices to disk. Fortunately, the change from cycle to cycle is often small, and so saving all of the cycles is rarely necessary. The mechanism for realizing temporal subsampling varies—some simulations save every N cycles, some save every time the simulation advances T units of time, some save every M minutes of execution time, and some save by studying phenomena occurring within the simulation and selecting key time slices.

In the post hoc setting, the simulation data is typically visualized using a stand-alone tool that is dedicated to visualization. The tool is normally directed by a domain scientist, who selects the data to visualize and how to visualize it. In response to the user's direction, the visualization tool will load the necessary data and produce the corresponding visualization. Importantly, the “human-in-the-loop” (HITL) nature of this model allows for interactivity, i.e., the domain scientist specifies desired visualizations, interprets the results, specifies new desired visualizations, and so on. The HITL model is particularly beneficial for exploratory use cases, i.e., when the domain scientist does not know the desired visualizations *a priori*. HITL is also useful for other types of use cases, like visual debugging, confirming phenomena within the simulation, and making visualizations to communicate results with a larger audience (e.g., “moviemaking”). However, some of these use cases can also be accomplished without a human-in-the-loop, as there often is *a priori* knowledge of what to visualize. In non-HITL scenarios, the visualization tools can be automated, to produce images, analyses, and movies that can be interpreted by domain scientists after the simulation is complete.

Especially for large data sets, post hoc visualization tools normally run on the same compute resources as the simulation. The reason for sharing compute resources is two-fold. First, using distinct resources requires data to be moved elsewhere, which can take a long time. Second, the amount of computations needed to carry out visualization tasks is often proportional to the amount of computations needed to generate the data in the first place. Using the same compute resources makes it very likely that the visualization tasks can be completed in a reasonable amount of time. This is particularly true in the context of supercomputers, which generate the very largest data sets. The same supercomputer that generates a data set is often subsequently used to visualize this data set. Finally, there are scenarios where distinct compute resources are used for post hoc visualization. In these scenarios, the data is typically small enough that it can be transferred in a reasonable amount of time, or the data is reduced to a smaller form so that it has this property.

1.2 High-Performance Computing Trends Increasingly Require In Situ Processing

Recent architectural trends are the primary driver behind the recent push for in situ processing. During the 2010s, the ability to compute (and thus generate data) increased on supercomputers much faster than the ability to store and load data. In broad terms, compute increased two orders of magnitude over this decade, while I/O performance only increased one order of magnitude. Some examples:

- Oak Ridge National Laboratory saw three generations of leading supercomputers—Jaguar (2009) to Titan (2012) to Summit (2018)—yield a 100X increase in computer power (from 1.75 petaFLOPS to more than 175 petaFLOPS), but only a 10X increase in filesystem performance (from 240 GB/s to 2.5TB/s) [1, 2].

- The Texas Advanced Computing Center saw a similar trend. In 2008, they introduced Ranger, the 5th fastest machine in the world, with a theoretical peak of 580 teraFLOPS and an I/O rate of 50 GB/s [3]. One decade later, Stampede2 appeared as the 12th fastest machine in the world, with a theoretical peak 30X greater (18 petaFLOPS) [4] but an I/O rate only 2X greater (100 GB/s) [5].

These examples are not outliers, but rather representative of trends observed community-wide.

This divergence between compute and I/O makes post hoc processing increasingly problematic for two reasons. First, visualization performance will likely be too slow. In 2010, before this divergence occurred, one study, focused on performance bottlenecks for very large data sets, showed visualization performance on supercomputers to already be I/O-bound, with as much as 98% of the execution time spent reading data from disk [6]. This problem has only worsened with each new generation of hardware. Second, the temporal resolution will likely not be sufficient. Simulation codes will not be able to afford to save time slices at the same rate, which can cause the data to become so temporally sparse that (1) important phenomena will be lost (i.e., a phenomenon begins after one time slice is saved and ends before the next save) or (2) features cannot be tracked over time.

One important development to mitigate I/O costs is the usage of “burst buffers.” The idea behind burst buffers is to create a location where computational simulations can stage their data. This enables simulations to continue advancing, since they no longer need to maintain the data for the current time slice. Concurrently, a separate program running on the burst buffer transfers the data to permanent storage. This has the benefit of hiding slow I/O costs from the computational simulation. However, it does not affect either of the factors identified in the previous paragraph—the burst buffer does not improve I/O performance for post hoc visualization nor does it enable temporal subsampling to be less sparse. That said, the burst buffer is noteworthy for in situ visualization, since it creates additional options for how computational simulations and in situ visualization routines can share data.

2 In Situ Systems

In situ visualization software is often referred to as a “system.” As is typically the case with computer “system” software, in situ systems need to optimize the use of constrained resources, and also provide services to other software (the computational simulation and possibly other software packages as well, including coordination between different in situ components).

In situ systems are diverse. A recent effort created a taxonomy of decisions for in situ systems [7], and we summarize the taxonomy here. The taxonomy looked at six independent factors, and enumerated the possibilities for each. Together, they help inform the space of possible in situ systems. The taxonomy’s six factors are:

- **Integration type.** Is the visualization code written specifically for the computational simulation? Does the computational simulation integrate external code? Does it need to be aware of this? Or can it be done through library interposition?
- **Proximity.** Do the visualization routines run on the same compute resources that the computational simulation uses? Do they run on nearby compute resources? (E.g., different compute nodes on the same supercomputer?)
- **Access.** Are the visualization routines in the same memory space as the simulation or not? If so, do they make a copy of the simulation's memory?
- **Division of Execution.** Do the visualization routines and computational simulation alternate use of the same resources (time division)? Or do they each have their own resources (space division)?
- **Operation Controls.** Is there a human-in-the-loop to steer the visualization? If so, does that block the simulation or not? If not, then can the visualizations performed be adapted in any way?
- **Output Type.** Does the system output a subset of the data? Does it make a smaller, transformed type?

3 Challenges and Solutions for In Situ Processing

In situ processing has advantages over post hoc processing, both in performance and quality of results. Notable performance advantages stem from the fact that in situ processing does not require I/O. Other possible performance advantages come from operating on data while it is still in cache and in using the larger compute resources available to the computational simulation. Significant quality advantages stem from the ability to access higher (or even full) temporal resolution data that may not be achievable with post hoc processing. Increased temporal resolution has many potential benefits, including improved results for feature tracking and discovery of phenomena, and increased accuracy for time-varying visualization techniques (like pathlines). Further, computational simulations sometimes generate data that they do not store at all, to save on storage costs. With in situ processing, this data can be accessed.

In situ processing also has disadvantages compared to post hoc processing. One disadvantage is the additional constraints imposed on the visualization algorithms when in situ routines run alongside the computational simulation. Depending on the specifics of the in situ system, in situ routines may need to minimize factors such as memory usage and execution time, and possibly even factors like network and energy usage. In situ systems also tend to be more complex. These systems connect with computational simulations via mechanisms like a communication protocol or via code linkage. These mechanisms, to date, have proven to be a bigger effort for developers than the post hoc approach of exchanging data through a file system. Fault tolerance also looms as a significant issue. In the post hoc world, there was no penalty to having visualization programs crash, as the data from the simulation was still available on the file system. In this setting, a bug in a visualization program could

be fixed, the program could be recompiled, and visualization could then occur. In an in situ setting, if the visualization program crashes, then the typical outcomes range from not being able to visualize the data to possibly even causing the simulation to crash.

In a related disadvantage, in situ places a burden on the visualization code to get the right visualizations while the data is available. If in situ processing generates one set of visualizations, and it is later determined that a different set of visualizations would have been more informative, then the simulation would need to be re-run, which can be very costly. Further, it can be complicated to control which time slices are visualized in situ. If an interesting phenomenon occurs, then it is not possible to go backwards in time and see what conditions led to the phenomenon, unless these time slices are saved in auxiliary memory or, again, if the simulation is re-run. A final issue is that of human-in-the-loop. It is possible to do in situ processing with HITL, and there have been multiple efforts to enable domain scientists to interact with their data as their simulations are running [8–11]. Despite these efforts, the large majority of in situ processing occurs with no HITL. One reason to avoid HITL is the associated computational costs. With some in situ approaches, it is possible for HITL to cause the entire compute resource to block, which is both expensive and wasteful. Another reason for avoiding HITL is that long-running computational simulations often run 24 h per day because the large cost of the machine promotes constant usage. Lastly, when there is no HITL and no a priori knowledge, then it is not clear what should be visualized. This challenge has inspired significant research, and is the focus of Part II of this book.

At the Dagstuhl seminar mentioned in the Preface, the participants identified the top ten research challenges. In the remainder of this section, we list the text of the challenges verbatim from the report [12]. Five of the ten challenges stem directly from the advantages and disadvantages listed above:

- Data quality and reduction, i.e., reducing data in situ and then exploring it post hoc to enable exploration of large data sets on future supercomputers.
- Workflow execution, i.e., how to efficiently execute specified workflows, including workflows that are very complex.
- Software complexity, heterogeneity, and user-facing issues, i.e., the challenges that prevent user adoption of in situ techniques because in situ software is complex, computational resources are complex, etc.
- Exascale systems, which will have billion-way concurrency and disks that are slow relative to their ability to generate data.
- Algorithmic challenges, i.e., algorithms will need to integrate into in situ ecosystems and still perform efficiently.

The remaining five challenges stem from new approaches for in situ to opportunities occurring as simulations become more complex or from developments in other fields:

- Workflow specification, i.e., how to specify the composition of different tools and applications to facilitate the in situ discovery process.

- Use cases beyond exploratory analysis, i.e., ensembles for uncertainty quantification and decision optimization, computational steering, incorporation of other data sources, etc.
- Exascale data, i.e., the data produced by simulations on exascale machines will, in many cases, be fundamentally different than that of previous machines.
- Cost models, which can be used to predict performance before executing an algorithm and thus be used to optimize performance overall.
- The convergence of HPC and Big Data for visualization and analysis, i.e., how can developments in one field, such as machine learning for Big Data, be used to accelerate techniques in the other?

Acknowledgements This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project 398122172. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This chapter describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the chapter do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

1. File Systems: Data Storage and Transfers. <https://www.olcf.ornl.gov/for-users/system-user-guides/summit/file-systems/>. Accessed: 2018-05-07
2. Bland, A.S., Kendall, R.A., Kothe, D.B., Rogers, J.H., Shipman, G.M.: Jaguar: the world's most powerful computer. In: Proceedings of Cray User's Group (CUG) Meetings, pp. 62–69 (2009)
3. Minyard, T.: Managing terascale systems and petascale data archives (2010). http://www.fujifilmsummit.com/wp-content/uploads/2017/05/2010-TACC_presentation.pdf
4. STAMPEDE2: TACC's Flagship Supercomputer. <https://www.tacc.utexas.edu/systems/stampeude2>. Accessed: 2018-05-07
5. Stockyard global file system. <https://www.tacc.utexas.edu/systems/stockyard>. Accessed: 2018-05-07
6. Childs, H., Pugmire, D., Ahern, S., Whitlock, B., Howison, M., Prabhat, Weber, G., Bethel, E.W.: Extreme scaling of production visualization software on diverse architectures. IEEE Comput. Graph. Appl. (CG&A) **30**(3), 22–31 (2010)
7. Childs, H., et al.: A terminology for in situ visualization and analysis systems. Int. J. High Perform. Comput. Appl. **34**(6), 676–691 (2020)
8. Bauer, A.C., Geveci, B., Schroeder, W.: The ParaView Catalyst User's Guide v2.0. Kitware, Inc. (2015)
9. Parker, S.G., Johnson, C.R.: Scirun: a scientific programming environment for computational steering. In: Supercomputing'95: Proceedings of the ACM/IEEE Conference on Supercomputing, p. 52 (1995)
10. Sanderson, A., Humphrey, A., Schmidt, J., Sisneros, R.: 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, pp. 201–214. Springer (2018)

11. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK, 2011. Proceedings, pp. 101–109 (2011)
12. Bennett, J.C., Childs, H., Garth, C., Hentschel, B.: In situ visualization for computational science (Dagstuhl Seminar 18271). *Dagstuhl Rep.* **8**(7), 1–43 (2018)

Data Reduction Techniques

Sampling for Scientific Data Analysis and Reduction



Ayan Biswas, Soumya Dutta, Terece L. Turton, and James Ahrens

Abstract With exascale supercomputers on the horizon, data-driven in situ data reduction is a very important topic that potentially enables post hoc data visualization, reconstruction, and exploration with the goal of minimal information loss. Sophisticated sampling methods provide a fast approximation to the data that can be used as a preview to the simulation output without the need for full data reconstruction. More detailed analysis can then be performed by reconstructing the sampled data set as necessary. Other data reduction methods such as compression techniques can still be used with the sampled outputs to achieve further data reduction. Sampling can be achieved in the spatial domain (which data locations are to be stored?) and/or temporal domain (which time steps to be stored?). Given a spatial location, data-driven sampling approaches take into account its local properties (such as scalar value, local smoothness etc.) and multivariate association among scalar values to determine the importance of a location. For temporal sampling, changes in the local and global properties across time steps are taken into account as importance criteria. In this chapter, spatial sampling approaches are discussed for univariate and multivariate data sets and their use for effective in situ data reduction is demonstrated.

1 Introduction

Conceptually, sampling can be loosely defined as the selection of a subset from a collection. In the context of a large-scale simulation code, sampling to decrease the output data size is reasonably common. Choosing a regular data output scheme of

A. Biswas (✉) · S. Dutta · T. L. Turton · J. Ahrens
Los Alamos National Lab, Los Alamos, NM, USA
e-mail: ayan@lanl.gov

S. Dutta
e-mail: sdutta@lanl.gov

T. L. Turton
e-mail: teturton@lanl.gov

J. Ahrens
e-mail: ahrens@lanl.gov

writing every n th time step is a ubiquitous approach to sampling. As mentioned in the motivating chapter of this book, when very large-scale scientific simulations are running on increasingly powerful supercomputers, one can only store a very small fraction of the generated data. Data movement and I/O restrictions will require more time between data dumps and naive time step selection as a data reduction technique risks losing important events that may occur between regularly scheduled output dumps.

Sampling techniques select m objects from the original pool of M elements with $m \leq M$ for an effective data reduction method. For data reduction and sampling for exascale codes, machine compute capability is orders of magnitude higher than I/O rates, and this translates to $m \ll M$, i.e., m is orders of magnitude smaller than M . Thus, for in situ large-scale data reduction purposes, simple sampling approaches are no longer sufficient to address the data reduction needs. Adaptive and data-driven sampling techniques become necessary to ensure that the data saved to disk has the relevant statistical properties of the original data and that the information saved prioritizes rare and/or important features and events in the data.

Sampling for data reduction is an attractive approach for in situ data reduction for various reasons. Sampling allows the users to select a representative subset of the raw data points where the true location and the data values for the sampled points are preserved. Downsampled data statistically represents the original data, preserving features and relationships in the data. Use of samples in the post hoc exploration phase does not always require expensive data reconstruction as the reduced data set can be visualized/queried to get an overview of the data.

With these benefits in mind, sampling becomes an attractive in situ solution. Similar to the other in situ data reduction methods discussed in this book, the main goal of sampling is preservation of important data properties or *features* given the limited I/O bandwidth and storage capabilities. In this chapter, scalar and multivariate data sets are addressed. The concept of *importance* is introduced with a discussion of how to define it via an automated data-driven approach to demonstrate how sampling can be a useful tool for preserving the *important* regions. Section 2 briefly discusses the foundations related to in situ sampling and visualization. Section 3 describes different scalar sampling methods for scientific data sets. Section 4 focuses on the sampling methods for multivariate data sets. In situ performance is discussed in Sect. 5, with limitations of the sampling methods are covered in Sect. 6. Future directions for in situ data-driven sampling are included in Sect. 7.

2 Prior Work

The development and deployment of novel sampling methods for in situ data reduction and feature preservation is built upon work across multiple disciplines. A flexible many-core capable infrastructure is critical for modern high performance computing simulation codes to access in situ analysis such as sampling. Bauer et al. [9] provide an overview of in situ infrastructures available to simulation codes for deploy-

ment of analysis routines. As high performance computing heads towards exascale, there are several commonly used infrastructures for scientific applications that can be leveraged to enable *in situ* sampling. These include ParaView/Catalyst [1, 6], VisIt/LibSim [14], Ascent [27], SENSEI [7], and ADIOS [32]. Each of these *in situ* APIs provides the visualization and analysis support necessary for *in situ* sampling use cases and details about many of these infrastructures can be found elsewhere within this book.

Sampling is one of many possible data reduction techniques available to address concurrency challenges at exascale. Son et al. [38] provide a survey of data compression techniques for exascale computing. Lossless compressors such as BLOSC [5] and FPZIP [30] are attractive to avoid introducing biases into the data. However, lossless compression typically does not provide sufficient data reduction to fully solve I/O bandwidth and storage issues. Modern compressors usually allow both lossless and lossy modes, allowing the user to set parameters such as error bounding. Transform methods, such as ZFP [29], SPECK [24] or wavelet approaches such as Li et al. [28], model the data, retaining the most important coefficients in the modelling scheme. Compressors based on predictive algorithms include SZ [17, 42] and FPZIP [30].

Another common data reduction technique is to move data analysis, visualization, or feature detection *in situ*, writing out reduced data extracts rather than full raw data sets. Image-based data abstracts have been shown to provide post hoc data analysis flexibility [2, 43] (see, for example, the chapter on Cinema in this book). Reduction through statistical summarization is another approach [23, 46, 50]. Sampling as a data reduction technique is similar to non-compression data reduction approaches. That said, sampling techniques can often be used in combination with compression to achieve higher levels of overall data reduction.

Information theory [16, 37, 45] forms a theoretical foundation used across many aspects of analysis and visualization of large scientific data sets [13]. Drawing from information theory, Nouanesengsy et al. [34] developed ADR, Analysis-Driven Refinement, in which user-defined importance metrics are used to select a sparse data set for fast post hoc analysis and visualization. Entropy maximization was used by Biswas et al. [12] in the development of an adaptive *in situ* sampling methodology that prioritized rare but important events in the data. Dutta et al. [18] developed a multivariate approach for statistical data summarization and downsampling based on pointwise information theoretic measures.

Woodring et al. [49] introduced and demonstrated a stratified random sampling scheme in a cosmology code. Sampling based on bitmap indexing was first used by Su et al. [40]. Wei et al. [48] proposed IGStS, *information guided stratified sampling*, to downsample data sets while preserving important regions in the data. Park et al. [35] advocated for a visualization-aware sampling approach that minimized a loss function based on visualization goals. Sampling in the context of visualization also led Battle et al. [8] to develop ScalaR which performs data reduction on the fly for database queries returning results too large for effective interactive visualization. Likewise, with *sampleAction*, Fisher et al. [20] focused on improving the visualization experience by visualizing incremental results. As described in this chapter,

data-driven sampling techniques draw heavily on importance metrics to develop sampling algorithms that prioritize the data of interest to the domain scientist.

3 Sampling Using Scalar Data Importance

3.1 Motivation for Generic Scalar Sampling

Scalar fields are commonly found across scientific domains and simulations. In the simplest case, a 2D or 3D spatial domain is discretized into smaller regular sized regions and a scalar quantity, e.g., *temperature* T, is computed at each discrete location. Although irregular grid discretizations are possible, this chapter will focus on regular grid discretizations only. After the scalar field *temperature* is produced by the simulation, domain scientists use that data to understand the features and phenomenon of interest, e.g., temperature can be used to explore and map the critical global currents that drive global warming and climate change.

A common workflow is for the domain scientist to analyze the features of interest in the data and develop data reduction methods that are specific to the scalar field and features of interest. When the scientist needs to perform a similar feature-driven analysis of a different variable, for example, *pressure* P, the analysis workflow may need to incorporate a different algorithm tuned to this new variable of interest. From the perspective of optimizing the time and effort of the domain scientist, this can be an inefficient workflow.

An alternative approach is to use more generic scalar field sampling methods. This approach is particularly useful in the in situ scenario as the same algorithm can potentially be used across multiple scalar fields during a run, significantly reducing overall I/O and storage needs. An overview of two generic scalar field sampling approaches is first presented before moving on to more sophisticated data-driven sampling methods.

3.2 Methods for Scalar Field Sampling

3.2.1 Random Sampling

Random sampling methods are popular due to their simplicity and have been heavily used for various sampling purposes. In a simple random sampling method, each data value has equal probability of being selected. For stratified random sampling, the data is first divided into strata (groups) and then random sampling is applied from within each strata. A schematic example of random sampling can be seen in Fig. 1.

Neither simple nor stratified random sampling take into account domain science knowledge or scientific goals. Likewise, neither method assigns priority to any

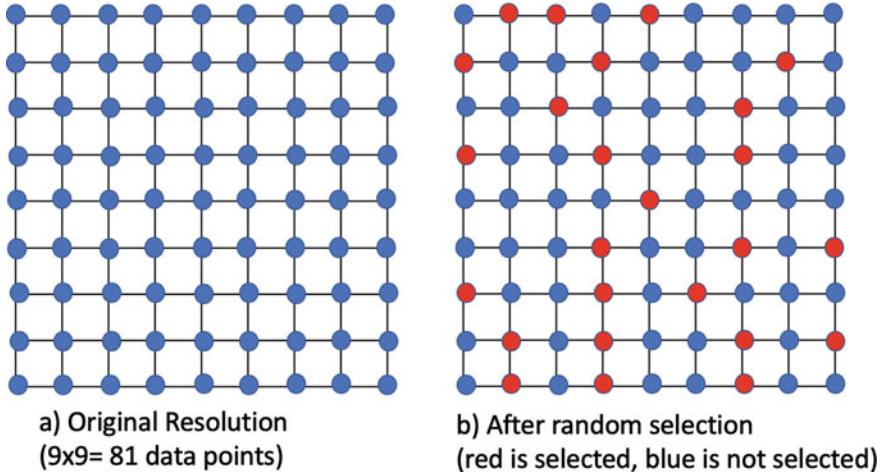


Fig. 1 Schematic example showing random sampling for regular grid data. **a** Original data **b** after randomly selecting 25 out of 81 data points

specific data values that might be of particular importance to the scientist. Hence, although these methods preserve the overall data distribution, important features of the data are not explicitly retained and some or all of these features may be unintentionally lost in the sampling process.

For a regular grid data set, random sampling will produce a particle data set as a result of the sampling. If N_{rand} samples are to be generated, then 4-tuples $\{x, y, z, val\}$ for each location will need to be stored where x, y, z represent the locations of the particles and v is some variable specific to the simulation. This amounts to $4N$ storage.

3.2.2 Regular Sampling

Another well known and commonly used sampling method is naive regular sampling. This method sub-samples the data by regularly selecting data points using a pre-defined interval. A schematic example is shown in Fig. 2. Similar to random sampling, this method also preserves the overall data distribution and yields similar statistics as the original data. However, again, there is no notion of feature preservation in the data generated by the sub-sampling scheme. After sampling, regular sampling keeps the regular grid structure. Thus the sampled data output is still on a regular grid and requires N_{reg} storage when N_{reg} samples are to be stored as only the values are stored at each grid point.

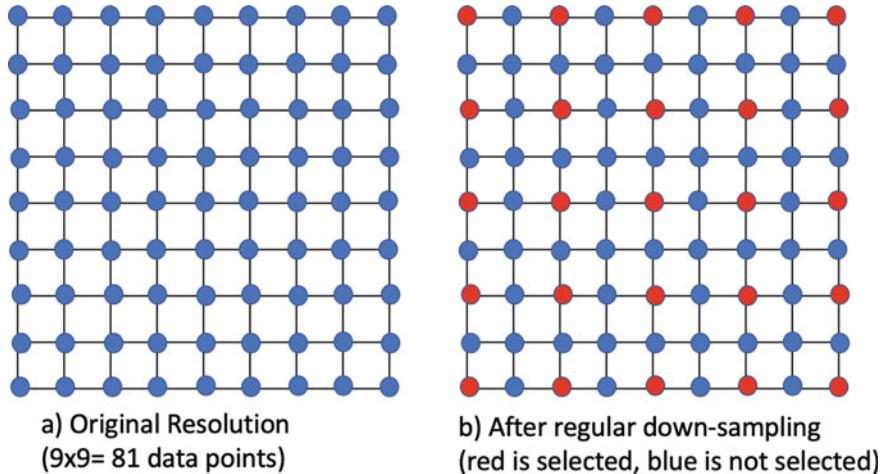


Fig. 2 Schematic example showing regular sampling for regular grid data. **a** Original data **b** after regular selection of 25 out of 81 data points

3.2.3 Sophisticated Data-Driven Sampling Approach

Sampling approaches that are based on random and regular selection have a limitation when used for scientific data analysis and reduction. Although the samples generated from such methods provide a good approximation to the original data distribution, all the data points are given equal *importance* when making the selection. This is not always the ideal way to choose a subset of values when dealing with data from scientific simulations. Generally, such simulations contain *features of interest* that are more important to the domain experts compared to the other parts of the data. Another key point is that these features are potentially much less probable in the data when compared to the chance of occurrence for non-feature regions. Based on these ideas, an *in situ* algorithm can be devised that can still be generic (i.e., applicable to a diverse set of simulations and scalar fields), but that prioritizes the important feature regions of the data while sampling.

Information theory provides methods applicable to developing data-driven sampling approaches that preferentially save data of interest to the domain scientist. Using the guiding principle that rare values in a data set are more likely to be a feature and the more abundant data values are likely to be background or non-feature regions, a data-driven sampling method can be formulated by assigning more importance to a data value that has lower probability of occurrence, and assigning lower importance to the samples that are more abundant. This simple procedure will likely retain the important regions of the data. There are many different ways this can be implemented.

Rather than introducing different user-defined parameters to the system for *in situ* use, the concept of Shannon's Entropy from information theory can be used

to formulate a fast and effective algorithm. Shannon's entropy, H , provides the total information content [16] of a random variable X . This is given as: $H(X) = -\sum_{x \in X} P(x) \log_2 P(x)$, where $P(x)$ is the probability of x , where $x \in X$.

The principle of maximum entropy can be used for creating generic sampling methods. The principle of maximum entropy states that the maximum entropy state of a random variable is the best representation when no other information is available [26]. Using this for a generic sampling method without knowing the features of the data, the best samples would be the ones that maximize the output entropy. Since a uniform distribution has maximum entropy, the goal of maximum entropy sampling will be to try to select equal number of samples for each scalar value irrespective of its abundance in the data set. Ideally, if selecting C samples for all scalars, then scalars that occur more frequently in the data will automatically get a lower probability of acceptance. Similarly, rare scalar values will have higher chance of selection.

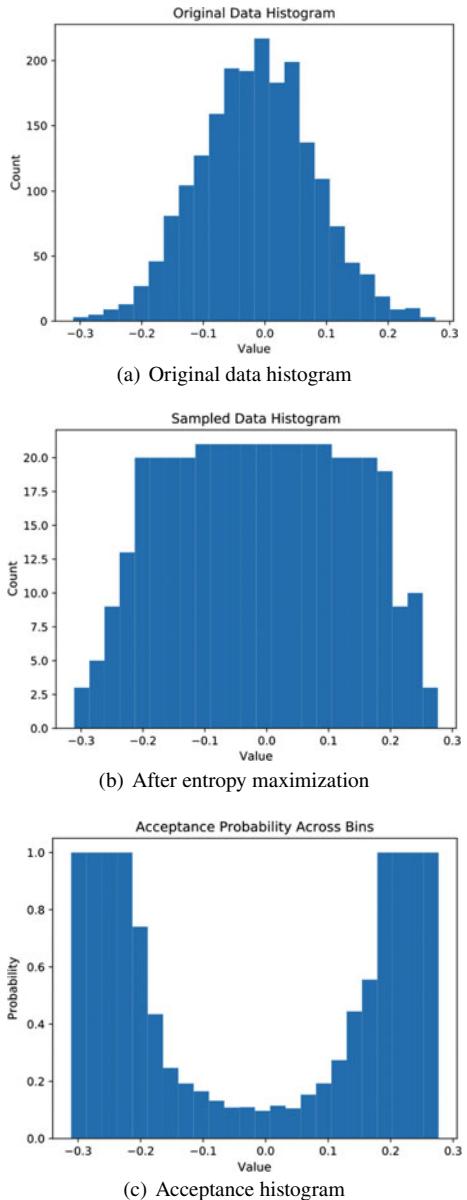
Figure 3 illustrates this concept. From a Gaussian distribution with the mean, $\mu = 0.0$, and the variance, $\sigma^2 = 0.01$, 2000 random data points are taken. The data histogram is shown in Fig. 3a. Computing entropy for this data set using 24 bins results in 3.9 bits. Selecting 20% samples from this data using the entropy-maximizing sampling method as discussed before, results in a histogram similar to Fig. 3b with an entropy of 4.45 bits. This method of sampling maximizes the entropy from these samples. An *acceptance histogram* can now be defined as a histogram that plots the data values along the x-axis while the y-axis denotes the probability of getting selected for that scalar. This can be seen in Fig. 3c. From this figure, it is easy to observe that the values with higher frequency in the original data were given lower importance and vice versa. This algorithm is summarized in Algorithm 1.

An application of this data-driven intelligent sampling method to the Hurricane Isabel data set [47] is shown in Fig. 4. For this data set, the feature of interest is the hurricane eye as shown in Fig. 4a. For demonstration purposes, the Pressure field from time step 25 is chosen. The effect of applying the three different scalar sampling methods (random, regular, and data-driven entropy-maximizing sampling) on this data can be seen. The sampling rates used were 0.5% for random and entropy-based data-driven intelligent sampling method (since they store the point locations) and 1.5% for regular sampling (as it does not need to store the locations). All the sampled data outputs are reconstructed back to original size for comparison purposes. As can be seen from Fig. 4, the data-driven entropy-maximizing method (as shown in Fig. 4c) outperforms the random (Fig. 4d) and regular (Fig. 4e) sampling methods in retaining a more complete visual representation of the hurricane eye.

3.3 Sample Analysis and Reconstruction

Often, scientists want to visualize the full-resolution data when exploring important features interactively. Consequently, using sub-sampled data requires a technique that can reconstruct the full-resolution data from the sampled data points. The reconstruction of the full-resolution data can be performed using a nearest-neighbor or linear

Fig. 3 Illustration of information-driven sampling method via entropy maximization. **a** Histogram of original data points, **b** histogram of the sampled data where the entropy of resulting histogram is maximized. **c** Acceptance histogram showing the probability of the values of each bin getting accepted after sampling. This image is reprinted from our previous work [12]



```

Input: Sim generated data at each time step
Output: Acceptance histogram
 $n_k$  = samples to be taken from full data
 $N$  = total points from full data
nbins = number of bins
 $n_{samps} = n_k \div nbins$  : samples to be taken from each bin
remaining-samples =  $N$ 
H = CreateHistogram(Data,nbins)
count,bin-edges = SortBinsAccordingToTheirCount(H)
while not all bins are visited do
    i=0;
    if count[i] <  $n_{samps}$  then
        | samples-taken = count[i];
    else
        | samples-taken =  $n_{samps}$ ;
    end
     $P_i$  = samples-taken  $\div$  count[i]
    remaining-samples = remaining-samples - samples-taken
    remaining-bins = nbins - i
     $n_{samps}$  = remaining-samples  $\div$  remaining-bins
    i=i+1;
end

```

Use P_i s as the probabilities for the corresponding bins of the acceptance histogram

Algorithm 1: Algorithm for the in situ generation of an acceptance histogram.

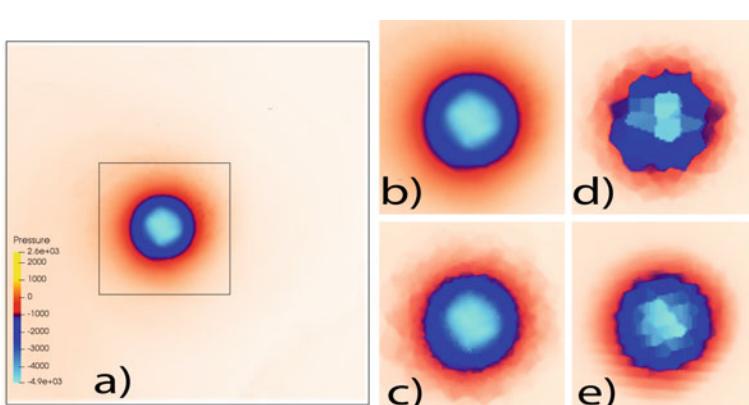


Fig. 4 Sampling results from Hurricane Isabel data set. **a** Original data **b** zoomed in view of the feature (hurricane eye) region. **c** Reconstruction using data-driven sampling method (sampling ratio 0.5%). **d** Reconstruction using random sampling method (sampling ratio 0.5%). **e** Reconstruction using regular sampling method (sampling ratio 1.5%). This image is reprinted from our previous work [12]

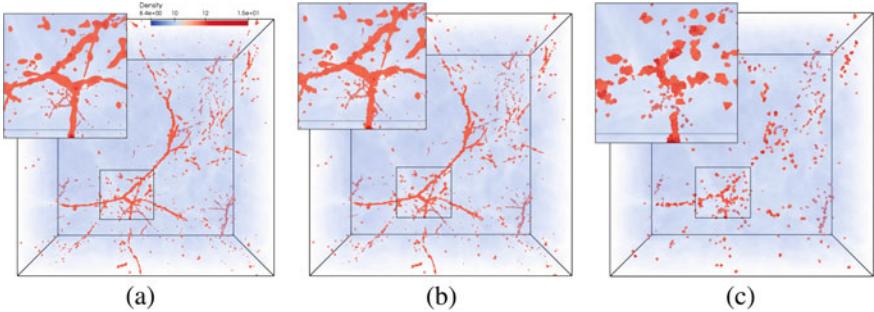


Fig. 5 Volume visualization of sampling results (sampling ratio 1%) using the Nyx simulation. **a** Original data and zoomed into a feature region with a dark matter halo. **b** Reconstruction using data-driven intelligent sampling method. **c** Reconstruction using stratified random sampling method. This image is reprinted from our previous work [12]

interpolation based technique. Nearest-neighbor interpolation is faster but generally less accurate. Given a location where a value is to be assigned in the reconstruction phase, this method assigns the value of the nearest sample location. Linear interpolation is more time-consuming but often more accurate. In this method, first a 3D convex hull is created using the sampled points and then a polygonal mesh is created using Delaunay triangulation. Then, for each grid point in the reconstruction grid, the simplices in the Delaunay mesh are used and the value is linearly interpolated from the simplex vertices that encloses the current grid point. Once the data is reconstructed (using nearest-neighbor or linear), all analysis/visualization techniques (including the ray casting-based volume rendering that is shown here) can be used to explore the full-resolution data.

3.4 In Situ Analysis and Quality Comparison

To demonstrate the performance of the three sampling methods, the Nyx cosmology simulation code [4] is used. The Nyx data resolution was $512 \times 512 \times 512$. The Nyx simulation produces a regular grid data set with multiple variables. Out of these, the baryon density is one of the most important variables to analyze because it provides the particle concentration information tracing back to the origin of universe. In this density field, high density regions correspond to dark matter *halos* forming over time in the universe. These high density regions are the most important to the scientist. In Fig. 5, visualizations of the reconstructed samples are shown for comparison purposes at 1% sampling rate for random and data-driven methods. Figure 6 visualizes the samples coming out of the sampling methods with even lower sampling ratios: 0.5% sampling rate for random and data-driven methods and 1.5% for the regular method. From both of these figures, it can be observed that the data-driven method preserves features within the data much better than the regular and random methods.

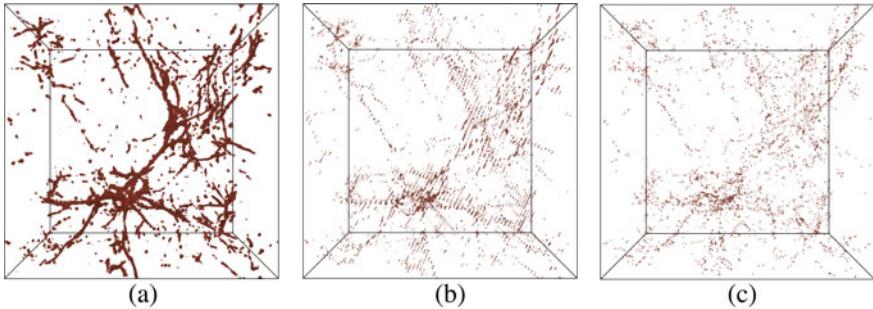


Fig. 6 Point rendering results from Nyx simulation. **a** Using the data-driven intelligent sampling method (sampling ratio 0.5%), **b** using the regular sampling method (sampling ratio 1.5%), **c** using the stratified random sampling method (sampling ratio 0.5%). This image is reprinted from our previous work [12]

Table 1 Quantitative similarity comparison of sampling method-produced images to the image produced by the original data under same rendering configuration

	Data-driven sampling	Random sampling	Regular sampling
Hurricane Isabel data	0.978	0.921	0.961
Nyx cosmology data	0.987	0.865	0.881

For a quantitative comparison of the quality across these methods, Pearson's correlation coefficient is calculated for both the Hurricane Isabel and Nyx data sets. This calculation is performed using volume rendered images from the original data and from the three sampling methods for the same camera angle and transfer function. Then using red, blue and green as three channels, the Pearson correlation value for each sampling method is calculated against the original data. This result is presented in Table 1. As can be seen from this Table, the data-driven intelligent sampling method clearly outperforms the other two methods.

Thus, it can be seen that intelligent scalar sampling methods can provide high quality and light-weight data reduction capabilities for scalar fields in large-scale data sets. The next section discusses the case where a simulation produces multiple variables to see how information theory can motivate effective multivariate sampling approaches.

4 Sampling Using Multivariate Association

4.1 Motivation for Multivariate Sampling

Large-scale simulations commonly produce data sets containing multiple variables. The above section described sampling techniques that work on a single variable at a time while sub-sampling the data. However, data analysis applications may require analysis of multiple variables together. Hence, similar to the univariate sampling technique, multivariate data sampling techniques are also important. Multivariate sampling can help in reducing a set of simulation variables together. For multivariate data sets, sub-sampled data must preserve the relationship and correlations between variables so that when a post hoc analysis is conducted using the sub-sampled data, the multivariate data features will be the same as an analysis done on the full data set.

Several previous studies have shown that the relationship among multiple variables can be complicated [11, 25, 31]. To summarize such variables together, first, one needs to identify the relationship among them. An effective way of performing sub-sampling of multivariate data would be to sample the points with higher fidelity from the region where the variables show strong statistical association among them. Previous studies have shown that such statistically associated regions often contain multivariate features that are of interest to the application experts for detailed analysis. For example, in a hurricane simulation, the hurricane eye is an important feature, which can be characterized by a spatial region with low pressure and high-velocity values [22, 23]. Similarly, to understand the complex turbulent mixing in a combustion simulation, the study of high valued hydroxyl regions together with the stoichiometric mixture fraction variable reveals more information than studying them individually [3]. Therefore, a multivariate data sub-sampling technique that preserves the statistical association among variables will be able to analyze and visualize such features with high accuracy in the post hoc analysis phase. The following Sections describe a multivariate data sampling algorithm that uses statistical data associations and multivariate distributions to sub-sample several data variables together and demonstrates the usefulness of the technique by providing various multivariate applications with qualitative and quantitative studies.

4.2 Multivariate Statistical Association-Driven Sampling

One of the primary requirements of a multivariate sampling technique algorithm is to preserve the multivariate properties, i.e., the interdependence among the chosen variables, and their correlation properties so that the sub-sampled data set can be used effectively for multivariate feature analysis. To achieve this, the multivariate sampling algorithm selects samples densely from the regions where the variable combinations show a strong statistical association or co-occurrence. Higher co-occurrence

indicates a stronger association among data values. The strength of the association is quantified for each spatial data point considering their value combination from multiple variables. After the quantification of statistical association is done, sub-sampling is performed according to the strength of the association values. Data points which show stronger statistical association have a higher chance of getting selected. At the end of the sampling process, an unstructured data set is produced and this reduced data can be stored onto disk for post hoc analysis. Note that, based on the storage budget, the application user can determine the percentage of data points that will be stored for post hoc analysis and thus determine the amount of data reduction that will be achieved through the sampling.

4.2.1 Estimation of Multivariate Pointwise Information

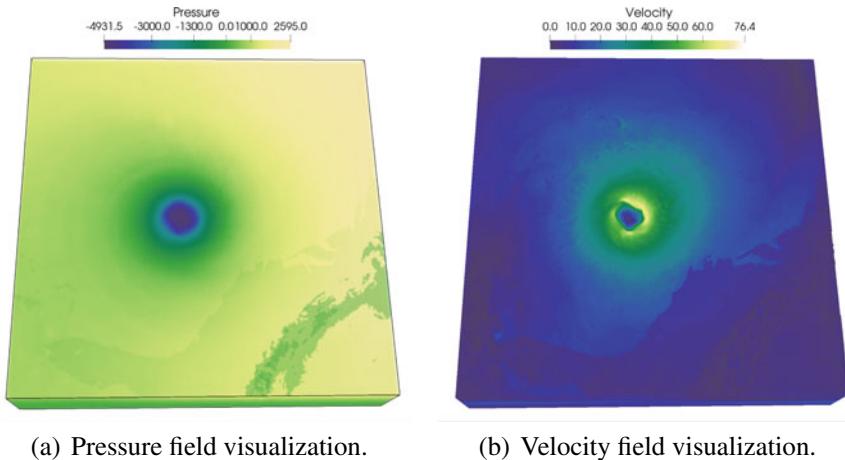
Multivariate data sampling first requires a criterion to quantify the importance of each data point in the multivariate sense. Each data point in the multivariate data has a value tuple consisting of values from each variable. Consider two variables, X and Y . At each data point, there is a value pair (x, y) where x is a specific value of variable X and y a value for Y . Next, a measure is needed that can quantify the importance of each such value pair so as to select data points that are more informative than others in order to perform sub-sampling. Using information theory, the importance of such value pairs can be estimated using an information-theoretic measure called pointwise mutual information (PMI). Pointwise mutual information was first introduced in the works of Church and Hanks [15] for the quantification of word association. PMI measures the strength of the statistical association of each value pair, thus for each data point. For two random variables X and Y , the PMI value for the value pair (x, y) can be formally defined as:

$$PMI(x, y) = \log \frac{p(x, y)}{p(x)p(y)} \quad (1)$$

Here, $p(x)$ is the probability of a particular occurrence x of X , $p(y)$ is the probability of y of variable Y and, $p(x, y)$ is their joint probability. When $p(x, y) > p(x)p(y)$, $PMI(x, y) > 0$, which means x and y have higher statistical association between them. When $p(x, y) < p(x)p(y)$, then $PMI(x, y) < 0$. This condition indicates that the two observations follow a complementary distribution. Finally, when $p(x, y) \approx p(x)p(y)$, then $PMI(x, y) \approx 0$ refers to the case where the variables are statistically independent. It is important to note that the mutual information (MI) $I(X; Y)$ is the expected PMI value over all possible instances of variables X and Y [44].

$$I(X; Y) = E_{(X,Y)}[PMI(x, y)] \quad (2)$$

Given this information-theoretic measure, if a value pair has higher PMI value, then it is more likely that the data point associated with it will be selected in the sub-sampled data set. Previous work has shown that regions with high PMI values



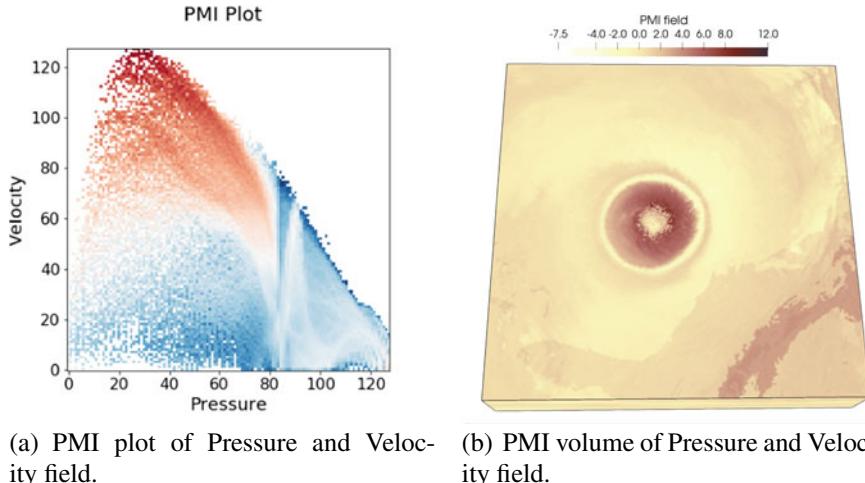
(a) Pressure field visualization.

(b) Velocity field visualization.

Fig. 7 Visualization of the Pressure and Velocity fields of Hurricane Isabel data set. The hurricane eye at the center of Pressure field and the high velocity region around the hurricane eye can be observed. This image is reprinted from our previous work [18]

often correspond to the multivariate features in the data set [19], and hence, selecting data points using their PMI values will also ensure the preservation of important multivariate features in the data set.

To demonstrate the concept of PMI, first consider two variables: Pressure and Velocity from the Hurricane Isabel data set. Figure 7 shows the volume visualization of these two variables. To estimate the PMI values for each data point for these two variables, one first estimates their probability distributions in the form of normalized histograms. Then for each value pair in the 2D histogram bin, the PMI value is estimated using Eq. 1. A 2D PMI plot is shown in Fig. 8a. The X-axis of the plot shows values of Pressure and the Y-axis shows values of Velocity. Since the computation of PMI values were done using a histogram, the axes in plot Fig. 8a show the bin IDs. The corresponding scalar value for each bin ID can be easily estimated from the range of data values for each variable. In this plot, the white regions in the plot represent value pairs with zero PMI values and red regions indicate high PMI values. It can be observed that the low Pressure and moderate to high Velocity values have high PMI values and therefore higher statistical association. Using the PMI values for each data point, a new scalar field can be constructed, namely the PMI field where each data point reflects the pointwise statistical association. In Fig. 8b, the PMI field for the Pressure and Velocity variable is shown. It can be seen that the darker brown region has higher values of PMI and is highlighting the eye-wall of the hurricane, an important multivariate feature in the data set. Hence, sampling using PMI values will select more data points from the eye-wall region since the data points there have higher PMI values and, by doing so, the eye-wall feature will be preserved with higher detail in the sub-sampled data set.



(a) PMI plot of Pressure and Velocity field.

(b) PMI volume of Pressure and Velocity field.

Fig. 8 PMI computed from the Pressure and Velocity fields of the Hurricane Isabel data set is visualized. Figure **a** shows the 2D plot of PMI values for all value pairs of Pressure and Velocity, Fig. **b** provides the PMI field for analyzing the PMI values in the spatial domain. It can be seen that around the hurricane eye, the eye-wall is highlighted as high PMI-valued region which indicates a joint feature in the data set involving Pressure and Velocity field. This image is reprinted from our previous work [18]

Generalized Pointwise Information. The pointwise mutual information measure allows us to analyze two variables at a time. To estimate the PMI values for more than two variables, a generalized information theoretic measure can be used, called specific correlation [44]. Specific correlation can be formally defined as:

$$SI(x_1, x_2, \dots, x_n) = \log \frac{p(x_1, x_2, \dots, x_n)}{p(x_1)p(x_2) \dots p(x_n)} \quad (3)$$

where $p(x_i)$ represents the probability of an observation x_i for the i th variable X_i , and $p(x_1, x_2, \dots, x_n)$ refers to the joint probability of the value tuple (x_1, x_2, \dots, x_n) . Note that, specific correlation is a generalized extension of the pointwise mutual information and can be used to quantify association for data points when more than two variables are used.

The pointwise information measures presented above depend on joint distributions of variables, raising the question of how these high-dimensional distributions can be computed effectively. Joint probability distributions can be computed using various modeling approaches, such as parametric distributions, non-parametric distributions, etc. Among parametric distribution models, Gaussian mixture model (GMM) is well known for its compactness as only model parameters need to be stored during its calculation. However, the estimation of parameters for high-dimensional Gaussian mixture models using an Expectation-Maximization (EM) technique [10] can be computationally expensive. In contrast, non-parametric models such as Kernel Den-

sity Estimation (KDE) and histograms are other alternatives that can be used to estimate joint probability distributions. The computation of high-dimensional KDE is expensive since a significant number of kernel evaluations are required. Furthermore, the storage increases as dimensionality increases. In contrast, the computation of histogram-based distributions is comparatively faster, but standard high-dimensional histogram representations are not storage efficient.

For a large number of variables, in general, all the above approaches suffer from the curse of dimensionality. However, sparse histogram representations reduce storage footprint significantly. Recently, to address the issues of dimensionality, a new technique for high-dimensional histogram estimation has been proposed [33]. This new technique is storage efficient and can be computed efficiently in a distributed parallel environment. Another effective and alternative way of modeling high-dimensional distributions is the use of statistical Copula functions [23]. A Copula-based distribution representation only stores individual independent distributions and the correlation information among the modeled variables. From this Copula model, the joint probability can be queried. This approach reduces the computational cost and storage cost significantly while estimating high-dimensional distributions [23]. It is also important to note that, in practice, multivariate features are mostly defined using two to three variables in combination, and hence sparse histograms can be used to estimate the joint distribution for the PMI calculation. If higher-dimensional distributions are required, other techniques [23, 33] can be used.

4.2.2 Pointwise Information-Driven Multivariate Sampling

This section covers the multivariate sampling process that uses the aforementioned pointwise information measures as the multivariate sampling criterion. The sampling method accepts data points with a higher likelihood if they have higher values of pointwise information. Therefore, regions with higher PMI values (such as the hurricane eye-wall region as seen in Fig. 8b will be sampled densely compared to regions with relatively low PMI values in the final sub-sampled data set.

The multivariate sampling method accepts a user-specified sampling fraction (α , where $0 < \alpha < 1$) as an input parameter and produces a sub-sampled data set with $n = \alpha \times N$, ($n < N$) data points where N represents the total number of data points. First, a joint histogram is constructed using all the variables that will be used for sampling. Note that, in the bi-variate case, this results in a 2D histogram. Each histogram bin in this joint histogram represents a value pair and a PMI value can be estimated for each histogram bin. Therefore, the normalized PMI values corresponding to each histogram bin can be used as a sampling fraction for that bin and the bins with higher PMI values will contribute more to the sample selection process.

For example, if a histogram bin has a normalized PMI value of 0.8, then 80% of the data points from this bin will be selected. Note that selecting sample points in this way ensures that the higher PMI valued data points are prioritized in the sampling process and, by doing so, data points with a stronger statistical association are preserved in the sub-sampled data. The interested reader can find more details of

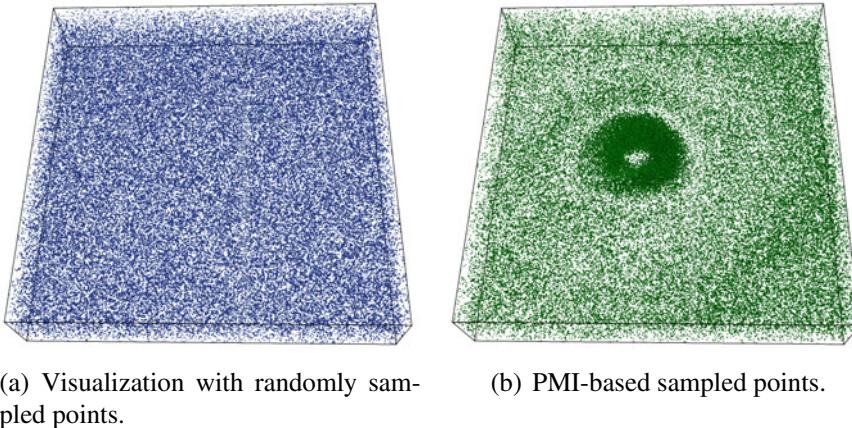


Fig. 9 Sampling result in Hurricane Isabel data set when Pressure and Velocity variables are used. Figure **a** shows results of random sampling and Fig. **b** shows results of the PMI-based sampling results for sampling fraction 0.03. By observing the PMI field presented in Fig. **b**, it can be seen that PMI-based sampling method samples densely from the regions where the statistical association between Pressure and Velocity is stronger (Fig. **b**). This image is reprinted from our previous work [18]

this sampling method in [18]. An example of this multivariate sampling is shown in Fig. 9. Figure 9b shows the sub-sampled data points for the Hurricane Isabel data set when Pressure and Velocity fields are used. It can be observed that data points from the eye-wall region were selected densely since the data points in that region have high PMI values. In Fig. 9a, a randomly sub-sampled field is shown. By comparing Fig. 9b with Fig. 9a, it is evident that PMI-based sampling preserves the statistically associated regions in the sub-sampled data set and accurate post hoc multivariate feature analysis using such a data set can be performed.

4.3 Applications of Multivariate Sampling

4.3.1 Sample-Based Multivariate Query-Driven Visual Analysis

Query-driven visualization (QDV) is a popular technique for analyzing multivariate features in a scientific data set [21, 23, 39]. Query-driven analysis helps the expert to focus quickly on the region of interest, effectively reducing their workload. In this example, the sub-sampled data set is used directly for answering domain specific queries using an asteroid impact data set. The Deep Water Asteroid Impact data set [36] was generated at the Los Alamos National Laboratory to study Asteroid Generated Tsunami (AGT). The data set contains multiple variables. The volume fraction of water variable, denoted by v_{02} , and the temperature variable denoted

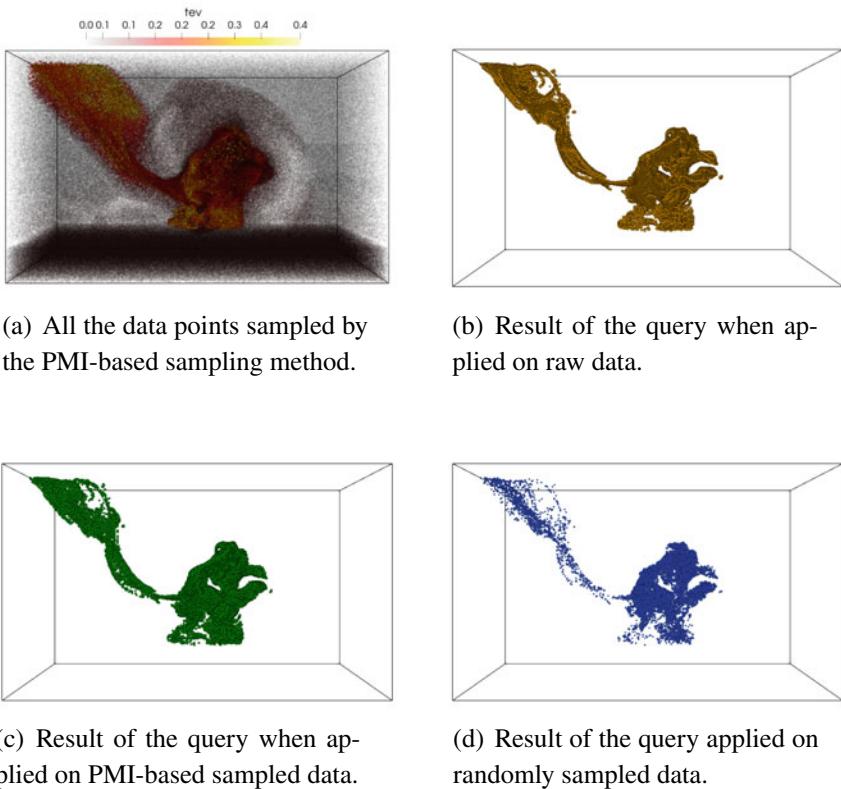


Fig. 10 Visualization of multivariate query driven analysis performed on the sampled data using the Asteroid impact data set. The multivariate query $0.13 < tev < 0.5 \text{ AND } 0.45 < v02 < 1.0$ is applied on the sampled data sets. Figure **a** shows all the points selected by the PMI-based sampling algorithm by using tev and $v02$ variable. Figure **b** shows the data points selected by the query when applied to raw data. Figure **c** shows the points selected when the query is performed on the sub-sampled data produced by the PMI-based sampling scheme and Fig. **d** presents the result of the query when applied to a randomly sampled data set. The sampling fraction used in this experiment is 0.07. This image is reprinted from our previous work [18]

by tev , are used in this study. First the data set is sub-sampled using the above association-driven multivariate sampling algorithm, retaining 7% of the data points. The interaction between tev and $v02$ can be studied by performing the following multivariate query: $0.13 < tev < 0.5 \text{ AND } 0.45 < v02 < 1.0$.

The result of the query-driven analysis is shown in Fig. 10. Figure 10a shows all the 7% sample points that were selected by the sampling method. In Fig. 10b the result of the query is shown when applied to the ground truth data. Figure 10c depicts the query results when the sub-sampled data generated from multivariate PMI-based sampling is used. Compared to the result generated from random sampling, Fig. 10(d), one

can see that the multivariate association-driven sampling can answer the query with higher accuracy.

4.3.2 Reconstruction-Based Visualization of Sampled Data

The combustion data set provides a second example. The reconstruction technique is as discussed in Sect. 3.3. An example visualization of reconstructed data for the Y_OH field of a combustion data set is presented in Fig. 11. The sub-sampled data set was generated using mixture fraction and Y_OH fields and, in this example, 5% data points were stored. As can be seen, the reconstructed data, Fig. 11b, generated using the data samples produced by the multivariate association-driven sampling technique, produces a more accurate visualization compared to the reconstruction obtained from randomly sampled data set, Fig. 11c. The black dotted regions highlighted in Fig. 11b, c show the regions where the reconstruction error is more prominent compared to the ground truth raw data shown in Fig. 11a.

4.3.3 Multivariate Correlation Analysis of the PMI-Based Sampling Method

While analyzing multivariate data, application experts often study multivariate relationships among variables to explore variable interdependencies. Scientific features in multivariate data sets typically demonstrate statistical association in the form of linear or non-linear correlations among variable values. Therefore, it is important to preserve such variable relationships in the sub-sampled data so that flexible post hoc analysis can be done. In this section, the multivariate correlations obtained from the

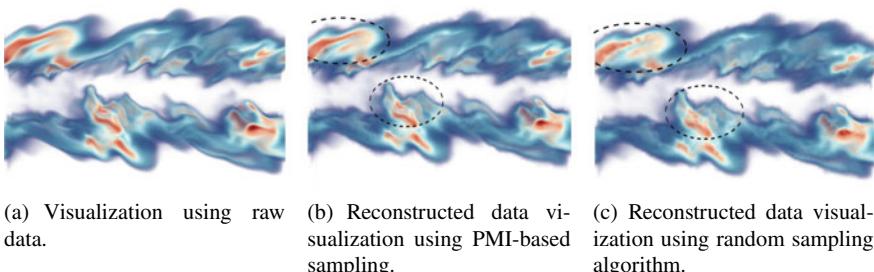


Fig. 11 Reconstruction-based visualization of Y_OH field for a turbulent combustion data set. Linear interpolation is used to reconstruct the data from the sub-sampled data sets. Figure **a** shows the result of the original raw data. Figure **b** provides the reconstruction result from the sub-sampled data generated by the PMI-based sampling method. Figure **c** presents the result of reconstruction from randomly sampled data. The sampling fraction used in this experiment is 0.05. This image is reprinted from our previous work [18]

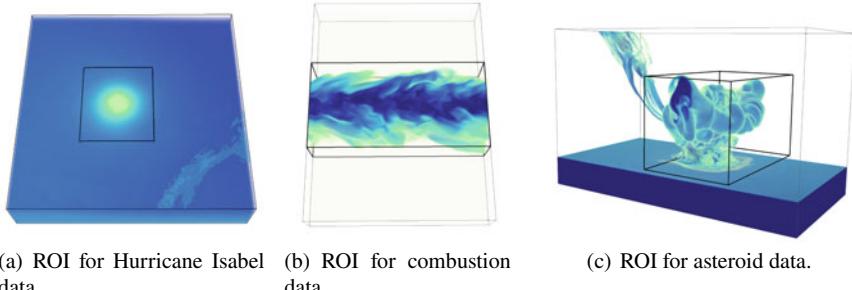


Fig. 12 Regions of interest (ROI) of different data sets used for analysis. Figure **a** shows the ROI in the Isabel data set, where the hurricane eye feature is selected. Figure **b** shows the ROI for the Combustion data set, where the turbulent flame region is highlighted. Finally, in Fig. **c**, the ROI selected in this example indicates the region where the asteroid has impacted the ocean surface and the splash of the water is ejected to the environment. This image is reprinted from our previous work [18]

Table 2 Evaluation of multivariate correlation for feature regions. The feature regions for each data set are shown in Fig. 12 indicated by a black box. This Table is reused from our previous work [18]

	Raw data		PMI-based sampling		Random sampling	
	Pearson's correlation	Distance correlation	Pearson's correlation	Distance correlation	Pearson's correlation	Distance correlation
Isabel data (Pressure and QVapor)	-0.19803	0.3200	-0.19805	0.3205	-0.1966	0.3213
Combustion data (mixfrac and Y_OH)	0.01088	0.4012	0.01624	0.04054	0.02123	0.4071
Asteroid data (tev and v02)	0.2116	0.2938	0.2273	0.2994	0.2382	0.31451

reconstructed data (the reconstruction is described in Sect. 4.3.2) that used multivariate association driven sampling are compared with the correlation values that were obtained from reconstructed data that used randomly selected samples. For analysis, a feature region (region of interest (ROI)) was selected for each data set. The ROI for each data set is shown in Fig. 12 using a dark black box. In Table 2 the results are presented. Pearson's correlation coefficient is again used for linear correlation and for measuring non-linear correlation, distance correlation [41] is used. The purpose of this study is to demonstrate that the multivariate association driven sampling technique is able to preserve the correlation among the modeled variables more accurately compared to the random sampling based method. Table 2 demonstrates that when the PMI-based multivariate sampling is used, the correlation in the reconstructed data matches closely with the correlation values obtained from the raw data. The correlation values obtained from random sampling have a higher deviation from the true correlation. This indicates that the multivariate PMI-based sampling technique preserves both the linear and non-linear correlations more accurately compared to the random sampling technique.

5 In Situ Performance

An in situ performance study for univariate spatial sampling methods was performed using the Nyx cosmology simulation code [4]. The three sampling methods were implemented in C++ and were integrated into the Nyx simulation code in the `writePlotFile()`; in situ I/O routine. As mentioned earlier, the sampling routines were called each time step and the test performed on a cluster with Intel Broadwell E5_2695_v4 CPUs (18 cores per node and 2 threads per core), and 125 GB of memory per node. For the in situ scaling study, the Nyx simulation was run for 100 time steps with $512 \times 512 \times 512$ resolution per time step. The data-driven univariate sampling algorithm is efficiently extended to a distributed memory setting since histograms are additive across data blocks. Consequently, only a few MPI-based communications are needed to compute the global data minimum and maximum values. Local histograms are computed based on the global data minimum and maximum and then the local histograms are “reduced” to generate the global histogram.

The in situ results are shown in Fig. 13 where the run times for different methods are shown. As can be seen from this figure, all the three methods scale quite well. It is also visible that the entropy-based method is slightly more expensive since it performs more data analysis in situ. However, this increase in time is compensated by the much higher quality samples generated by this method. As shown in Fig. 13b, the in situ I/O time is similar for all the three methods. Table 3 lists the time spent by the sampling methods as a percentage of total simulation time. This table indicates that, although the entropy-based method spends more time than other two naive methods, the percentage time spent compared to the actual simulation time is still negligible

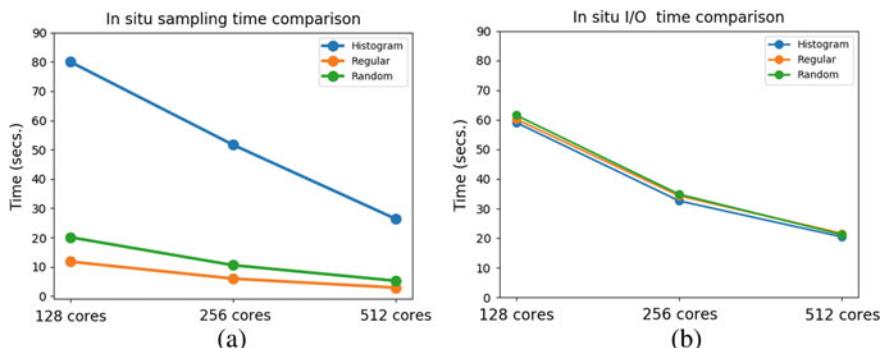


Fig. 13 **a** Comparison of in situ sampling performance among the three sampling methods: histogram-based sampling (blue), regular sampling (orange), and random sampling (green). **b** Comparison of in situ sampled data I/O times among the three sampling methods: histogram-based sampling (blue), regular sampling (orange), and stratified random sampling (green). This image is reprinted from our previous work [12]

Table 3 In situ percentage timings of different sampling methods and I/O timings w.r.t the simulation timings

	Hist. Samp %sim time	Reg. Samp %sim time	StRand. Samp %sim time	Hist. I/O %sim I/O	Reg. I/O %sim I/O	StRand. I/O %sim I/O
128 Cores	1.39	0.20	0.35	2.86	2.92	2.97
256 Cores	1.83	0.21	0.37	3.07	3.23	3.28
512 Cores	1.41	0.16	0.28	2.61	2.75	2.70

(only about 1.5%). Percentage disk I/O time for the sampled data is also shown in this table and they are all quite a small fraction of the simulation raw I/O.

Similar performance trends are expected for the multivariate sampling algorithm since it also uses a distribution-based approach. However, the computation of joint probability distributions will require more time compared to the univariate distributions. In this case, to achieve viable in situ computational performance, one can follow a joint probability distribution estimation approach [33]. Another alternative is to use the Copula-based modeling schemes that are computationally efficient for in situ environments [23].

6 Discussions and Limitations

In this chapter, in situ sampling methods for both univariate and multivariate data sets have been discussed in detail. Examples of data-driven approaches demonstrate their effectiveness compared to the naive (albeit faster) regular/random methods. It is also important to note that data-driven methods may not necessarily be applicable for all the time steps of a simulation. Specifically, for simulations (such as Nyx) that are initialized on a random field and for first few time steps may have no features in the data. For such cases, it is recommended that users try to employ random/regular sampling methods as the data-driven method may not produce any meaningful samples or capture any interesting features. Due to the lack of features in the data, essentially, data-driven methods will reduce to behaving like random methods. Multivariate sampling methods work well when there is some association or correlation among variables that are captured by PMI, an information-theoretic measure used to guide the samples. When simulation variables are statistically independent, then the univariate sampling technique can be applied instead of multivariate sampling, which behaves the same as random sampling when variables are statistically independent.

7 Future Directions and Conclusion

This chapter described several different sampling approaches that can be performed *in situ* for data reduction. When considering univariate data for spatial sampling, the data distribution can be used to identify and save important scalars that are more likely to be features in the data set. Similarly, for multivariate data set, point-wise mutual information can be leveraged to identify locations that capture multivariate importance. Looking forward in this research area, there are additional ways to leverage and apply information-theoretic sampling methods. For example, they can be applied to vector fields and they can be applied across time, in addition to space. Using these light-weight yet scalable methods, data-driven sampling can yield results that are more meaningful to the scientist, compared to naive methods such as random or regular sampling.

Acknowledgements We would like to thank our Data Science at Scale Team colleagues: D. H. Rogers, L.-T. Lo, J. Patchett, our colleague from the Statistical Group CCS-6: Earl Lawrence, our industry partners at Kitware and other collaborators: C. Harrison, M. Larsen. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. The Hurricane Isabel data set has kindly been provided by Wei Wang, Cindy Bruyere, Bill Kuo, and others at NCAR. Tim Scheitlin at NCAR converted the data into the Brick-of-Float format. The Turbulent Combustion data set is made available by Dr. Jacqueline Chen at Sandia National Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visualization. This research was released under LA-UR-20-21090.

References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large data visualization. *The Visualization Handbook*, vol. 717 (2005)
2. Ahrens, J., Jourdain, S., O'Leary, P., Patchett, J., Rogers, D.H., Petersen, M.: An image-based approach to extreme scale *in situ* visualization and analysis. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 424–434. IEEE Press (2014)
3. Akiba, H., Ma, K., Chen, J.H., Hawkes, E.R.: Visualizing multivariate volume data from turbulent combustion simulations. *Comput. Sci. Eng.* **9**(2), 76–83 (2007). <https://doi.org/10.1109/MCSE.2007.42>
4. Almgren, A.S., Bell, J.B., Lijewski, M.J., Lukić, Z., Van Andel, E.: Nyx: a massively parallel AMR code for computational cosmology. *apj* **765**, 39 (2013). <https://doi.org/10.1088/0004-637X/765/1/39>
5. Alted, F.: BLOSC (2009). <http://blosc.pytables.org/>. [online]
6. Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., Mauldin, J.: Paraview catalyst: enabling *in situ* data analysis and visualization. In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp. 25–29. ACM (2015)

7. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: 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). <https://doi.org/10.1109/ISAV.2016.013>
8. Battle, L., Stonebraker, M., Chang, R.: Dynamic reduction of query result sets for interactive visualizaton. In: 2013 IEEE International Conference on Big Data, pp. 1–8 (2013). <https://doi.org/10.1109/BigData.2013.6691708>
9. Bauer, A.C., et al.: In Situ methods, infrastructures, and applications on high performance computing platforms, a state-of-the-art (STAR) report. In: Computer Graphics Forum, Proceedings of Eurovis 2016, vol. 35(3) (2016). LBNL-1005709
10. Bilmes, J.: A gentle tutorial on the em algorithm including gaussian mixtures and baum-welch. Technical report, International Computer Science Institute (1997)
11. Biswas, A., Dutta, S., Shen, H., Woodring, J.: An information-aware framework for exploring multivariate data sets. *IEEE Trans. Vis. Comput. Graph.* **19**(12), 2683–2692 (2013). <https://doi.org/10.1109/TVCG.2013.133>
12. Biswas, A., Dutta, S., Pulido, J., Ahrens, J.: In situ data-driven adaptive sampling for large-scale simulation data summarization. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV ’18, pp. 13–18. Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3281464.3281467>
13. Chen, M., Feixas, M., Viola, I., Bardera, A., Shen, H., Sbert, M.: Information Theory Tools for Visualization. CRC Press, Boca Raton, FL, USA (2006)
14. Childs, H., et al.: VisIt: an end-user tool for visualizing and analyzing very large data. In: High Performance Visualization—Enabling Extreme-Scale Scientific Insight, pp. 357–372. CRC Press/Francis–Taylor Group (2012)
15. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. In: Proceedings of the 27th annual meeting on Association for Computational Linguistics, ACL ’89, pp. 76–83. Association for Computational Linguistics, Stroudsburg, PA, USA (1989). <https://doi.org/10.3115/981623.981633>
16. Cover, T., Thomas, J.: Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing, 2nd edn. Wiley-Interscience, New York, NY, USA (2006)
17. Di, S., Cappello, F.: Fast error-bounded lossy HPC data compression with sz. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 730–739 (2016). <https://doi.org/10.1109/IPDPS.2016.11>
18. Dutta, S., Biswas, A., Ahrens, J.: Multivariate pointwise information-driven data sampling and visualization. *Entropy* **21**(7), 699 (2019)
19. Dutta, S., Liu, X., Biswas, A., Shen, H.W., Chen, J.P.: Pointwise information guided visual analysis of time-varying multi-fields. In: SIGGRAPH Asia 2017 Symposium on Visualization, SA ’17, pp. 17:1–17:8. ACM, New York, NY, USA (2017). <https://doi.org/10.1145/3139295.3139298>
20. Fisher, D., Popov, I., Drucker, S., Schraefel, M.: Trust me, i’m partially right: incremental visualization lets analysts explore large datasets faster. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI ’12, pp. 1673–1682. Association for Computing Machinery, New York, NY, USA (2012). <https://doi.org/10.1145/2207676.2208294>
21. Gosink, L., Anderson, J., Bethel, W., Joy, K.: Variable interactions in query-driven visualization. *IEEE Trans. Vis. Comput. Graph.* **13**(6), 1400–1407 (2007). <https://doi.org/10.1109/TVCG.2007.70519>
22. Gosink, L.J., Garth, C., Anderson, J.C., Bethel, E.W., Joy, K.I.: An application of multivariate statistical analysis for query-driven visualization. *IEEE Trans. Vis. Comput. Graph.* **17**(3), 264–275 (2011). <https://doi.org/10.1109/TVCG.2010.80>
23. Hazarika, S., Dutta, S., Shen, H., Chen, J.: Codda: a flexible copula-based distribution driven analysis framework for large-scale multivariate data. *IEEE Trans. Vis. Comput. Graph.* **25**(1), 1214–1224 (2019). <https://doi.org/10.1109/TVCG.2018.2864801>

24. Islam, A., Pearlman, W.A.: Embedded and efficient low-complexity hierarchical image coder. In: Electronic Imaging'99, pp. 294–305. International Society for Optics and Photonics (1998)
25. Jänicke, H., Wiebel, A., Scheuermann, G., Kollmann, W.: Multifield visualization using local statistical complexity. *IEEE Trans. Vis. Comput. Graph.* **13**(6), 1384–1391 (2007). <https://doi.org/10.1109/TVCG.2007.70615>
26. Jaynes, E.T.: Information theory and statistical mechanics. *Phys. Rev.* **106**(4), 620–630 (1957). <https://doi.org/10.1103/PhysRev.106.620>
27. Larsen, M., Ahrens, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: 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, ISAV'17, pp. 42–46. ACM, New York, NY, USA (2017). <https://doi.org/10.1145/3144769.3144778>
28. Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J., Childs, H.: Achieving portable performance for wavelet compression using data parallel primitives. In: Proceedings of the 17th Eurographics Symposium on Parallel Graphics and Visualization, PGV '17, p. 73–81. Eurographics Association, Goslar, DEU (2017). <https://doi.org/10.2312/pgv.20171095>
29. Lindstrom, P.: Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.* **20**(12), 2674–2683 (2014)
30. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. *IEEE Trans. Vis. Comput. Graph.* **12**(5), 1245–1250 (2006)
31. Liu, X., Shen, H.W.: Association analysis for visual exploration of multivariate scientific data sets. *IEEE Trans. Vis. Comput. Graph.* **22**(1), 955–964 (2016). <https://doi.org/10.1109/TVCG.2015.2467431>
32. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible io and integration for scientific codes through the adaptable io system (adios). In: Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE '08, pp. 15–24. Association for Computing Machinery, New York, NY, USA (2008). <https://doi.org/10.1145/1383529.1383533>
33. Lu, K., Shen, H.W.: A compact multivariate histogram representation for query-driven visualization. In: Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV), LDAV '15, pp. 49–56 (2015)
34. Nouanesengsy, B., Woodring, J., Patchett, J., Myers, K., Ahrens, J.: ADR visualization: a generalized framework for ranking large-scale scientific data using analysis-driven refinement. In: 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), pp. 43–50 (2014). <https://doi.org/10.1109/LDAV.2014.7013203>
35. Park, Y., Cafarella, M., Mozafari, B.: Visualization-aware sampling for very large databases. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 755–766 (2016). <https://doi.org/10.1109/ICDE.2016.7498287>
36. Patchett, J., Gisler, G.: Deep water impact ensemble data set. Los Alamos National Laboratory, LA-UR-17-21595 (2017). <http://dssdata.org>
37. Shannon, C.E.: A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.* **5**(1), 3–55 (2001). <https://doi.org/10.1145/584091.584093>
38. Son, S., Chen, Z., Hendrix, W., Agrawal, A., Liao, W., Choudhary, A.: Data compression for the exascale computing era - survey. *Supercomput. Front. Innov. Int. J.* **1**(2), 76–88 (2014). <https://doi.org/10.14529/jsfi140205>
39. Stockinger, K., Shalf, J., Wu, K., Bethel, E.W.: Query-driven visualization of large data sets. In: VIS 05. IEEE Visualization 2005, pp. 167–174 (2005). <https://doi.org/10.1109/VISUAL.2005.1532792>
40. Su, Y., Agrawal, G., Woodring, J., Myers, K., Wendelberger, J., Ahrens, J.: Taming massive distributed datasets: data sampling using bitmap indices. In: Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing, HPDC '13, pp. 13–24. Association for Computing Machinery, New York, NY, USA (2013). <https://doi.org/10.1145/2462902.2462906>
41. Székely, G.J., Rizzo, M.L., Bakirov, N.K.: Measuring and testing dependence by correlation of distances. *Ann. Stat.* **35**(6), 2769–2794 (2007). <http://www.jstor.org/stable/25464608>

42. Tao, D., Di, S., Chen, Z., Cappello, F.: Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1129–1139 (2017). <https://doi.org/10.1109/IPDPS.2017.115>
43. Tikhonova, A., Correa, C.D., Ma, K.: Explorable images for visualizing volume data. In: 2010 IEEE Pacific Visualization Symposium (PacificVis), pp. 177–184 (2010)
44. Van de Cruys, T.: Two multivariate generalizations of pointwise mutual information. In: Proceedings of the Workshop on Distributional Semantics and Compositionality, DiSCo '11, pp. 16–20. Association for Computational Linguistics, Stroudsburg, PA, USA (2011). <http://dl.acm.org/citation.cfm?id=2043121.2043124>
45. Verdu, S.: Fifty years of Shannon theory. *IEEE Trans. Inf. Theory* **44**(6), 2057–2078 (1998). <https://doi.org/10.1109/18.720531>
46. Wang, K., Kewei Lu, Wei, T., Shareef, N., Shen, H.: Statistical visualization and analysis of large data using a value-based spatial distribution. In: 2017 IEEE Pacific Visualization Symposium (PacificVis), pp. 161–170 (2017)
47. Wang, W., Bruyere, C., Kuo, B., Scheitlin, T.: IEEE visualization 2004 contest data set (2004). NCAR. <http://sciviscontest.ieeevis.org/2004/data.html>
48. Wei, T., Dutta, S., Shen, H.: Information guided data sampling and recovery using bitmap indexing. In: 2018 IEEE Pacific Visualization Symposium (PacificVis), pp. 56–65 (2018). <https://doi.org/10.1109/PacificVis.2018.00016>
49. Woodring, J., Ahrens, J., Figg, J., Wendelberger, J., Habib, S., Heitmann, K.: In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. *Comput. Graph. Forum* **30**(3), 1151–1160 (2011). <https://doi.org/10.1111/j.1467-8659.2011.01964.x>
50. Ye, Y.C., Neuroth, T., Sauer, F., Ma, K., Borghesi, G., Konduri, A., Kolla, H., Chen, J.: In situ generated probability distribution functions for interactive post hoc visualization and analysis. In: 2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV), pp. 65–74 (2016)

In Situ Wavelet Compression on Supercomputers for Post Hoc Exploration



Shaomeng Li, John Clyne, and Hank Childs

Abstract Wavelet compression is a popular approach for reducing data size while maintaining high data integrity. This chapter considers how wavelet compression can be used for data visualization and post hoc exploration on supercomputers. There are three major parts in this chapter. The first part describes the basics of wavelet transforms, which are essential signal transformations in a wavelet compression pipeline, and how their properties can be used for data compression. The second part analyzes the efficacy of wavelet compression on scientific data, with a focus on analyses involving scientific visualizations. The third part evaluates how well wavelet compression fits in an in situ workflow on supercomputers. After reading this chapter, readers should have a high-level understanding of how wavelet compression works, as well as its efficacy for in situ compression and post hoc exploration.

1 Motivation

As discussed in the introductory chapter, two prominent challenges for in situ processing are the lack of a human-in-the-loop and the need to process a current time step entirely before it is replaced by the new one. An important approach for solving this problem is to transform and reduce the data set to a small enough form that time steps can be saved on disk. This enables human-in-the-loop interactivity, since the reduced data can be explored post hoc by a domain scientist using fewer resources than were needed for the simulation. This chapter explores this direction from the perspective of wavelets, i.e., using wavelet transformations and lossy compression

S. Li (✉) · J. Clyne
National Center for Atmospheric Research, Boulder, CO, USA
e-mail: shaomeng@ucar.edu

J. Clyne
e-mail: clyne@ucar.edu

H. Childs
University of Oregon, Eugene, OR, USA
e-mail: hank@uoregon.edu

to reduce data sizes to the point that individual time slices can be saved in a timely manner for post hoc exploration.

2 Introduction to Wavelet Compression

This section is broken into three parts: Sect. 2.1 gives an overview of how wavelets can be used for compression, Sect. 2.2 describes wavelet transforms in more detail, and Sect. 2.3 describes choices for carrying out compression.

2.1 Overview of Using Wavelets for Compression

Wavelet compression is a transform-based technique, because it relies on *wavelet transforms* as its core operation. Wavelet transforms operate similarly to Fourier transforms, in that they both transform data and represent it in another domain. Where Fourier transforms represent data in the frequency domain, wavelet transforms represent data in the wavelet domain, which contains both frequency and time information.

The wavelet domain representation has some desirable properties that make it useful in a wide range of applications, in particular signal and image processing. Examples of such applications include edge detection [10, 42], pattern recognition [29, 38], signal-noise separation [1, 33], multi-resolution analysis [14], and of course lossy compression (notably the JPEG 2000 image compression standard [36]). The final entry in this list, lossy compression, and more specifically general-purpose in situ scientific data reduction, is the focus of this chapter. That said, other visualization works have considered wavelets as well. One important example is wavelet-based rendering, and in particular for 3D volume renderings. These works are often GPU-centric, with a pre-processing step to store encoded wavelet coefficients in a form that is friendly to modern GPU architectures and graphics programming models. Then, during the rendering phase, they approximate the original data on-the-fly while performing rendering tasks on the GPU. Additional details on wavelet-based rendering can be found in works by Wang and Shen [40], Guthe et al. [11], Ihm and Park [13], and Kim and Shin [17]. The focus of this chapter differs in that this chapter considers a general-purpose lossy compression technique.

Wavelet-based lossy compression is enabled by the *information compaction* property of wavelet transforms: the ability to decorrelate a signal, removing redundant information, and expressing the non-redundant information using only a small number of coefficients in the wavelet domain. Lossy compression is then achieved by prioritizing the storage budget towards that small number of information-rich coefficients. Usually the more coherence in the input data, the better wavelets will compact information, and thus the better data will be compressed.

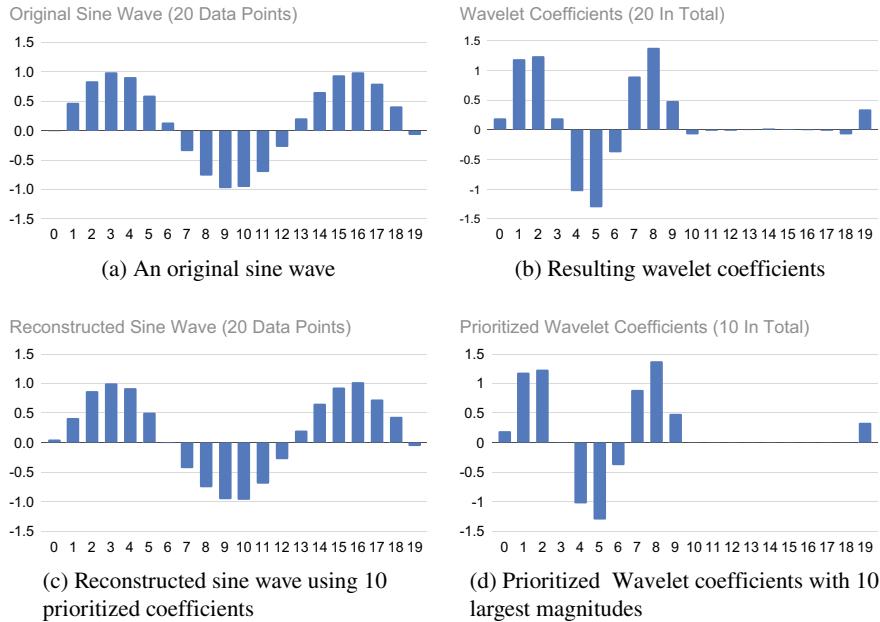


Fig. 1 Demonstration of information compaction of wavelets. This figure is organized as four subfigures, with the two figures on the left column (Subfigure **a** and **c**) representing twenty data values and the two figures on the right column (Subfigure **b** and **d**) representing twenty wavelet coefficients. Subfigure **a** shows the input data to a wavelet transformation, a discrete sine wave with 20 data points. Subfigure **b** shows the output of the wavelet transformation, 20 wavelet coefficients. Importantly, the wavelet transform achieves no data savings. However, Subfigure **d** shows one approach for achieving data savings, namely “prioritized coefficients.” With this approach, the largest magnitudes are retained (10 in this case), and the rest are zeroed out. This approach achieves a 2X storage saving. That said, the sine wave reconstructed from prioritized coefficients (Subfigure **c**) is no longer perfectly accurate—although it highly resembles the original input from Subfigure **a**, it is not the same, and in particular the values at locations zero and six are clearly different

Figure 1 illustrates an example, specifically when applying wavelet transforms to a discrete sine wave. Here, the resulting wavelet coefficients have information content proportional to their magnitude. As the illustration shows, much of the information is compacted into a few large-magnitude coefficients, resulting in many near-zero coefficients. Lossy compression removes these near-zero coefficients and, when reconstructing the data later, the missing coefficients are treated as zeros. The strength of wavelets is that the resulting impact on the reconstructed field is small.

2.2 Basics of Wavelet Transforms

This subsection describes the basics of wavelet transforms, and does so in three parts. Section 2.2.1 focuses on the simplest case, which is a one-dimensional wavelet transform; Sect. 2.2.2 describes extensions for multi-dimensional wavelet transforms; and Sect. 2.2.3 describes different choices for wavelet kernels.

2.2.1 One-Dimensional Wavelet Transform in Detail

One-dimensional wavelet transform is the building block for wavelet transforms in multi-dimensional volumes. These transforms can be viewed as expressing an input signal $x(t)$ as an invertible, linear expansion about a set of basis functions:

$$x(t) = \sum_{j \in \mathbb{Z}} \sum_{k \in \mathbb{Z}} a_{j,k} \psi_{j,k}(t), \quad (1)$$

where $a_{j,k}$ are real-valued coefficients, and $\psi_{j,k}(t)$ are wavelet functions typically forming an orthonormal basis. Conceptually, the coefficients $a_{j,k}$ measure the similarity between the input signal and the basis functions. For lossy compression, a small subset of available wavelet transforms, namely *non-expansive* transforms, are often used. With these non-expansive transforms, a finite, discrete series of data points $x[n]$ will result in the same number of coefficients $a_{j,k}$, instead of expanding the coefficient number. Because a wavelet transform is invertible, the original signal can be reconstructed from *all* wavelet coefficients without losing any accuracy.

The resulting wavelet coefficients have two flavors: the “approximation” coefficients which provide a coarsened representation of the data, and the “detail” coefficients which contain the missing information from the coarsened representation. Figure 1b illustrates this intuition: the first ten coefficients approximate the input data, and the second ten coefficients record the deviation. Wavelet transforms can then be applied recursively on approximation coefficients from previous wavelet transforms, resulting in a hierarchy of coefficients. This recursive application of wavelet transforms concentrates information content into fewer and fewer coefficients, and the resulting coefficient hierarchy forms a data representation spanning multiple resolutions. This hierarchy is reflected in the linear expansion in Eq. 1, where j indicates the different scales.

2.2.2 Wavelet Transform for Higher Dimensional Volumes

One-dimensional wavelet transforms can be extended to multiple dimensions by successively applying a one-dimensional transform along each axis, i.e., output coefficients of one transform become the input of the next transform along a different axis. This practice takes advantage of data coherence along all dimensions, as the more coherence being removed from the data, the more compression will be achieved.

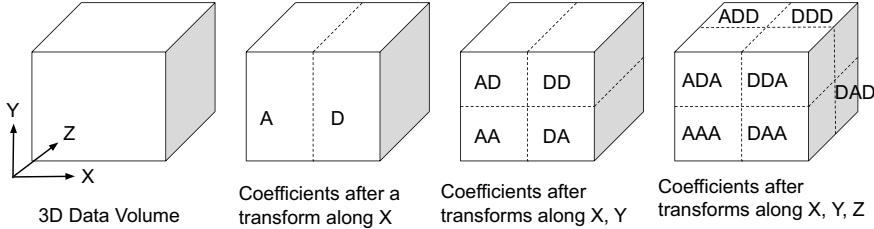


Fig. 2 Illustration of one level of wavelet transform for a three-dimensional volume. Approximation and detail coefficients are denoted using “A” and “D.” From left to right are the original data volume, and the resulting coefficients after wavelet transforms are applied in the X , Y , and Z axes, respectively. After each wavelet transform, every coefficient receives a label of either “A” or “D” with respect to the axis the transform was performed. The ordering of the “A’s” and “D’s” in the labels corresponds to the axis the transform was applied to (X first, followed by Y , then Z). The “AAA” label on the bottom-left sub-volume of the right-most subfigure indicates that this sub-volume contains approximation coefficients from transforming all three axes. This image is adapted from our previous work [22]

Figure 2 illustrates a three-dimensional wavelet transform. In this example, each row goes through a wavelet transform pass in the X direction, resulting in approximation and detail coefficients with respect to the X axis (subfigure that is second from the left). Second, these coefficients go through wavelet transforms in the Y direction as columns, resulting in approximation and detail coefficients with respect to the Y axis (subfigure that is third from the left). Third, the output coefficients from the second set of transforms go through wavelet transforms in the Z direction, resulting in approximation and detail coefficients with respect to the Z axis (the last subfigure). The result of these three sets of transforms is that every coefficient in the volume now represents either approximation or detail coefficients with respect to all three axes.

Finally, similar to recursive application of one-dimensional wavelet transforms, multi-dimensional transforms can also be applied recursively to achieve further information compaction. In this use case, the next level of wavelet transforms will be applied on the sub-volume which represents approximation coefficients with respect to all axes, i.e., the “AAA” sub-cube of Fig. 2.

2.2.3 Wavelet Kernel Choices

Wavelet kernels are used to generate basis functions, specifically the basis functions $\psi_{j,k}(t)$ from Eq. 1. There are various choices of wavelet kernels. For example, the Haar wavelet is the simplest kernel whose approximation coefficients represent the average of two neighboring data points. In this case, detail coefficients represent the difference between this average and the two data points. Popular wavelets beside Haar include Coiflets [41], Daubechies wavelets [6] and Cohen-Daubechies-Feauveau (CDF) wavelets [4], which are all families of wavelets with flexible configurations.

Two newly developed wavelets, namely curvelets [3] and surfacelets [25], have also gained traction in multi-resolution analysis [30, 31]. The wide array of available wavelet kernels enables the broad application of wavelets discussed in Sect. 2.1.

In lossy compression applications, two members from the CDF family, CDF 5/3 and CDF 9/7, are particularly popular because they have excellent information compaction capabilities, and they allow for non-expansive wavelet transform on aperiodic, finite-length inputs. The latter property is especially important since almost all other wavelet families are expansive over aperiodic finite signals, resulting in more coefficients than inputs. In fact, CDF 5/3 is used in the JPEG 2000 [36] standard for lossless compression, while CDF 9/7 is used for JPEG 2000's lossy compression, as well as many other applications in image compression research [8, 28, 32, 34]. For lossy scientific data compression, CDF 9/7 has been demonstrated to provide better compression efficiency compared to CDF 5/3 and the Haar kernel, with only a modest performance disadvantage [19]. In the remainder of this chapter, the wavelet kernel for compression will be CDF 9/7, unless specified otherwise.

2.3 *Compression Strategy Options*

As discussed in Sect. 2.2.1, the output of a wavelet transform is at best the same size as the input data array using a non-expansive transform—wavelet transforms by themselves do not reduce data. Instead, wavelet compression occurs subsequently, and this is where loss of information occurs. This subsection focuses on this subsequent lossy compression step, and describes three major compression strategies and their properties.

2.3.1 Multi-resolution

With a multi-resolution approach, data is reconstructed at lower resolutions in each dimension. More specifically, coefficients are laid out naturally with respect to the coefficient hierarchy that is created by the recursive application of wavelet transforms. Because each level of coefficients reconstructs an approximation of the original data, compression is achieved by storing only some levels of coefficients from the hierarchy, which provides approximations of the data at lower resolutions.

Each application of a wavelet transform coarsens the data array into half of its previous resolution, i.e., the approximation coefficients. As a result, the multi-resolution compression strategy offers a pyramid representation that is strictly limited to power-of-two reductions along each axis. In the case of three-dimensional volumes, the applicable compression ratios are of the form $8^N:1$, where N is the number of iterations of wavelet transforms applied. That is, applicable compression ratios include 8:1, 64:1, 512:1, etc.

Though multi-resolution provides a means for reducing data sizes on disk, it does not fully take advantage of wavelet's information compaction properties. This

contrasts with the compression strategies discussed in the following subsections, which almost always provide better data integrity at any given compression ratio. The strength of the multi-resolution approach, however, is in reducing memory footprint and speeding up analysis during data exploration and visualization phases. This is because performing exploratory visualizations on low resolution approximations makes the processing speed orders of magnitude faster. Further, when interesting phenomena are observed, domain scientists can zoom in on that region, and bring in higher resolution data. This usage scenario fits Shneiderman's visualization mantra: "overview first, zoom and filter, then details-on-demand" [35].

Finally, multi-resolution takes a unique position in wavelet technologies because it can be used in conjunction with compression strategies discussed in the following subsections. That means a single copy of the compressed data both reduces storage on disk and can be used to reduce memory footprint during data exploration. This versatility of wavelets is uncommon among other compression techniques [23].

2.3.2 Coefficient Prioritization

Coefficient prioritization (illustrated in Fig. 1) is different than the multi-resolution approach, since it focuses on the relative importance of each coefficient rather than on a hierarchical representation. When applying recursive wavelet transforms, a small number of coefficients typically contain the vast majority of information. The coefficient prioritization strategy, then, prioritizes all coefficients based on their information content. Compression is then achieved by storing only the collection of information-rich coefficients, and discarding the rest of them (effectively treating them as zeroes).

A related issue is how to identify the coefficients that are information-rich. Fortunately, with certain wavelet kernels, including CDF 9/7 and CDF 5/3, the magnitude of a wavelet coefficient is proportional to its information content, which makes the prioritization step easy to conduct. As a result, the largest wavelet coefficients are saved, and the rest are discarded.

Coefficient prioritization is a lossy compression strategy, as it supports reconstructing the mesh on its full resolution. That said, the reconstruction at individual data points may be inaccurate. Coefficient prioritization also differs from the multi-resolution strategy in that: (1) it supports arbitrary compression ratios, by choosing what percentage of total coefficients to keep; and (2) it requires extra mechanisms to keep track of where the prioritized coefficients belong in the coefficient hierarchy. For the second point, the typical way to keep track of which coefficients are saved is to explicitly store their addresses. Therefore, while coefficient prioritization has benefits (i.e., arbitrary compression ratios and focusing on the most information-rich coefficients), it also has drawbacks (i.e., storage overhead and extra complexity). On the whole, however, research has shown that its benefits outweigh its drawbacks: the focus on storing the most information-rich coefficients more than offsets the storage overhead [19].

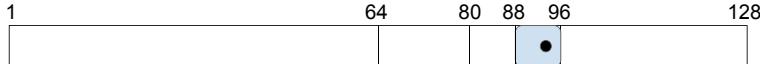


Fig. 3 Illustration of using progressive quantization to encode a quantized version of a data point within the data range of (1, 128). The black dot represents the data point of interest. In the encoded form, the first bit specifies that the data point is within the upper half of the data range, i.e., (64, 128); the second bit further narrows the range to (64, 96); the next bit narrows the range to (80, 96); and the last bit narrows the range to (88, 96). Within a data range, all data points are considered to be at the middle of that range, so in this example, the data point of interest is quantized as 92. This particular encoding takes four bits of storage

2.3.3 Advanced Codecs

The previous sections have described how to organize wavelet coefficient hierarchies; this section describes data structures designed specifically to encode such hierarchies efficiently. We refer to these data structures as *advanced codecs*. The process of building and maintaining these data structures is significantly more complex than the procedures to achieve multi-resolution and coefficient prioritization, so this section only provides a high-level overview on these codecs.

Advanced codecs achieve efficient storage by making use of the hierarchical structure of wavelet coefficients and the similarities between different levels. They also use *progressive quantization* [7] to encode quantized versions of all coefficients, during which process coefficients with larger magnitudes effectively receive more bit budget. The idea of progressive quantization is to record a series of binary decisions on whether a data value is larger or smaller than a set of quantization thresholds. The number of decisions recorded for a data value may vary—the more binary decisions recorded, the more accurate the quantized approximations become. Figure 3 presents an example of progressive quantization which uses four bits to locate a data point to a narrower range that is $\frac{1}{16}$ th of the original data range. In advanced codecs, wavelet coefficients are encoded by different numbers of bits. This is because a wavelet coefficient is treated as zero without taking storage when it is below the quantization threshold, and only starts to be encoded when the threshold falls below its magnitude. As the quantization threshold always begins from a big value and decreases each iteration, coefficients with larger magnitudes start to receive bits early, and smaller coefficients start to receive bits later. This practice makes sure that more storage budget is allocated to the most information-rich coefficients.

The most well-known data structures and their corresponding codecs include ZeroTrees [34], Set Partitioning In Hierarchical Trees (SPIHT) [16, 32], Set Partitioned Embedded bloCKs (SPECK) [28, 37], and Embedded Block Coding with Optimized Truncation (EBCOT) [39]. The final entry in this list, EBCOT, has been adopted by the JPEG 2000 [36] standard.

Advanced codecs are lossy, as they support reconstruction on the mesh’s native resolution with inaccuracy occurring on individual data points. They also support arbitrary compression ratios. Regarding storage, these special data structures already keep coefficient locations implicitly, thus eliminating the storage overhead

incurred by addressing coefficients in the coefficient prioritization strategy. As a result, advanced codecs represent state-of-the-art wavelet compression in terms of rate-distortion (i.e., how much error is introduced given a certain compression ratio). Finally, we note that advanced codecs have not seen significant usage outside of the scope of image compression, indicating that more study is required to better understand their viability in scientific simulation and in situ visualization settings.

3 Evaluating the Effects of Wavelet Compression on Scientific Visualization

Error characteristics for wavelet compression depend on the input data. Wavelet transforms typically compact information most effectively when operating on smooth input, meaning that adjacent data values change in similar fashions. An important question, then, is on the efficacy of wavelet compression in real world settings on scientific data from computational simulations. Further, an important aspect when considering efficacy is how the reconstructed data is evaluated. In many areas, traditional statistics for measuring error introduced by compression, such as root-mean-square error (RMSE) and L^∞ -norms, are easy to understand and applicable. However, scientific visualization and visual analytics pose additional difficulties for assessing compression, since different visualizations and visual analytics often have very different levels of tolerance to error in the data.

This section examines multiple visualization tasks and evaluates the effects of wavelet compression on visual analytics tasks. Two techniques are used to conduct this evaluation. The first includes statistics on the compressed data (RMSE and L^∞ -norm). They provide a basic understanding of the compression efficiency. The second includes visualization-specific metrics. These metrics are derived based on the characteristics of specific visualizations, and are able to capture the direct impact of compression on specific analysis outcomes. For each of the visualization tasks in this section, at least one of these evaluation techniques is used.

The evaluations in the remainder of this section have some commonalities. First, each uses coefficient prioritization as its wavelet compression strategy. Second, their exact statistics are calculated using the following equations:

$$RMSE = \sqrt{\left(\sum_{i \in N} (v[i] - \tilde{v}[i])^2\right)/N}, \quad L^\infty\text{-norm} = \max_{i \in N} (v[i] - \tilde{v}[i]). \quad (2)$$

where $v[i]$ and $\tilde{v}[i]$ are data values from the original and compressed data sets, and N is the total number of data values. In practice, both RMSE and L^∞ -norm are often normalized by the data range. The full details of the experiments from these studies can be found in their original works (Sects. 3.1: [19], Sect. 3.2: [24], and Sect. 3.3: [21]).

3.1 Critical Structure Identification

3.1.1 Visual Analytics Use Case Description

The first efficacy evaluation considers a visual analytics method from Gaither et al. [9]. Their approach focuses on critical structures within a turbulent flow simulation, and studied the global population of these structures. Critical structures were defined as the output of a two-step process. The first step was to isolate regions with enstrophy values higher than α , a fixed value provided by domain scientists, on a $4,096^3$ grid. Contiguous regions were found through a connected components algorithm [12]. The result of this step was millions of these high-enstrophy regions. The second step was to eliminate structures with a volume smaller than some threshold β , which was again a fixed value provided by domain scientists. The result of this step was to reduce the number of critical structures down to hundreds. Figure 4a shows a screenshot of these identified critical structures in the $4,096^3$ volume, and Fig. 4b shows a close-up look at one of the critical structures.

This analysis routine can potentially be quite sensitive to changes in the enstrophy field from compression. If the compressed enstrophy breaks a component, then the result may put that component below β . Similarly, if the compressed enstrophy joins two disjoint components, then the result may put the joined component above β . Such a change would affect the statistics of the global population of critical structures.

3.1.2 Evaluation—Structure Identification

Figure 4c–f illustrate the visualization using compressed data. They show that even with aggressive compression (i.e., 256:1), the visual analytics routine still identifies many critical structures. Upon close examination, differences between the baseline and compressed renderings reveal that the visual analytics routine can cause both disappearance and creation of structures. That said, the overall trend is that there are fewer and fewer structures being identified as compression increases. To better understand this phenomenon and quantitatively evaluate the effect of compression, two types of error are considered. The first type of error is referred to as *False Positives (FP)*, meaning that a structure which does not exist in the baseline rendering was falsely identified from the compressed data. The second type of error is referred to as *False Negatives (FN)*, meaning that a structure appeared in the baseline rendering was not identified from the compressed data. The rest of structures that are identified from both original and compressed data are referred to as *Common Structures (Comm)*; details on how a structure is classified can be found in the original study [19]. The percentage of the two types of errors introduced by compression can be calculated as:

$$FP_Percentage = \frac{FP}{FP + Comm} \times 100, \quad FN_Percentage = \frac{FN}{FN + Comm} \times 100. \quad (3)$$

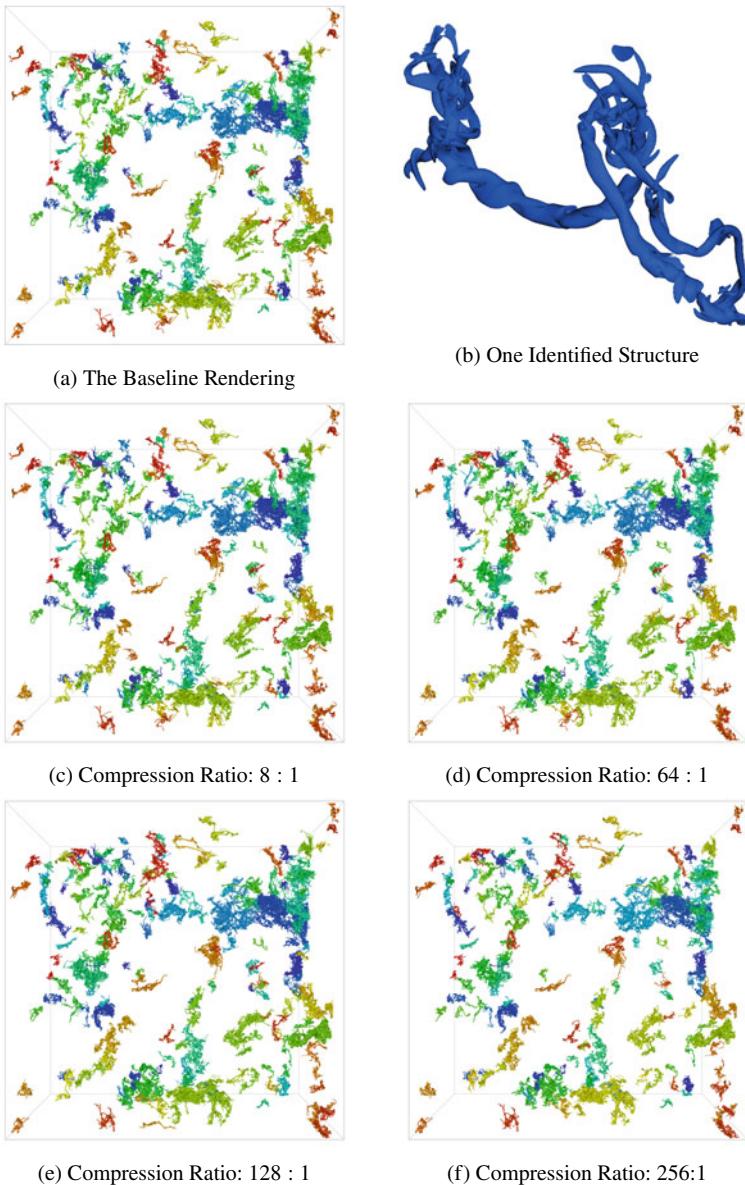


Fig. 4 Visualization of the critical structure identification analysis with the baseline result as well as renderings from compressed data. Within each rendering, each individual structure is colored by a unique color. This image is reprinted from our previous work [19]

Table 1 False positive and false negative percentage of structure identifications

Comp. Ratio	8 : 1	64 : 1	128 : 1	256 : 1
FP Pct.	3	14	17	16
FN Pct.	3	17	20	28

Table 2 Normalized RMSE and L^∞ -norm values at different compression ratios

Comp. Ratio	32 : 1	64 : 1	128 : 1	256 : 1
RMSE	$9.6e - 6$	$1.4e - 5$	$2.1e - 5$	$3.1e - 5$
L^∞ -norm	$3e - 3$	$5e - 3$	$1e - 2$	$3e - 2$

Table 1 provides the evaluation results. Both types of error grow as the compression ratio grows. Further, the false negative value is always higher than false positive, meaning compression tends to make the visual analytics task fail in identifying certain structures. Finally, at 256:1, the false positive value falls lower than that of 128:1, while the false negative value grows. This counter-intuitive result is really a failing in the metric; the visual analytics identifies a much smaller total number of structures at this compression ratio, so it becomes hard to have a high false positive value. More discussion about this visual analytics and evaluation is available in [19], which also includes evaluations of multiple wavelet kernels.

3.1.3 Evaluation—Statistics

Table 2 provides RMSE and L^∞ -norm evaluations at multiple compression ratios. These values are all normalized by the range of this data set. The average error (RMSE) is much smaller than the maximal error (L^∞ -norm), with differences as large as three orders of magnitude. Considering that the test data contained more than 68 billion data points ($4,096^3$) and any one of them could set the L^∞ -norm with the largest individual difference, this large disparity between RMSE and L^∞ -norm is not too surprising. Overall, this analysis shows that the average error is quite low, but that the error at a given data point can be much higher.

3.2 Pathline Integration Analysis

3.2.1 Visual Analytics Use Case Description

This evaluation was performed on a simulation of an EF5 tornado [27]. Its output comprised a sequence of 220 time steps depicting the most violent stage of this tornado. The mesh is sized at $490^2 \times 280$, covering a physical space of $14,670 \times$

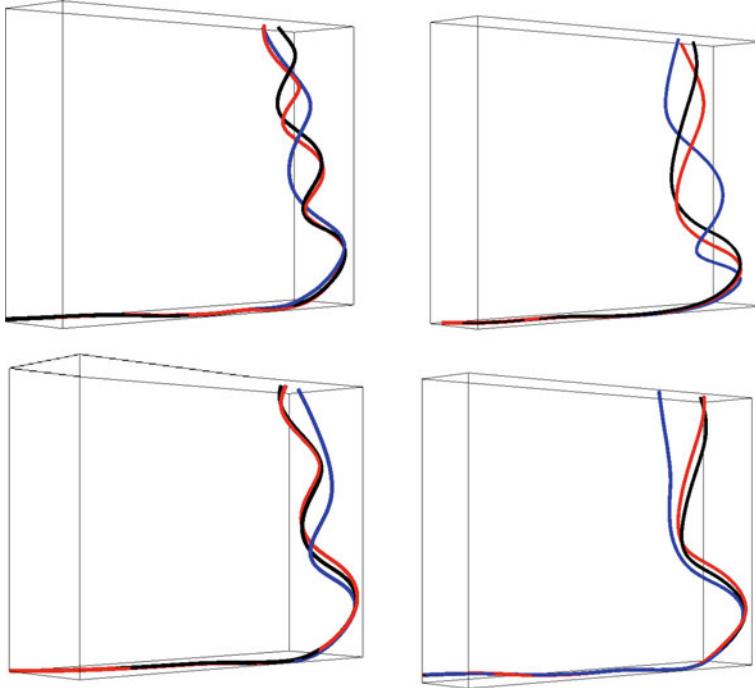


Fig. 5 Each subfigure visualizes the pathlines for a single seed particle being advected using the original version of the data, as well as 3D and 4D compressed versions at the 128:1 ratio. **Black** pathlines are from the original data set; **red** ones are from the 4D compression; and **blue** ones are from the 3D compression. The top two instances show that 4D (red) and 3D (blue) pathlines have similar ending positions, but the 4D ones more closely resemble the baseline (black) for longer durations. The bottom two instances show a clear disadvantage of 3D pathlines (blue) both in terms of early deviation and in terms of far apart final positions. This image is reprinted from our previous work [24]

14,670 × 8,370 m. The visualization task was pathline integration for 144 total particles from different locations in the volume. All particles were advected using the Runge–Kutta 4 method with a step size of 0.01 s. Velocity values between time steps were calculated using linear interpolation.

Compression on this data set was performed on each component of velocity (X, Y, and Z) individually. In addition to different compression ratios (8:1, 32:1, 64:1, and 128:1), two types of wavelet compression were also performed: spatial-only (3D) and spatiotemporal (4D) compression. While the 3D compression was individually performed on all 220 time steps, the 4D compression was performed on groups of time steps, with each group consisting of 18 time steps. This 4D compression configuration is motivated by the coherence between time steps, and the desire to exploit this coherence while performing compression. More details about this 4D compression were provided in the original publication [24]. The total number of

Table 3 Our error metric for each wavelet-compressed data set and deviation tolerance, averaged over all 144 seeds

	D = 10 (%)	D = 50 (%)	D = 150 (%)	D = 300 (%)	D = 500 (%)
8 : 1, 3D	8.5	2.3	1.3	1.1	1.0
8 : 1, 4D	3.4	1.3	1.1	0.8	0.6
32 : 1, 3D	35.9	10.3	4.5	3.1	2.4
32 : 1, 4D	24.4	6.4	3.2	2.1	1.6
64 : 1, 3D	48.4	17.3	7.5	5.3	3.9
64 : 1, 4D	35.7	9.8	5.1	3.3	2.7
128 : 1, 3D	60.7	27.8	10.8	6.7	5.2
128 : 1, 4D	45.8	16.3	7.5	5.1	3.9

versions of this data set was then nine: eight compressed versions (four compression ratios times two compression types), and one original version.

3.2.2 Evaluation—Pathline Deviation

Evaluation involved performing particle advection on all nine versions of the data, and then examining how much the pathlines calculated from compressed data deviate from the ones calculated from the original data. Figure 5 illustrates four example seeds and their resulting pathlines using the original and compressed data.

The metric for quantitative evaluation focused on the trajectory followed by a pathline: let D be a fixed distance, let T be the total time of advection for a particle, and let T_0 be the first time a pathline deviates distance D away from its baseline. Then error was defined as a percentage: $(1.0 - T_0/T) \times 100$. Taking an example: if a particle first deviates distance D from its baseline after six seconds and travels for a total of ten seconds, then its error would be 40%. Five values of D were tested in this evaluation based on suggestions from a domain scientist for this particular data set: $D = 10, 50, 150, 300$, and 500 m .

Table 3 presents results from this evaluation. Each evaluation percentage is averaged over all 144 seed particles. Considering that the deviation thresholds are two to three orders of magnitude smaller than the domain extents, these errors, especially with lower compression ratios, are likely sufficiently small for many visualization tasks. This test also revealed the advantage of 4D compression—with a fixed compression ratio, it consistently yields smaller errors, and it sometimes achieves the same error with a more aggressive compression ratio. This discovery is consistent with the wavelet theory—better compression is achieved by making use of data coherence along multiple dimensions (see discussion in Sect. 2.2.2).

3.3 Shock Wave Front Rendering

3.3.1 Visual Analytics Use Case Description

This study considers the Lulesh [15] proxy-application, a 3D Lagrangian shock hydrodynamics code. Lulesh implements the Sedov test problem, which deposits initial energy at one corner of a cube, and propagates a shock wave from the origin outward to the rest of the cube. While this problem can be run for arbitrarily long periods of time by keeping the propagation going, the phenomenon is most obvious in the first few thousands of time steps. The evaluation, then, considered a time in this initial, interesting period. The variable fields involved in this study include energy (e), relative volume (v), pressure (p), artificial viscosity (q), and the x component of velocity (xd), each on a spatial domain of resolution 128^3 .

Figure 6 presents renderings of the pressure field using the original data (first row), compressed data at various levels (second row), and the image difference caused by compression (third row). While the 16:1 result looks almost indistinguishable from the baseline, the 64:1 result has clear “ripples” along the shock wave front, since that is where most of the energy resides and prone to error. The 128:1 result is even more deteriorated with not only a rough front, but also artifacts in the volume.

3.3.2 Evaluation—Statistics

Evaluation of the efficacy of compression on this data set is done using RMSE and L^∞ -norms. They are measured on all five variables from this simulation, and are normalized by the range of the data. Table 4 reports these statistics. The most interesting finding from this study is likely that the statistics vary vastly from variable to variable. This is because different variables exhibit different levels of data coherence, while a high level of coherence is essential for effective wavelet compression. Another observation is that the difference between RMSE and L^∞ -norm at each configuration is one order of magnitude among all variables and compression ratios, which is in contrast to the two to three orders of magnitude difference observed in Sect. 3.1.3. This is because the smaller dimensions of this data set (128^3 as opposed to the $4,096^3$ in Sect. 3.1.3) are less prone to extreme inaccuracy.

4 Wavelet Compression on High-Performance Computers

This section considers two topics regarding the viability of carrying out wavelet compression on high-performance computers. First, Sect. 4.1 considers how to perform wavelets calculations efficiently on modern computing architectures. Second, Sect. 4.2 considers the overall I/O impact of wavelets.

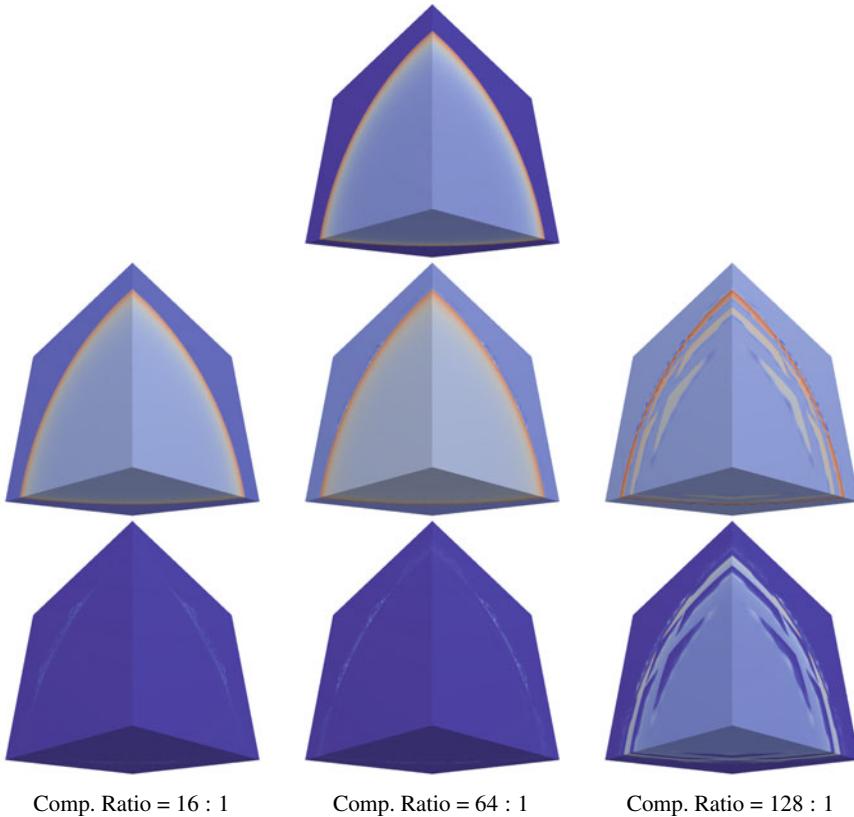


Fig. 6 Renderings of the pressure field from Lulesh depicting the shock wave front. The first row shows the original rendering, and the second row shows renderings from compressed data at different compression ratios. The third row shows rendering difference between those from the original and compressed data, with lighter colors indicating bigger differences. This image is adapted from our previous work [21]

Table 4 Error measurements for each compression ratio. The five data fields evaluated are: energy (e), relative volume (v), pressure (p), artificial viscosity (q), and x-velocity (xd)

		e	v	p	q	xd
16 : 1	RMSE	$6.7e - 8$	$4.1e - 8$	$3.5e - 4$	$2.2e - 4$	$1.1e - 3$
	L^∞ -norm	$9.2e - 7$	$4.7e - 7$	$5.3e - 3$	$4.4e - 3$	$1.5e - 2$
64 : 1	RMSE	$7.7e - 7$	$4.8e - 7$	$4.9e - 3$	$5.8e - 3$	$1.0e - 2$
	L^∞ -norm	$1.8e - 5$	$1.6e - 5$	$1.0e - 1$	$1.2e - 1$	$2.0e - 1$
128 : 1	RMSE	$1.7e - 6$	$1.1e - 6$	$1.0e - 2$	$1.2e - 2$	$1.8e - 2$
	L^∞ -norm	$3.8e - 5$	$2.3e - 5$	$2.6e - 1$	$2.9e - 1$	$3.4e - 1$

4.1 *Wavelets on Modern Computing Architectures*

4.1.1 Portable Performance over CPUs and GPUs

The rise of many-core architectures on supercomputers, particular GPUs, poses both opportunities and challenges for algorithm developers. The opportunities lie in the great performance potential of such architectures, while the challenges lie in the difficulties of harnessing it.

In this subsection, we consider whether wavelets are able to achieve portable performance, i.e., a single code base that can execute efficiently on both CPUs and GPUs with desired performance. This direction follows a line of research by Blelloch [2] in defining code in terms of data-parallel primitives (DPPs). Rather than each iteration of a loop (e.g., `for` or `while`) processing individual data elements, DPP-based programming processes groups of data in parallel. Each data-parallel primitive, such as `map`, `reduce`, `gather`, and `scatter`, provides a unique pattern to process a group of data, and a generic algorithm can then be built upon available DPPs. With this approach, the burden of achieving good performance on a specific hardware architecture (and even running on the architecture itself) shifts to the implementation of the data-parallel primitives. This approach can save developer time when porting to new architectures, since there are many more visualization algorithms (hundreds) than data-parallel primitives (on the order of twenty). Further, the VTK-m project [26] is championing this approach, and providing a framework for algorithm developers to write portably performant code. Returning to wavelets, then, the challenge is to design a wavelet compression algorithm out of data-parallel primitives, which would enable wavelets to run on the variety of many-core architectures found on modern supercomputers.

4.1.2 Wavelet Compression and Portable Performance

This section considers the performance of the wavelet compression filter in the VTK-m framework. This implementation primarily uses two data-parallel primitives: `map` and `sort`. The former is used in the wavelet transform step, where each input data value is “mapped” to an output wavelet coefficient by a sequence of calculations. The latter is used in the coefficient prioritization step, where all coefficients are sorted based on their magnitudes. The algorithm also uses other DPPs for data movement within multi-dimensional wavelet transforms, for example `gather` and `scatter`, but these operations are nowhere near as computationally intensive as `map` and `sort`. More details on the implementation can be found in the study describing this algorithm [22].

To evaluate portable performance of wavelet compression, a study [22] compared the data-parallel primitive-based approach with platform-specific implementations for multi-core CPUs and CUDA GPUs, namely VAPOR [20] for multi-core CPUs,

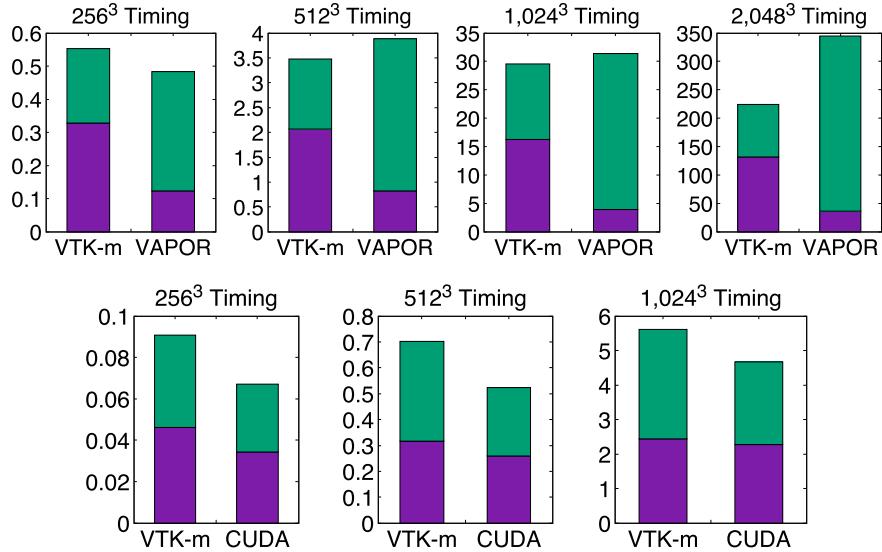


Fig. 7 Execution time comparison (in seconds) between a data-parallel primitive-based implementation and a platform-specific implementation of a wavelet transform on two different architectures. The top row compares execution time between VTK-m and VAPOR implementations on a multi-core CPU. The bottom row compares execution time between VTK-m and CUDA implementations on a GPU. The purple region indicates the time for wavelet transforms, and green is for sorting. The GPU tests had one fewer set of experiments because of memory limitations for large data sets. This image is reprinted from our previous work [22]

and a native CUDA implementation for GPUs. These implementations represented best practices on respective architectures, so they are good comparators.

In the study, the VTK-m implementation was compared to VAPOR on a multi-core CPU system with 32 Xeon Haswell CPU cores at 3.2 GHz, while it was compared to CUDA using an Nvidia Tesla K40 GPU with 2,880 cores at 745 MHz. In both cases, the same test data set is used: a Gaussian distribution with four sizes, 256^3 , 512^3 , $1,024^3$, and $2,048^3$.

Figure 7 shows results of the experiments. In the CPU cases, VTK-m's data-parallel primitive approach was faster, because it was able to sort faster by using highly optimized routines. In the GPU cases, the comparison was not as favorable, primarily because of overheads. On the whole, however, performance was considered to be acceptable. In summary, this study provided three main takeaways: (1) wavelets can be implemented in a hardware-agnostic manner, (2) the resulting code performed well, and thus (3) wavelets can be viably deployed on the varied architectures of modern supercomputers.

4.2 Overall I/O Impact

The purpose of in situ wavelet compression is to reduce the I/O cost for large-scale simulations, both in terms of bytes to store and time to store and load for post hoc analysis. While there is no doubt that less data requires fewer bytes of storage, the time savings are less clear. First, in situ compression does incur additional computational cost, and second, the parallel filesystems on supercomputers also have less intuitive characteristics. All these factors require experiments to test the proposition that in situ compression saves on I/O costs. Such experiments should be on a real supercomputer, and measure the overall cost, which includes time spent both on computation and on actual I/O. This section describes one such set of experiments.

4.2.1 Experiment Overview

The experiment was designed as a weak scaling problem over many compute nodes, meaning that each compute node holds a fixed-sized domain, and the problem grows by adding more compute nodes (and thus domains) into the problem being solved. During every iteration of the execution, every compute node performed simulation, compression, and I/O for its own fixed-sized domain. All compute nodes performed these actions in a synchronized manner, meaning that they were each performing the same stages of tasks at the same time. As more and more compute nodes were added to the execution, more and more concurrent I/O requests were issued to write out larger and larger amounts of data, resembling bursty I/O in real-world applications. This experiment measured the overall I/O with or without in situ compression.

Software used in this experiment included Ascent [18], Lulesh [15], and the wavelet compression implementation using VTK-m described in Sect. 4.1. Ascent oversaw the execution, orchestrating the simulation, in situ compression, data I/O, and communication among all compute nodes. The specifics of the simulation were the same as those used in Sect. 3.3: a shock wave propagation with Lulesh. The experiment was configured to have each compute node holding a domain of size 320^3 , while the shock wave front traveled across domains. Lulesh outputted seven data fields, and in this experiment all data fields were compressed and written to disk.

The supercomputer used in this experiment was Cheyenne [5], the flagship supercomputer of the National Center for Atmospheric Research as of 2020. Every compute node of Cheyenne is equipped with 36 Xeon Broadwell cores and 64 GB system memory. In this experiment, the number of compute nodes used are 1^3 , 2^3 , 3^3 , 4^3 , through 10^3 , reflecting the weak scaling experiment design. The filesystem connected to this cluster is a GPFS parallel filesystem with 200 GB per second bandwidth, which is representative for modern supercomputers. On a final note, this experiment was performed during a lull on the machine in the middle of the night, in an effort to minimize I/O and network contention with other jobs running on the supercomputer.

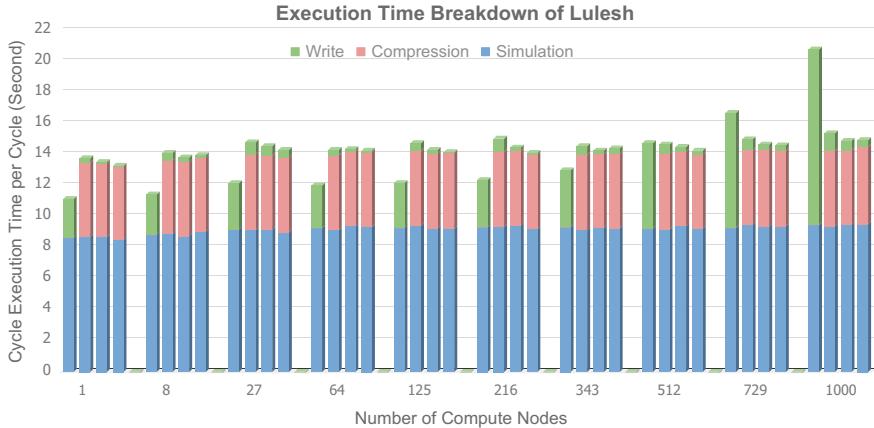


Fig. 8 Execution time breakdown of an in situ wavelet compression experiment with Lulesh. Each group of four uses the same number of compute nodes (X labels), but different compression ratios: no compression, 16:1, 64:1, and 128:1 (from left to right within each group). This image is adapted from our previous work [21]

4.2.2 Experiment Results

Figure 8 presents the execution time breakdown of this experiment, from 1 compute node to 1,000 compute nodes. The in situ compression overhead stayed mostly consistent, taking around 5 s per cycle regardless of compression ratio. With smaller problem sizes (i.e., 1 to 343 domains), this overhead was greater than the I/O time with no compression. As the problem size grew (i.e., greater than 343), the I/O time without compression grew rapidly, making it advantageous to use in situ compression. For this particular supercomputer and this experiment, 343 concurrent write requests were around the point where the parallel filesystem's I/O capacity was saturated, and the I/O bottleneck shifted dramatically to the filesystem itself from the interconnection between compute nodes and the filesystem.

One main conclusion from this experiment was that in situ compression can increase overall I/O time with small problem sizes because of its computational overhead. At the same time, it can save on overall I/O costs with large problems, making wavelet compression a viable in situ operator. Given the current trend of widening gaps between computation and I/O capacities on supercomputers, in situ wavelet compression can be expected to reduce overall I/O costs in more and more use cases. Additional analysis on the experiment results as well as discussion on the viability of in situ wavelet compression is available in the original study [21].

5 Conclusion

This chapter evaluated the applicability of wavelets on supercomputers for post hoc data exploration and scientific visualization. The studies that it summarizes established that this approach is effective, considering two major sub-questions: (1) whether the data integrity is still acceptable after lossy wavelet compression; and (2) whether wavelet compression is viable for in situ compression on supercomputers. With both these sub-questions answered, wavelet compression is well positioned to play the role in between in situ visualization and post hoc analysis: it uses in situ processing to enable human-in-the-loop exploration of data in a post hoc fashion and does it in a manner that minimizes the risk of “losing science.” That said, while wavelets are useful right now, future work could make the technology even more useful for domain scientists. We believe there are two particularly important future directions. The first direction is in strict error bounds while performing compression, which is a desired property of most domain scientists. The second direction is in better understanding of the characteristics and potentials of advanced wavelet codecs, as discussed in Sect. 2.3.3.

Acknowledgements This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977. Computing resources were provided by the Climate Simulation Laboratory at NCAR’s Computational and Information Systems Laboratory (CISL). This work was also supported by the DOE Early Career Award for Hank Childs, Contract No. DE-SC0010652, Program Manager Lucy Nowell.

References

1. Abbate, A., Koay, J., Frankel, J., Schroeder, S.C., Das, P.: Signal detection and noise suppression using a wavelet transform signal processor: application to ultrasonic flaw detection. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control* **44**(1), 14–26 (1997)
2. Blelloch, G.E.: Vector models for data-parallel computing, vol. 75. MIT press Cambridge (1990)
3. Candes, E., Demanet, L., Donoho, D., Ying, L.: Fast discrete curvelet transforms. *Multiscale Model. Simul.* **5**(3), 861–899 (2006)
4. Cohen, A., Daubechies, I., Feauveau, J.C.: Biorthogonal bases of compactly supported wavelets. *Commun. Pure Appl. Math.* **45**(5), 485–560 (1992)
5. Computational and Information Systems Laboratory, National Center for Atmospheric Research: Cheyenne: A SGI ICE XA System (2017). <https://doi.org/10.5065/D6RX99HX>
6. Daubechies, I.: Orthonormal bases of compactly supported wavelets. *Commun. Pure Appl. Math.* **41**(7), 909–996 (1988)
7. Dürst, M.J.: A new method for image compression and progressive transmission. Ph.D. thesis, University of Tokyo (1990)
8. Fowler, J.E.: QccPack: an open-source software library for quantization, compression, and coding. In: Proceedings of SPIE, Applications of Digital Image Processing XXIII, vol. 4115, pp. 294–301. SPIE (2000)

9. Gaither, K.P., Childs, H., Schulz, K.W., Harrison, C., Barth, W., Donzis, D., Yeung, P.K.: Visual analytics for finding critical structures in massive time-varying turbulent-flow simulations. *IEEE Comput. Graph. Appl.* **32**(4), 34–45 (2012)
10. Grossmann, A.: Wavelet transforms and edge detection. In: Stochastic processes in physics and engineering, pp. 149–157. Springer (1988)
11. Guthe, S., Wand, M., Gonser, J., Straßer, W.: Interactive rendering of large volume data sets. In: Proceedings of IEEE Visualization (VIS'02), pp. 53–60. IEEE (2002)
12. Harrison, C., Childs, H., Gaither, K.P.: Data-parallel mesh connected components labeling and analysis. In: Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV), pp. 131–140. Llandudno, Wales (2011)
13. Ihm, I., Park, S.: Wavelet-based 3D compression scheme for interactive visualization of very large volume data. In: Computer Graphics Forum, vol. 18, pp. 3–15. Wiley Online Library (1999)
14. Jawerth, B., Sweldens, W.: An overview of wavelet based multiresolution analyses. *SIAM Rev.* **36**(3), 377–412 (1994)
15. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973 (2013)
16. Kim, B.J., Pearlman, W.A.: An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In: Proceedings of Data Compression Conference (DCC'97), pp. 251–260. IEEE (1997)
17. Kim, T.Y., Shin, Y.G.: An efficient wavelet-based compression method for volume rendering. In: Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications, pp. 147–156. IEEE (1999)
18. Larsen, M., Aherns, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: The alpine in situ infrastructure: ascending from the ashes of strawman. In: Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop, ISAV2017. ACM, New York, NY, USA (2017)
19. Li, S., Gruchalla, K., Potter, K., Clyne, J., Childs, H.: Evaluating the efficacy of wavelet configurations on turbulent-flow data. In: IEEE Symposium on Large Data Analysis and Visualization (LDV), pp. 81–89 (2015)
20. Li, S., Jaroszynski, S., Pearse, S., Orf, L., Clyne, J.: Vapor: A visualization package tailored to analyze simulation data in earth system science. *Atmosphere* **10**(9) (2019). <https://doi.org/10.3390/atmos10090488>. <https://www.mdpi.com/2073-4433/10/9/488>
21. Li, S., Larsen, M., Clyne, J., Childs, H.: Performance impacts of in situ wavelet compression on scientific simulations. In: Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop, ISAV2017. ACM, New York, NY, USA (2017)
22. Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J., Childs, H.: Achieving portable performance for wavelet compression using data parallel primitives. In: Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV). Barcelona, Spain (2017)
23. Li, S., Marsaglia, N., Garth, C., Woodring, J., Clyne, J., Childs, H.: Data reduction techniques for simulation, visualization and data analysis. *Comput. Graph. Forum* **37**(6), 422–447 (2018)
24. Li, S., Sane, S., Orf, L., Mininni, P., Clyne, J., Childs, H.: Spatiotemporal wavelet compression for visualization of scientific simulation data. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 216–227 (2017)
25. Lu, Y.M., Do, M.N.: Multidimensional directional filter banks and surfacelets. *IEEE Trans. Image Process.* **16**(4), 918–931 (2007)
26. Moreland, K., Sewell, C., Usher, W., Lo, L., Meredith, J., Pugmire, D., Kress, J., Schroots, H., Ma, K.L., Childs, H., Larsen, M., Chen, C.M., Maynard, R., Geveci, B.: VTK-m: accelerating the visualization toolkit for massively threaded architectures. *IEEE Comput. Graph. Appl. (CG&A)* **36**(3), 48–58 (2016)
27. Orf, L., Williamson, R., Lee, B., Finley, C., Houston, A.: Evolution of a long-track violent tornado within a simulated supercell. *Bull. Am. Meteorol. Soc.* **98**(1), 45–68 (2017)
28. Pearlman, W.A., Islam, A., Nagaraj, N., Said, A.: Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Trans. Circuits Syst. Video Technol.* **14**(11), 1219–1235 (2004)

29. Pittner, S., Kamarthi, S.V.: Feature extraction from wavelet coefficients for pattern recognition tasks. *IEEE Trans. Pattern Anal. Mach. Intell.* **21**(1), 83–88 (1999)
30. Pulido, J., Livescu, D., Kanov, K., Burns, R., Canada, C., Ahrens, J., Hamann, B.: Remote visual analysis of large turbulence databases at multiple scales. *J. Parallel Distrib. Comput.* **120**, 115–126 (2018)
31. Pulido, J., Livescu, D., Woodring, J., Ahrens, J., Hamann, B.: Survey and analysis of multiresolution methods for turbulence data. *Comput. Fluids* **125**, 39–58 (2016)
32. Said, A., Pearlman, W.A.: Image compression using the spatial-orientation tree. In: *IEEE International Symposium on Circuits and Systems (ISCAS'93)*, pp. 279–282. IEEE (1993)
33. Sardy, S., Tseng, P., Bruce, A.: Robust wavelet denoising. *IEEE Trans. Signal Process.* **49**(6), 1146–1152 (2001)
34. Shapiro, J.M.: Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Process.* **41**(12), 3445–3462 (1993)
35. Shneiderman, B.: A grander goal: a thousand-fold increase in human capabilities. *Educom Rev.* **32**(6), 4–10 (1997)
36. Skodras, A., Christopoulos, C., Ebrahimi, T.: The JPEG2000 still image compression standard. *IEEE Signal Process. Mag.* **18**(5), 36–58 (2001)
37. Tang, X., Pearlman, W.A., Modestino, J.W.: Hyperspectral image compression using three-dimensional wavelet coding. In: *Electronic Imaging 2003*, pp. 1037–1047. International Society for Optics and Photonics (2003)
38. Tang, Y.Y.: Wavelet theory and its application to pattern recognition, vol. 36. World Scientific (2000)
39. Taubman, D.: High performance scalable image compression with EBCOT. *IEEE Trans. Image Process.* **9**(7), 1158–1170 (2000)
40. Wang, C., Shen, H.W.: A framework for rendering large time-varying data using wavelet-based time-space partitioning (wtsp) tree. Technical Report OSU-CISRC-1/04-TR05 (2004)
41. Wei, D.: Coiflet-type wavelets: theory, design, and applications. Ph.D. thesis, University of Texas at Austin (1998)
42. Zhang, L., Bao, P.: Edge detection by scale multiplication in wavelet domain. *Pattern Recognit. Lett.* **23**(14), 1771–1784 (2002)

In Situ Statistical Distribution-Based Data Summarization and Visual Analysis



Soumya Dutta, Subhashis Hazarika, and Han-Wei Shen

Abstract As the era of exascale computing approaches, the need for effective, scalable, and flexible data reduction techniques is becoming more and more prominent. As discussed in the introductory chapter, this need is primarily due to the bottleneck stemming from output data size and I/O speed compared to the ever-increasing computing speed. With this chapter, we consider a promising solution: data summarization techniques that work in the in situ environment while the data is getting produced, and preserve the important information from the data compactly, which minimizes information loss and enables a variety of post hoc analyses. Specifically, this chapter shows that statistical distribution-based in situ data summaries are a pragmatic solution and able to preserve important statistical data features. Using only the in situ generated statistical data summaries, which is significantly smaller in size compared to the original raw data, a wide range of data analysis and visualization tasks can be performed such as feature detection, extraction, tracking, query-driven analysis, etc. In addition, reconstruction of the full-resolution data is also possible, in order to visualize the data in its entirety with the added advantage of uncertainty quantification. To this end, this chapter presents several distribution-based data modeling algorithms, considering both their in situ performance and the usefulness of their distribution data summaries on several application studies.

1 Statistical Distribution Models for Data Summarization

Probability distributions are well known for capturing various statistical properties of data sets. Furthermore, since these distributions are capable of representing a

S. Dutta (✉) · S. Hazarika
Los Alamos National Lab, Los Alamos, NM, USA
e-mail: sdutta@lanl.gov

S. Hazarika
e-mail: shazarika@lanl.gov

H.-W. Shen
The Ohio State University, Columbus, OH, USA
e-mail: shen.94@osu.edu

large set of data samples in a compact format, they have been used successfully for modeling scientific data sets and, as a result, different types of distribution-based data summaries have been proposed as a means of reduced data representation. Before we go into the details of modeling large-scale simulation data using distributions, let us first briefly discuss several statistical distribution representations that have been used in the data science and visualization community for summarizing large-scale data sets. Distribution-based modeling techniques can be classified into two broad categories: (1) Non-parametric distribution models; and (2) Parametric distribution models. Histogram and Kernel Density Estimators (KDE) are popular non-parametric distribution models used extensively in the visualization community, whereas, parametric distributions, such as Gaussian distributions or Gaussian Mixture Models (GMM), have also been found to be very effective in data analysis. In the following, we briefly introduce the most popular distribution models that used in various in situ applications and discuss their advantages and disadvantages in the in situ context.

1.1 Non-parametric Distribution Models

Given a set of discrete data samples $\{x_i\}$, a non-parametric distribution in the form of a histogram can be formally defined as:

$$H(s) = \sum_i \delta(x - x_i) \quad (1)$$

where δ is the Dirac delta function defined as:

$$\delta(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The area under a histogram can be normalized, and such histograms are often used as a discrete probability distribution function. Another well known non-parametric distribution model Kernel Density Estimator (KDE) is defined as:

$$f(x) = \frac{1}{nb} \sum_{i=1}^n K\left(\frac{x - x_i}{b}\right) \quad (3)$$

where $f(x)$ denotes the probability density at x , n is the number of data samples, $b (> 0)$ is the bandwidth, a smoothing parameter, and $K(\cdot)$ is the non-negative kernel function. A range of kernel functions such as uniform, triangular, Gaussian, and Epanechnikov kernels can be used for estimating data density.

1.2 Parametric Distribution Models

Compared to non-parametric distribution models, parametric distribution models offer a more compact distribution representation, since only the parameters of the models are sufficient to represent the distribution model. The use of parametric Gaussian distributions for data modeling is widely known across various scientific domains. However, the assumption of the normality of data is not always true and can introduce modeling inaccuracies. Gaussian mixture models (GMM) remove this assumption of normality by modeling the data as a convex combination of several Gaussian distributions. The storage footprint for a GMM consists of the parameters of the Gaussian distributions and their weights. Formally, the probability density $p(X)$ of a Gaussian mixture model for a random variable X can be written as:

$$p(X) = \sum_{i=1}^K \omega_i * \mathcal{N}(X|\mu_i, \sigma_i) \quad (4)$$

where K is the number of Gaussian components, and ω_i , μ_i and, σ_i are the weight, mean, and standard deviation for the i th Gaussian component respectively. Note that the sum of weights in the mixture, $\sum_{i=1}^K \omega_i$, is always equal to 1. The computation of parameters for the GMMs is typically done by Expectation-Maximization (EM), which uses an iterative approach to maximize a likelihood function [5]. For an approximate and computationally efficient estimation of parameters of a GMM, an alternative incremental estimation scheme is also available [32, 35] which can satisfy the need of fast-processing in the in situ environment.

1.3 Advantages and Disadvantages of Different Distribution Models in the Context of In Situ Data Reduction

To use distributions as a viable solution for performing in situ data summarization, several constraints need to be discussed. Any in situ data analysis algorithm is expected to be computationally fast so that it does not stall the underlying scientific simulation, and also the additional memory requirement should be as small as possible. In this context, the computation time for the non-parametric model histogram is low as it only requires a scan of data values and counting the frequencies of discretized data values by converting them into bins. However, the storage requirement of histograms is not always small since the frequency of each bin needs to be stored. Furthermore, if summarization is performed on multivariate data, then the storage footprint of high-dimensional histograms increases exponentially. The other widely used non-parametric model KDE is computationally more expensive than histograms. Compared to the non-parametric models, the storage requirement of the parametric distributions is always small since only the model parameters are

stored. While computation time for estimating parameters for a Gaussian distribution is low, estimation of model parameters can be expensive at times for high-dimensional parametric distributions such as Gaussian mixture models. Therefore, instead of using the traditional Expectation-Maximization (EM) algorithm all the time for GMM parameter estimation, a faster and approximate incremental mixture model estimation algorithm has been explored for in situ GMM-based data summarization [12]. Another important point is that using only one type of distribution model for all the data may not be the optimal modeling strategy. For example, based on the statistical properties of the data, different distribution models might be suitable at different regions of the data. Therefore, a hybrid distribution-based data summarization would be possible where, based on various statistical tests, the most suitable distribution models will be used for summarization [21, 22]. In the following, we briefly discuss various statistical tests that can be done to pick the most suitable distribution model for data summarization.

1.3.1 Various Statistical Tests for Picking the Suitable Distribution Model

Depending on factors like initial data size, type of post hoc analyses targeted and/or in situ computational complexity involved, both parametric and non-parametric models have distinct advantages and disadvantages. The choice of suitable distribution model, therefore, plays an important role in determining the efficiency of distribution-based in situ data summarization strategies. Many standard statistical tests currently exist to decide which distribution model can best represent the underlying data. However, often a single test may not be enough to address all the concerns and trade-offs associated with a real-world in situ scenario. Therefore, users have to carefully design their tests based on their requirements. Depending on the application and scale of operation, the task of selecting a distribution model can be as simple as graphical validation of the shapes of distributions to as complex as solving an optimization function with desired requirements as the function variables. Here, we put forward some of the most commonly used practices prevalent in the fields of Statistics and Visualization.

Normality Test: One of the simplest, yet effective, statistical tests that can be performed is to check for Gaussian/normal behavior in the data distribution. Studies have shown that for the same sample size, the Shapiro-Wilk test [33] is the most powerful (i.e., *statistical power*¹) normality test [31]. It returns a likelihood value, commonly referred to as *pValue*, which lies between 0 and 1. Small *pValues* lead to the rejection of the normality hypothesis, whereas, a value of 1 ascertains normality with high confidence. A *pValue* in the range of [0.05, 0.1] is often considered as a good threshold value to decide normality. For data not satisfying this threshold further evaluations need to be done to find a more suitable distribution model.

¹ *Statistical power* of any test is defined as the probability that it will reject a false null hypothesis.

Goodness-of-fit Test: Normality tests do not offer a means to evaluate the best-fitted distribution model for the underlying data out of possible candidate models. The Kolmogorov-Smirnov (KS) test [36], a type of goodness-of-fit test, is a more generic platform for such comparative validation. It compares the cumulative density function (CDF) of a reference distribution against the empirical CDF (ECDF) of data. Goodness-of-fit is decided by how close the CDF of a distribution is to the ECDF. If $F(x)$ represents the CDF of the hypothesized distribution and $F_e(x)$ represents the ECDF, then the KS test measure is given as,

$$K = \sup_x |F(x) - F_e(x)| \quad (5)$$

Unlike many other statistical tests, the KS test can evaluate the goodness of both parametric and non-parametric distributions at the same time.

Bayesian Information Criterion: Bayesian Information Criterion (BIC) [16] is a commonly used metric for selecting among a finite set of parametric models. It is based on the log-likelihood of a given model on the sample data. It is defined as,

$$BIC = -2L_p + p \log(n) \quad (6)$$

where n is the sample size, L_p is the maximized log-likelihood of the chosen model and p is the number of parameters in the model. A low BIC value indicates a better model. BIC attempts to address the risk of over-fitting by introducing a penalty term $p \log(n)$, which grows with the number of parameters. Therefore, the BIC score is designed to avoid overly complicated models (i.e., with a large number of parameters), which is ideal for distribution-based in situ data summarization approaches. The BIC test is often used for finding out the correct number of Gaussian distributions that would best fit a sample while modeling using a Gaussian mixture model [37, 38].

2 In Situ Distribution-Based Data Summarization Techniques

One of the primary advantages of using distribution-based summaries is that the distributions can capture various statistical properties of the data robustly in a compact format. Therefore, in the absence of the full resolution raw data, during post hoc analysis, a variety of data analysis and visualization tasks can be carried out using such distribution-based data. Furthermore, while the generated results will have uncertainties as the full resolution data is not available, distribution-based data summaries will allow uncertainty quantification for the produced results by conveying uncertainty information to the application scientists.

While modeling scientific data sets using distributions, one can use a global distribution for the whole data domain. For example, a one-dimensional histogram

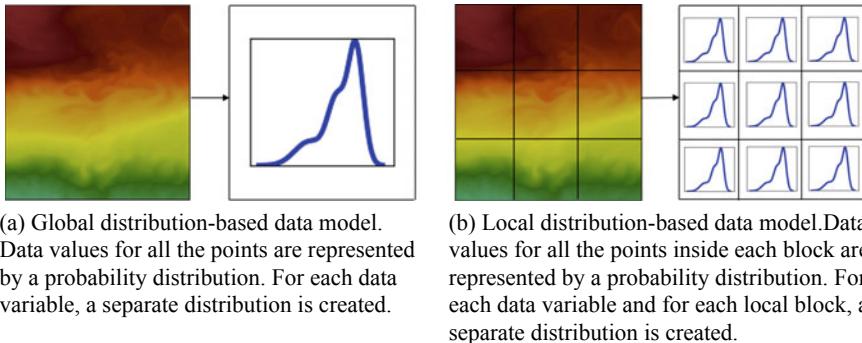


Fig. 1 Illustration of Local and Global distribution-based data modeling schemes. This image is reprinted from our previous work [11]

can be used to model a data variable in a data set. In this case, the histogram will be able to answer questions regarding the likelihood of specific values of the scalar field in the data [6, 13, 20], but will not be able to answer questions such as where those specific values occur in the domain. This is because the global data distribution is only a coarse statistical summarization of the complete data domain and does not capture any spatial information. Hence, even though significant data reduction using global distribution models can be achieved, still, the applicability and flexibility of such global distribution-based data summaries during the post hoc analysis phase is minimal (Fig. 1).

In contrast, to capture the data properties in much finer detail for enabling detailed visual analysis, local region-based distribution modeling techniques have shown great success. In this case, the data domain is first divided into smaller regions/blocks and then a suitable distribution is used to model the data for each region. In this way, even though the storage footprint increases compared to the global model, such a local model-based summarization can capture the statistical properties of the data in much more detail compared to the global distributions. In the following, we introduce different schemes of in situ local distribution-based data summarization in detail.

2.1 Local Distribution-Based In Situ Data Summarization

As discussed above, the local distribution-based summarization techniques divide the data domain into smaller regions and then use suitable distribution models to reduce the data at each local region. If variables are summarized individually, then univariate distribution models are used. When relationships among multiple variables are required to be captured in the distribution-based data summaries, multiple variables are summarized together using multivariate distribution modeling techniques. Whether univariate modeling is sufficient or multivariate data summaries

are needed depends on the specific application tasks. As an in situ data summarization technique, while the univariate distribution-based modeling has its own challenges, multivariate distribution-based modeling techniques are significantly more complex as both the computation cost and storage footprint increase significantly. Therefore, sophisticated distribution modeling schemes are often preferred over standard multivariate distribution-based modeling techniques to address such issues. In the next section, we first discuss various local distribution-based univariate data summarization techniques and then introduce multivariate data summarization schemes.

2.1.1 Distribution-Based Summarization for Univariate Data

Individual data variables can be summarized compactly using univariate distribution-based models such as univariate histograms, Gaussian distributions, GMMs, etc. An important advantage of using the local region-based modeling approach is that it allows modeling of the local statistical properties of the data in detail and therefore, the generated distribution data summaries are flexible and can address a wide range of analysis and visualization tasks in the post hoc exploration phase. Firstly, the data domain is divided into smaller sub-regions (data blocks), and then the desired data variables in each region are summarized using separate univariate distribution models. Secondly, all the region-wise distributions are stored into the disk for post hoc analyses.

A straightforward data domain decomposition scheme used in the literature is regular non-overlapping blocks-wise partitioning. Regular partitioning based data decomposition is computationally less expensive as well as storage efficient. However, since regular partitioning does not consider any data properties, the resulting distribution-based models generated from the data at each partition often show high value variance, and consequently high uncertainty. Furthermore, the distribution summaries only capture the statistical properties of the data values and the spatial organization of such data values inside each block is not preserved in the univariate distribution. Therefore the naive regular partitioning scheme is limited in application in the post hoc analysis phase. To remedy this issue and capture spatial information from the local univariate distribution-based data summaries, two approaches can be taken: (1) By augmenting the spatial distribution information directly to the regular block-wise data summaries; (2) Instead of using regular partitioning, irregular partitioning schemes can be used where spatially contiguous similar data values will be grouped and the distribution-based analysis error will be reduced. Below we briefly present these two approaches.

A

- **Spatial Distribution-augmented Statistical Data Summarization** An explicit approach of capturing spatial information into distribution-based data summaries is the direct augmentation of spatial distribution data summaries with value distribution based data summaries. In this case, the data is still partitioned using regular blocks. Then, for a selected data variable that needs to be summarized, the

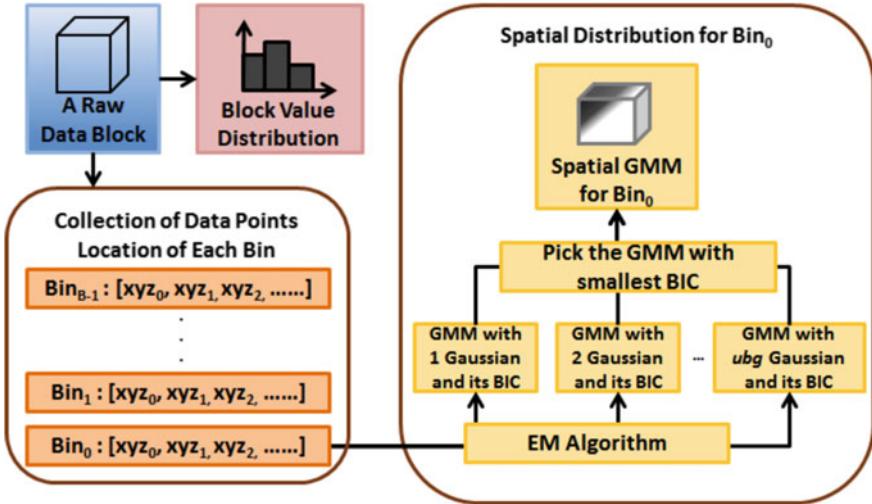


Fig. 2 This diagram shows the steps used to compute the spatial GMM for a raw data block (shown in blue). Besides the computation of the value distribution, the raw data in the block is used to construct the Spatial GMM. First, the locations of the data samples are collected into the corresponding bin interval according to the data value at that location (shown in the bottom left). Then, a Spatial GMM is constructed (shown on the right) for each bin interval using the locations in the interval (illustrated here for Bin₀). This image is reprinted from our previous work [37]

data values of each partition are first summarized using a value-based histogram as shown in Fig. 2 (the pink box). Next, to incorporate the spatial information to this value distribution, a spatial Gaussian mixture model (GMM) is estimated for the data points in each histogram bin. For each bin of the value histogram, the data points are identified and then, using their spatial locations, a multivariate GMM, termed as spatial GMM, is estimated. Each histogram bin is associated with its unique spatial GMM. While estimating the spatial GMMs, the suitable number of modes for each spatial GMM can be identified by applying Bayesian Information Criterion (BIC) as has been illustrated in Fig. 2. Therefore, for each data block, using this approach, a value histogram and a set of spatial GMMs are stored as the reduced summary data.

Exploration using this spatial GMM augmented distribution-based data summaries is done post hoc. While inferring the data value at a queried location, information from the value histogram and the spatial GMMs are combined using Bayes' rule. Bayes' rule is a popular theorem that is widely used in classification problems. It tells us how to augment the known information with additional evidence from a given condition. In this case, the block value distribution is the known information and the additional evidence are the probabilities from each Spatial GMM at the queried location. More details of this technique can be found in [37].

- **Homogeneity-guided Data Partitioning and Summarization** A second implicit approach for capturing spatial information in distribution-based modeling and

reducing error during post hoc analysis is the use of irregular data-driven partitioning techniques. Naive regular partitioning of the data domain does not consider data continuity, and as a result, produces partitions with high data value variance. When distributions are used to reduce such partitions, the resulting distribution models contain high data value variance. Consequently, post hoc analysis using such distribution summaries produces high sampling error leading to increased uncertainty. Therefore, to reduce the data value variation in the partitions, a supervoxel generation algorithm SLIC (Simple Linear Iterative Clustering) [2] is used. SLIC is a computationally efficient variant of the local K-means clustering algorithm and produces spatially connected data clusters, which are homogeneous in nature. Each SLIC cluster/supervoxel is treated as a local data partition and is summarized using an appropriate distribution model. Since the data are partitioned into near homogeneous regions, using normality tests often it is found that a single Gaussian distribution is sufficient to capture the statistical data properties of each partition. When a single Gaussian is not sufficient, a GMM can be used for summarization. A detailed description of this hybrid distribution-based in situ data summarization scheme can be found in [15].

Compared to the traditional K-means clustering, SLIC adopts a local neighborhood-based approach, where similar data points within a local neighborhood are grouped into one cluster. During the optimization stage, from each cluster center, distances only to the points in the predefined neighborhood are compared. This reduces the total number of distance computations significantly by limiting search in a local neighborhood. As a result, the algorithm performance is boosted significantly. Furthermore, SLIC uses a weighted distance measure that provides contributions from both the spatial locality of the data points and their scalar value similarities. The distance measure can be defined as:

$$D(i, j) = \beta \cdot \|c_i - p_j\|_2 + (1 - \beta) \cdot |val_i - val_j| \quad (7)$$

Here, c_i is the location of the cluster center i and p_j is the location of point j . val_i and val_j are the data values at i th cluster center and j th data point respectively. The mixing weight β is configured based on the importance of spatial versus value components, such that $0 \leq \beta \leq 1$, and $\beta + (1 - \beta) = 1$. Smaller values of β will give higher weightage on the difference of data values than their spatial locations. Due to these properties, SLIC partitions the data domain into smaller sub-regions where each partition contains points which are: (a) spatially as contiguous as possible; (b) homogeneous in value domain. In Fig. 3b, we show an illustrative example of the SLIC algorithm applied on a 2D data. As can be seen, SLIC partitions similar valued data points along non-axis aligned boundaries compared to the regular partitioning scheme shown in Fig. 3a.

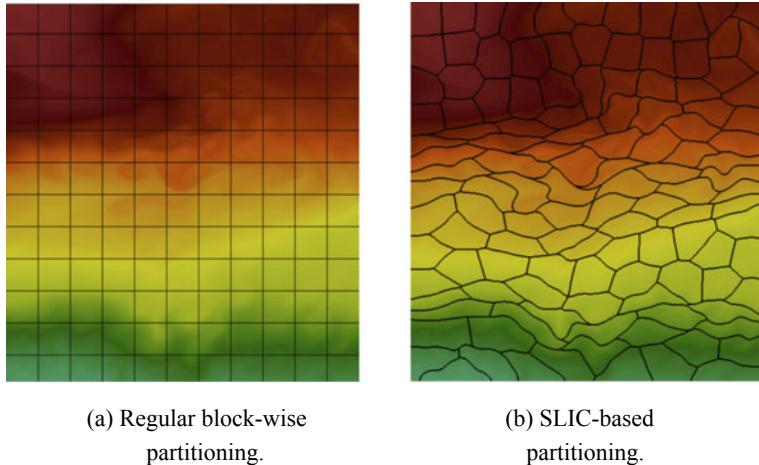


Fig. 3 Different types of data partitioning schemes. This image is reprinted from our previous work [11]

2.1.2 Distribution-Based Summarization for Multivariate Data

Many times, scientific simulations produce multiple physical variables/attributes (like pressure, temperature, precipitation, etc.) at the same time. These variables are used to perform various multivariate analyses to gain in-depth insights into the underlying physical phenomenon. Therefore, instead of modeling individual variables as independent univariate distributions, it is often necessary to model them together as multivariate distributions in order to preserve the variable inter-dependence. However, the benefits of distribution-based data summarization are not always readily applicable when using standard multivariate distributions. Unlike their univariate counterparts, it becomes increasingly difficult to work with the corresponding standard multivariate distribution representations when the number of variables (i.e., dimensionality) increases. In this section, we discuss the challenges associated with multivariate distribution-based data summarization and pragmatic solutions to address them.

- **Multivariate Histogram.** Compared to univariate histograms (Sect. 1.1), computing and storing multivariate histograms is a non-trivial task. The storage footprint of a multivariate histogram can increase exponentially with the number of variables and the desired level of discretization (i.e., number of bins). This makes them ineffective for the purpose of in situ data reduction. Sparse representations of the multivariate histograms can be constructed to bring down the exponential storage cost [29]. Based on the sparseness of the multivariate histogram, the large multi-dimensional array can be transformed into a much smaller size. This transformation, encoded with dictionary-based data structures, can be used to map the transformed multi-dimensional array back to the original array. To further

reduced the storage overhead, the multivariate histogram can be stored as a sequence of the index and frequency pairs where the indices are represented as bitstrings computed from a space-filling curve traversal of the multi-dimensional array. However, such sparse representations are sensitive to how the data is distributed and the number of histogram bins used. Therefore, despite cutting down the exponential storage cost of multivariate histogram representations, they are not always effective for data reduction when compared with the original size of the raw data.

- **Multivariate GMM.** As discussed in Sect. 1.2, because of their compact representation and good modeling accuracy, univariate GMMs are frequently used for distribution-based data summarization. Their multivariate counterparts can also be represented by Eq. 4 above, with multivariate Gaussian kernels instead of univariate Gaussians. However, the estimation of multivariate GMM using Expectation-Maximization is computationally expensive compared to the univariate GMMs. The computation time and model complexity increase rapidly with the number of variables. As a result, the in situ estimation of multivariate GMMs will only add to the overall simulation execution time for large-scale simulations. This can overshadow the advantages of data reduction and I/O bottleneck alleviation for distribution-based data summarization.
- **Copula-based Multivariate Distribution Modeling.** Given the challenges associated with standard multivariate distribution models, it is important to take a fresh look at modeling multivariate distributions for in situ analysis. One such way is to use copula functions [22]. Copula functions offer a statistically robust mechanism to decouple the process of multivariate distribution estimation into two independent tasks: *univariate distribution estimation* and *dependency modeling*. As a result, the exponential cost of storage and/or distribution estimation time can be reduced significantly because we can independently model the individual variables using arbitrary univariate distribution types, while the copula function captures the dependency among them separately.

A copula function is a multivariate distribution function, whose univariate marginals are standard uniform distributions. In terms of cumulative density functions (CDF), $C : [0, 1]^d \rightarrow [0, 1]$ represents a d -dimensional copula (i.e., d -dimensional multivariate CDF) with uniform marginals. Sklar's theorem [34] stated that every joint CDF in \mathbb{R}^d implicitly consists of a d -dimensional copula function. If F is the joint CDF and F_1, F_2, \dots, F_d are the marginal CDF's for a set of d real valued random variables, X_1, X_2, \dots, X_d respectively, then Sklar's theorem can be formally represented as;

$$\begin{aligned} F(x_1, x_2, \dots, x_d) &= C(F_1(x_1), F_2(x_2), \dots, F_d(x_d)) \\ &= C(u_1, u_2, \dots, u_d) \quad (\text{using } F_i(x_i) = u_i \sim U[0, 1]) \end{aligned} \tag{8}$$

where the joint CDF F is defined as the probability of the random variable X_i taking values less than or equal to x_i . Therefore, to model any multivariate distribution using a copula-based strategy, we need the following two sets of information.

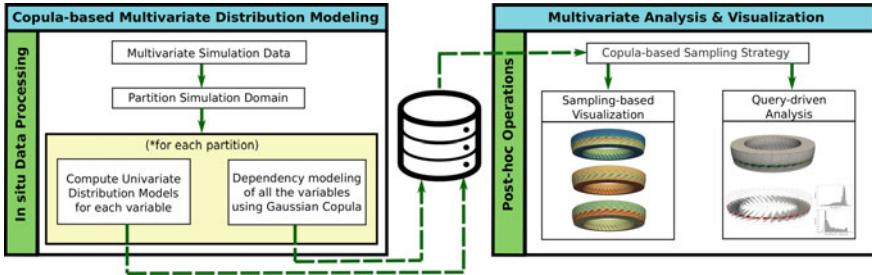


Fig. 4 Overview of a copula-based in situ multivariate data summarization workflow. Multivariate data summaries are created in situ using Gaussian copula functions. The summaries can be utilized later to perform different multivariate post hoc analysis and visualization tasks. This image is reprinted from our previous work [22]

1. Univariate marginal distributions of the individual variables (i.e., F_i 's).
2. A copula function that captures the dependency among the variables (i.e., C).

Copula-based multivariate distribution modeling techniques generally approximate the function $C(\cdot)$ using standard copula functions [30]. One such popular copula function is the Gaussian copula, which is derived from a standard multivariate normal distribution. For the purpose of data reduction, the Gaussian copula is ideal because it requires the storage of only the correlation matrix, which can be efficiently computed in an in situ environment. Using this flexible multivariate distribution modeling approach, for each local region, we can store the multivariate data summaries (comprising of univariate distributions and a copula function) to achieve multivariate relationship-aware in situ data reduction. Figure 4 provides a schematic overview of a copula-based in situ multivariate data summarization workflow. The summaries can be utilized to carry out various multivariate post hoc analyses.

3 Post Hoc Visual Analyses Using Distribution-Based Data Summaries

One of the primary requirements of any in situ data summarization technique is to be flexible during post-hoc analysis so that a variety of visualization and analysis tasks can be performed using it. Since data analysis algorithms are often constrained by storage and computation cost in the in situ environment, a majority of the exploration tasks are still preferred to be done post hoc by the application scientists where they can refine the analysis results interactively, change search criteria as new information is learned, and visualize the data on demand. In this section, we discuss how the various types of aforementioned in situ distribution-based data summaries can be used to enable a wide range of analyses tasks in the post hoc exploration phase.

3.1 Stochastic Feature Analysis

Analysis and visualization of various scientific features in simulation data sets is one of the primary tasks that application scientists perform routinely. Distribution-based data summaries can be used to carry out this task robustly. By representing the user specified target features in the form of a distribution, such features can be searched for in distribution-based data summaries and the features can be extracted and visualized. Feature extraction can be done by matching the target feature distribution to the in situ generated distributions of the local regions and all the regions with a high similarity can be explored interactively. In Fig. 5 an example of distribution-based feature extraction is shown. This example uses the homogeneity-guided SLIC-based data partitioning scheme and the data for SLIC partitions is summarized using univariate GMM-based modeling. As can be seen in Fig. 5c, the SLIC-based data summaries are able to model the data accurately and hence the extracted features does not have discontinuity and artifacts which are visible from the results generated using a naive regular partitioning scheme (Fig. 5a), and also in a K-d tree based partitioning scheme (Fig. 5b). More results, and a comprehensive quantitative study of this technique, can be found in our previous work [15].

Another example of post hoc feature exploration using the spatial distribution augmented data summaries is shown in Fig. 6. Given a target feature, a new feature similarity field is generated where the high valued regions are highlighted as the regions of interest. The true similarity field is shown in Fig. 6a, which is generated using the ground truth raw data for comparison purposes. Figure 6b shows the similarity field generated using regular block-wise partitioning and GMMs are used as the distribution model. Finally, Fig. 6c depicts the feature similarity field resulted from spatial GMM augmented data summaries. As can be seen, the spatial GMM based data summaries produce the most accurate feature similarity field with minimal artifact [37].

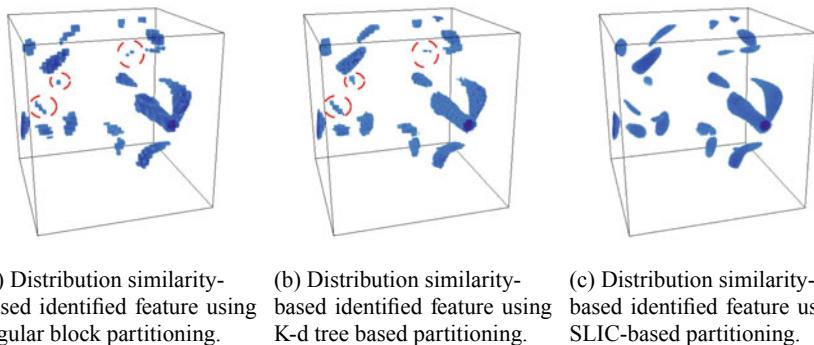


Fig. 5 Distribution data-driven probabilistic feature search using SLIC-based data summaries in the Vortex data set. This image is reprinted from our previous work [15]

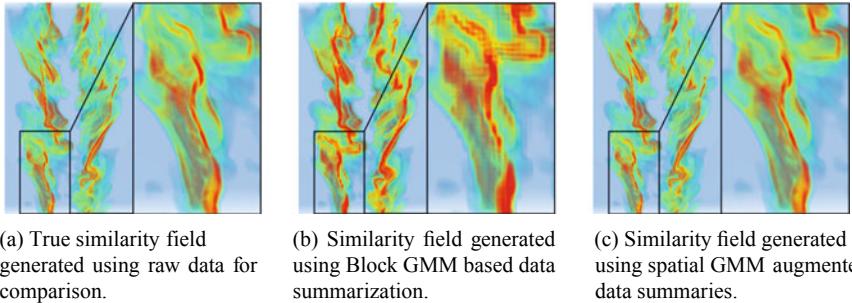


Fig. 6 Distribution-based feature similarity field visualization using spatial GMM based data summaries in Turbulent Combustion data set. This image is reprinted from our previous work [37]

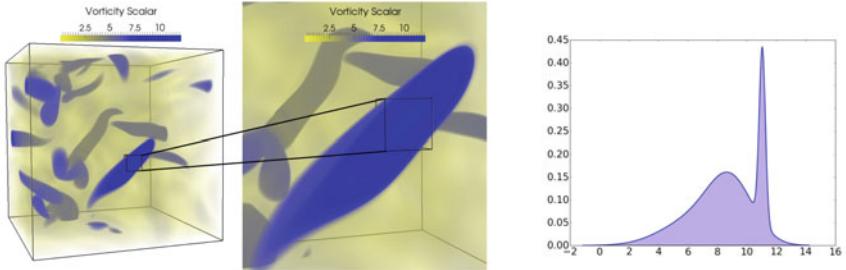
3.2 Feature Extraction and Tracking

Feature tracking in scientific data sets is an important task. Application scientists are often interested in extracting and tracking the temporal evolution of scientific features (a region of interest) such as vortex cores, hurricane eye, eddies in an ocean, etc., to learn about the temporal development of various physical phenomenon in detail. The proposed distribution-based data summaries can be used to track such scientific features robustly over time. In this study, a regular block-wise distribution modeling is used where a parametric distribution Gaussian mixture model is used to model the data for each local block. Since features in scientific simulations are often hard to define with a precise descriptor, a value-based distribution is used to represent the target feature. Finally, using stochastic similarity measures and extracted motion information from the distribution-based data summaries, the feature is extracted and tracked over time robustly. More details of this distribution-driven feature tracking algorithm be found in an article on this topic [14].

In Fig. 7, target feature selection in the form of a GMM is displayed where a user can highlight a region of interest using an interactive box filter. The feature shown here is a vortex core in a pseudo-spectral simulation of coherence vortex structures. The tracking results of this feature is provided in Fig. 8 where the tracked vortex feature is shown for four different time steps. Note that, even though the shape of the feature changes over time, the tracking algorithm is still able to extract and track the feature robustly in future time steps.

3.3 Multivariate Query-Driven Analysis and Visualization

Query-driven analysis techniques are highly effective for analyzing and visualizing large scale data. By selecting a subset of the data domain that meets a user-defined criteria, analysis activities can be focused only on the selected region instead



(a) Feature selection in the Vortex data set with a zoomed in view.

(b) Estimated GMM of the user interested feature.

Fig. 7 Selected feature in the Vortex data set, a zoomed in view and the GMM of the selected region. This image is reprinted from our previous work [14]

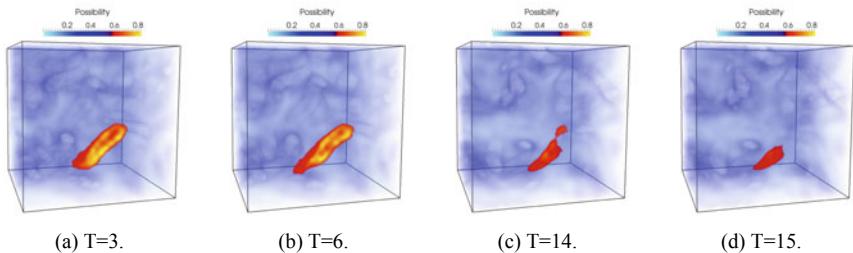


Fig. 8 Extraction and tracking using the Vortex data set. Tracked feature for 4 selected time steps are displayed. This image is reprinted from our previous work [14]

of considering the entire domain. This makes the workflow of scientists more manageable and effective. These type of selective analyses are particularly common with multivariate data to trim down the variable subspace. Many query-driven strategies rely on computing local data statistics to execute the queries efficiently. Therefore, the use of statistical distributions as local data summaries inherently facilitates such query-driven strategies. With distributions as the underlying data representation, we can report queried region as a probability field. A high probability value at a region signifies that the query of interest has high likelihood at that region. Figure 9 shows the query results on the Combustion data set for the bi-variate query $0.3 < mixfrac < 0.7$ and $y_oh > 0.0006$. Figure 9a shows the ground truth deterministic region of the original raw data, while Fig. 9b shows the corresponding probability field satisfying the given query, i.e., $P(0.3 < mixfrac < 0.7 \text{ AND } y_oh > 0.0006)$.

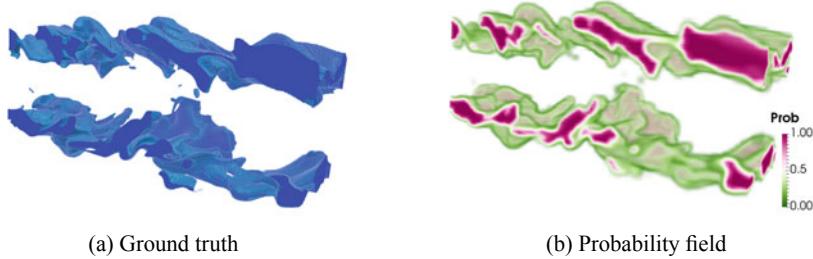


Fig. 9 Multivariate query-driven analysis of the Combustion data set for the query $0.3 < mixfrac < 0.7$ and $y_oh > 0.0006$. This image is reprinted from our previous work [22]

3.4 Distribution Sampling-Based Data Reconstruction

Application scientists often want to visualize their data in its entirety to explore certain data features in detail. To enable visualization on full resolution data, distribution-based data summaries can be used to reconstruct this data. To reconstruct the data, statistical sampling techniques [17] are used to sample data values from distribution-based data summaries. In the following, we present reconstruction results created from various types of distribution-based data summaries for both univariate and multivariate data.

Figure 10 shows the reconstruction result for the U-velocity field of the Hurricane Isabel data set. In this example, GMM-based data summaries were generated from the in situ SLIC-based partitioning scheme [15]. For comparison, in Fig. 10b, we have shown the reconstruction result when a regular block-wise partitioning scheme is used. As the image shows, the reconstructed result produced from the SLIC-based data summaries (Fig. 10c) match closest to the ground truth shown in Fig. 10a. The result of regular block-wise partitioning contains artifacts and discontinuities (as highlighted by black dotted regions), which are corrected in reconstruction obtained from SLIC-partitioning based data summaries.

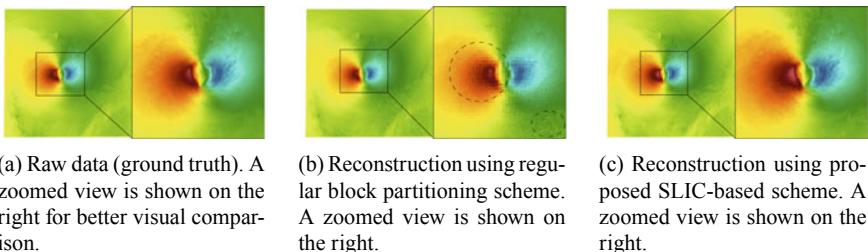


Fig. 10 Visual comparison of U-velocity of Hurricane Isabel data. The reconstructed fields are generated using Monte Carlo sampling of distribution-based summarized data. This image is reprinted from our previous work [15]

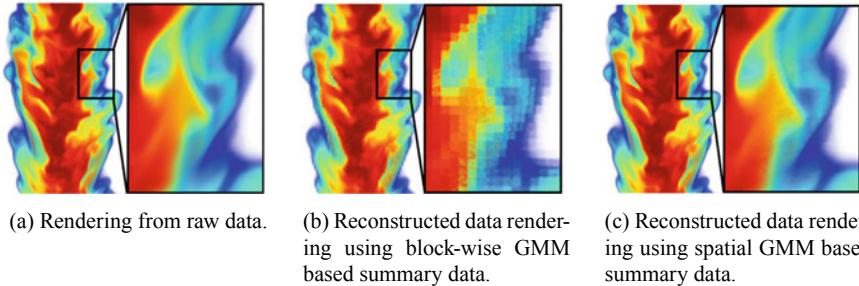


Fig. 11 Visual comparison of volume rendering in the Combustion data set. The samples are drawn from the PDFs, which are calculated at all grid points of the raw data, using Monte Carlo sampling. This image is reprinted from our previous work [37]

Visualization of the reconstructed full resolution data using the spatial distribution-augmented data summaries [37] is presented in Fig. 11. In this example, mixture fraction field of turbulent combustion data is used. The rendering of the ground truth data is depicted in Fig. 11a. For demonstrating the efficacy of the spatial distribution-based data summaries, in Fig. 11b, we have provided the reconstruction result generated from regular block-wise GMM-driven data summaries, which do not use any spatial distribution information. Finally, Fig. 11c shows the result produced from spatial distribution-augmented data summaries, which obtains a smooth reconstruction of the data. It is evident that the augmentation of the spatial distribution information makes the reconstruction more accurate and removes the block boundary irregularities, which are visible in the reconstruction result created from block-wise GMM-guided data summaries (Fig. 11b).

For multivariate data, it is important to reconstruct scalar fields of different variables at the same time. This can be achieved only when the variable relationships are factored in during distribution modeling. Figure 12 shows the multivariate reconstruction results for the Hurricane Isabel data set with 11 physical variables, using multivariate histograms (sparse), multivariate GMM, and copula-based multivariate modeling strategy. Figure 12a–d shows the qualitative reconstruction results of only the Pressure variable. Whereas, Fig. 12e shows the quantitative results of normalized root mean squared error (RMSE) for all the 11 variables for the three different multivariate distribution modeling approaches. As can be seen, the flexible copula-based multivariate distribution modeling approach performs better than standard multivariate distribution model. The storage overhead and estimation times for the 3 different multivariate models are reported in Fig. 12f and g respectively, for both the Isabel and Combustion data sets. The results highlight the fact that multivariate histograms have higher storage footprint, while, multivariate GMMs have large associated estimation time costs as compared to the flexible copula-based method.

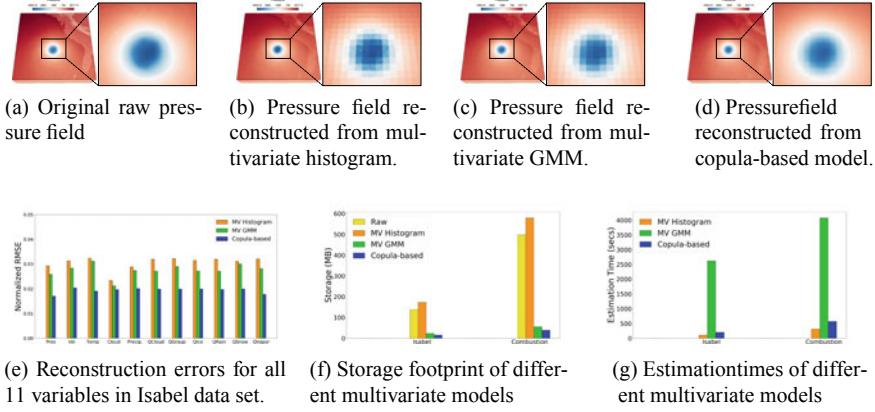


Fig. 12 Qualitative and quantitative results for multivariate sampling-based scalar field reconstruction of Hurricane Isabel data set. This image is reprinted from our previous work [22]

4 Demonstration of an In Situ Distribution-Guided End-to-End Application Study

In this section, we describe an end-to-end real-life example of an application study using in situ generated distribution-based data summaries for solving a practical domain-specific problem. In this application study, we explore rotating stall phenomenon in a transonic jet engine simulation data sets. The data is generated from a large-scale computational fluid dynamics (CFD) simulation code, TURBO [9, 10]. TURBO is a Navier-Stokes based, time-accurate simulation code, which was developed at NASA. A TURBO simulation models the flow of air through a rotor in the engine turbine compressor stage. The model of the rotor of the compressor stage is shown in Fig. 13. The rotor consists of 36 blades and so there are 36 blade passages. A zoomed-in view of the rotor is shown on the right where the tip, the hub, and the leading edge of the blade is highlighted. It has been shown previously that the data generated from TURBO can capture the stall phenomenon in great detail. However, the volume of data generated from TURBO is very large, and therefore, in situ data summarization is critical for enabling timely exploration of the simulation data with high temporal fidelity at an interactive rate.

One of the primary goals of this study was to develop techniques that can detect the rotating stall as early as possible such that the experts can employ stall preventing measures. Rotating stall, if fully developed, can potentially damage the turbine compressor blades. Therefore, early detection of the stall is critical. Furthermore, the reasons behind the inception of rotating stall in transonic engines are still not fully understood and hence is an open research problem. Besides identifying the precursors of the rotating stall, the experts also want to understand the role of different variables during the inception of the stall. In the following discussion, we present two application studies for analyzing and visualizing the rotating stall phenomenon

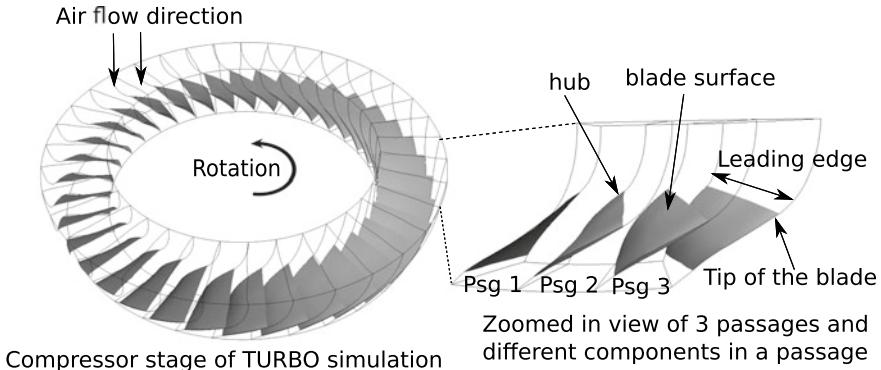


Fig. 13 A diagram of the compressor stage of TURBO simulation and a zoomed in view of it on the right. Different components of a blade is shown. This image is reprinted from our previous work [12]

using both univariate and multivariate in situ distribution-based data summaries and demonstrate their effectiveness.

4.1 Univariate Distribution Anomaly-Guided Stall Analysis

Since the rotating stall is referred to as an instability in the flow data, it can be characterized as an abnormality in the simulation data. In an ideal condition, the simulation is expected to be axisymmetric, and hence, variables such as Pressure and Entropy are expected to have identical values around the compressor stage. Any region where Pressure or Entropy values deviate from its expected behavior can be identified as abnormalities and therefore a region containing potential stall. To capture such regions and interactively analyze rotating stall post hoc, the large-scale simulation data was first summarized in situ using block-wise GMM-based data summaries. To compute the GMM-models efficiently, in this work, instead of using the traditional Expectation-Maximization (EM) algorithm, an approximate incremental mixture model estimation technique was used. More details of this incremental modeling can be found here [12]. The summarization was performed for the Pressure and Entropy variables and their distribution-based summaries were stored into disks. Then, using the reduced GMM-based distribution data, post hoc stall analysis was carried out. Figure 14 presents a schematic of the complete end-to-end analysis workflow. As we can see, the data was summarized in situ and then, in the post hoc analysis phase, the data summaries were used to detect regions that showed spatial and temporal distribution anomalies. Through interactive visualization, the domain experts verified their hypothesis and explored the stall features efficiently.

The GMM-based data summaries were first used to estimate the spatial and temporal region-wise anomalies in the data set. To detect such regions, block-wise

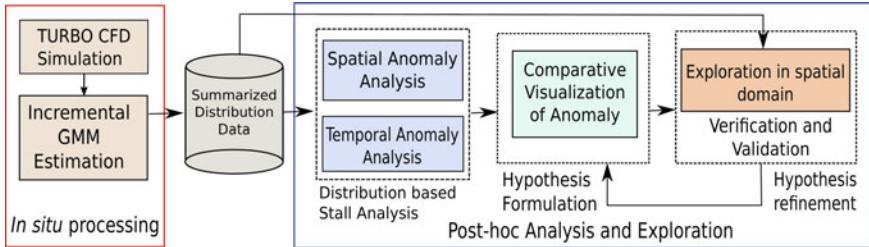


Fig. 14 A schematic of the in situ distribution anomaly-guided stall analysis. This image is reprinted from our previous work [12]

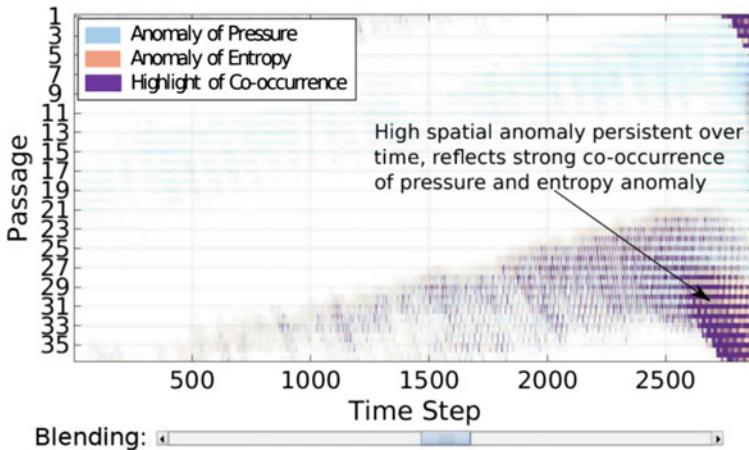


Fig. 15 In situ generated distribution-based spatial anomaly pattern study. The image shows spatial anomaly of Pressure and Entropy variables where the stalled regions are highlighted in blended purple color. This image is reprinted from our previous work [12]

GMM-based distribution-based summaries were compared over space and time for each blade passage. Finally, the detected regions that indicated spatial anomaly were plotted as shown in Fig. 15. This figure shows that abnormalities develop gradually over time and, when the stall happens, such abnormal regions become pronounced (indicated by the dark purple region in the plot). Visualization of such detected spatially anomalous regions in the data domain is shown in Fig. 16. As can be seen, the detected regions are near the tip of the blades as expected and the anomalies are observed for both Pressure (the blue-colored regions) and Entropy variable (the red-colored regions). The compressor blade passages containing such abnormalities are identified as stalled regions. A similar analysis was also done using the temporal anomaly plots. Using both spatial and temporal anomaly-based analysis, the domain expert was able to confirm the effectiveness of distribution-based data summaries in detecting rotating stall. For more information, interested readers are referred to a detailed discussion on this topic [12].

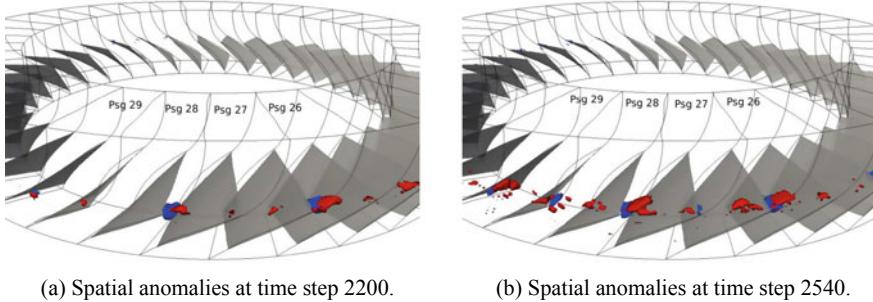


Fig. 16 Visualization of detected spatial anomaly regions of Pressure (in blue surfaces) and Entropy (in red surfaces). The regions are detected near the blade tip regions of several rotor passages. These regions act as blockage to the regular airflow and create flow instability which eventually leads to stall. This image is reprinted from our previous work [12]

4.2 Multivariate Distribution Query-Driven Stall Exploration

Scientists were also interested in understanding the importance of the variables Entropy, U-velocity, and Temperature towards the formation of stall-like features in the turbine passages. This requires that the distribution-based data summaries capture the multivariate relationship among the variables for post hoc analyses. The copula-based multivariate distribution modeling strategy, as discussed in Sect. 2.1.2, was employed to create multivariate data summaries for local partitions. To model the individual variables, a Gaussian distribution was used for partitions with a high normality test score, and a GMM (with 3 modes) was used otherwise. To retain the spatial context of data within a partition, spatial variables (x , y , and z dimensions) were also modeled using uniform distributions. The dependency among all these variables (i.e., 6 total, 3 physical + 3 spatial) was modeled using Gaussian copula functions.

The in situ multivariate data summaries were later used to perform various multivariate post hoc analyses and visualizations. Figure 17a, c, e show the original scalar fields for Entropy, U-velocity, and Temperature respectively. The corresponding sampled scalar fields, reconstructed from the data summaries are shown in Fig. 17b, d, f respectively. Scientists knew that Entropy values greater than 0.8 and negative U-velocities corresponded to potentially unstable flow structures, which can lead to stalls. To focus the study on regions with such multivariate properties, a multivariate query $\text{Entropy} > 0.8$ and $\text{Uvel} < -0.05$ was made to select the region. The corresponding probability field is shown in Fig. 17g, whereas, Fig. 17h shows the isosurfaces of probability value 0.5 across the blade structures. Figure 17i shows the distribution of Temperature values in this queried region (i.e., $P(\text{Temp} | \text{Entropy} > 0.8 \text{ AND } \text{Uvel} < -0.05)$). The peak in the distribution suggests that Temperature values around 0.9 can be related to potential engine stall. Figure 17j and k show how Temperature is correlated with Entropy and U-velocity respectively, in the selected region. There is a strong positive correlation with Entropy and a significant negative

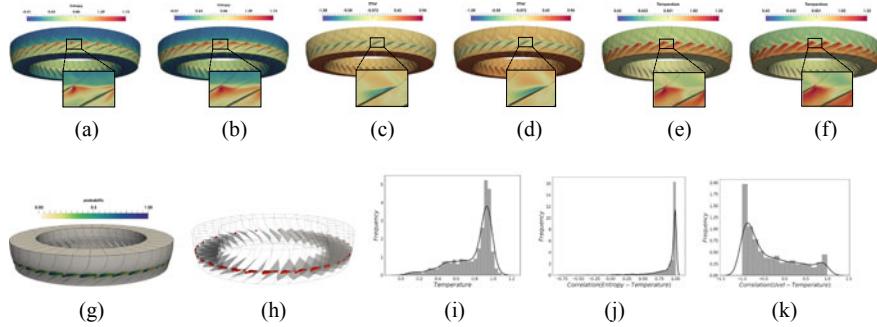


Fig. 17 Post hoc analysis results of the jet turbine data set. **a** Raw Entropy field. **b** Sampled scalar field of entropy. **c** Raw U-velocity field. **d** Sampled scalar field of U-velocity. **e** Raw temperature field. **f** Sampled scalar field of Temperature. **g** Probabilistic multivariate query result for $P(\text{Entropy} > 0.8 \text{ AND } \text{Uvel} < -0.05)$. **h** Isosurface for probability value of 0.5. **i** Distribution of Temperature values in the queried region i.e., $P(\text{Temp} | \text{Entropy} > 0.8 \text{ AND } \text{Uvel} < -0.05)$. **j** Distribution of correlation coefficients between Entropy and Temperature for the queried region. **k** Distribution of correlation coefficients between U-velocity and Temperature for the queried region. This image is reprinted from our previous work [22]

correlation with U-velocity. Such exploratory analysis activity can help scientists to gain more insights into the multivariate relationships in their simulation. For more information, interested readers are referred to a detailed discussion on this topic [22].

From the above analyses, we can observe that the various distribution-based techniques led to a detailed understanding of the rotating stall inception problem and also how different variables can be used to detect stall quickly before it becomes destructive. In the future, the early detection capabilities developed can be used to implement some stall preventing measures. One potential measure is to install sensors at the appropriate places that will be measuring abnormalities using the proposed techniques for early detection of the event in the turbine stage so that when abnormalities are detected, these sensors would recommend the users to act and prevent engine destruction. Also, the knowledge learned from these analyses could lead to a better turbine stage design that will make the engine safer.

4.3 Storage and Performance Evaluation

The performance studies presented here with TURBO simulation were done using a cluster, Oakley [7, 8], at the Ohio Supercomputer Center. Oakley contains 694 nodes with Intel Xeon x5650 CPUs (12 cores per node) and 48 GB of memory per node. A parallel high-performance and shared disk space running Lustre was used for I/O.

4.3.1 Performance Evaluation of Univariate Data Modeling

One single revolution of the complete rotor, i.e. the full annulus model, in a TURBO simulation generates 5.04 TB raw data. To perform the studies presented above, 8 revolutions were run, which generated a total of 40.32 TB data and a total of 28800 time steps. Since the in situ call was made at every 10th time step, it required processing of 4.032 TBs of data. However, for this experimentation, only the rotor was considered and the data for the two stators were not stored. The raw data for the rotor part in PLOT3d format is 690 MB per time step, and hence, storing all the raw data for 8 revolutions at every 10th time step would require 993.6 GB storage. In this study, for summarizing the data using local distribution-based models, the spatial domain was partitioned using non-overlapping regular blocks of size 5^3 . The output of the in situ summarized data for two variables in VTK format took only 51.8 GB for 8 revolutions resulting in approximately 95% data reduction.

Figure 18 presents timing performance for the in situ processing. This figure shows that the in situ summarization time is significantly smaller compared to the simulation time. Furthermore, the raw data I/O shown here can be completely removed if the in situ pathway is taken. Table 1 shows that in situ distribution-based summarization

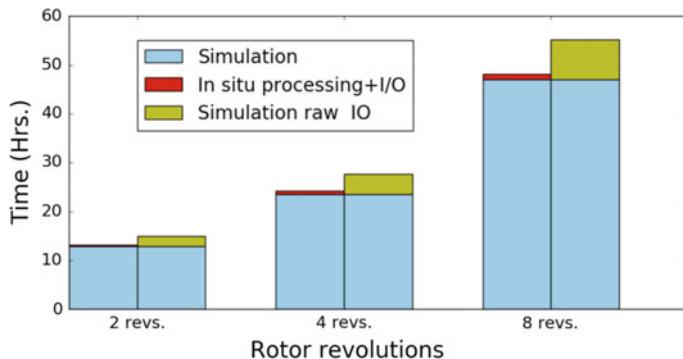


Fig. 18 In situ timing comparison for univariate data modeling using TURBO simulation with and without raw output. Pressure and Entropy variables were summarized using regular block-wise partitioned data and GMMs were used as the distribution model. With the *in situ* pathway, the raw I/O time can be saved. This image is reprinted from our previous work [12]

Table 1 Percentage timing of in situ GMM-based univariate summarization with half and full annulus runs. This table is reused from our previous work [12]

Configuration	2 revs.		4 revs.	
	Simulation	In situ (%)	Simulation (%)	In situ (%)
Half annl. (164 cores)	97.3	2.7	97.5	2.5
Full annl. (328 cores)	97.63	2.37	97.42	2.58

for two variables only takes about 2.5% additional time. While estimating the in situ processing overhead, both the half-annulus model (which consists of 18 passages) and the full-annulus model (the complete rotor with 36 passages) were tested. It is observed that the in situ data summarization time is consistent for both these runs. Hence, by performing in situ processing, we have demonstrated a scalable rotating stall analysis to help the expert achieve a better understanding of the phenomenon. In the following, we present performance results for the multivariate data summarization case.

4.3.2 Performance Evaluation of Multivariate Data Modeling

For the copula-based in situ multivariate data summarization case study, the simulation domain was partitioned regularly into non-overlapping partitions of size 5^3 . Two full revolutions of the turbine data comprised of 7200 time steps. In situ data summaries for local partitions were created every 10th time step, thereby storing 720 time steps. Since the size of the raw data produced at each time step was 690 MB, two full revolutions of the simulation accumulated 496.8 GB of data. Compared to this size, storing multivariate data summaries took only 19.6 GB which resulted in 96% data reduction. Moreover, writing the raw data to the storage disk took around 13% of the simulation execution time, whereas the combined time to estimate the multivariate summaries and write them out to disk took 15.4% of the overall simulation time. Besides reducing the storage footprint, the data summaries offer significantly faster post hoc analysis time. Performing multivariate queries on a regular workstation machine took on average 64.6 s, whereas, reconstruction of the full scalar fields from the data summaries took only 178.3 s on average.

5 Discussion and Guidelines for Practitioners

5.1 Discussion

The above sections demonstrate the efficacy and usefulness of various in situ distribution-based data summarization techniques for performing flexible and exploratory data analysis and visualization. It has been shown that when the data is reduced in the form of distributions then features in such summary data can be searched efficiently by defining the feature as a distribution. This is primarily beneficial for scientific features which are hard to define precisely due to the complexity of the feature [14]. In such cases, a statistical distribution-based feature descriptor is found to be effective. Another advantage is that the distribution-based data representations can be directly used in these cases and a full reconstruction is not necessary for finding or tracking features. This also helps in accelerating the post hoc analysis. However, a reconstruction of the full resolution data is also possible

from the distribution-based data summaries which can be used to explore scientific features using traditional techniques when precise feature descriptors are available.

Another observation for the distribution-based data analysis techniques is that typically the distribution estimation is done *in situ* and feature analysis and visualization are conducted in the post hoc analysis phase. This strategy is adopted by keeping in mind that a majority of the visual analysis tasks require interaction with the data, forming new hypotheses, and then refining and verifying those hypotheses as results are studied. This process engages application scientists in the exploration loop and often can take a considerable amount of time. Therefore, these kinds of exploratory analyses are not suitable for an *in situ* environment when the simulation is running because doing so could slow down the simulation significantly which is undesirable. However, we acknowledge that if the application scientists know precisely about the data features that they are interested in, then extraction and visualization of such features directly in the *in situ* environment might be a viable solution. In such cases, visualization artifacts such as images of the features can be stored for post hoc analysis.

Further, besides applying traditional statistical methods for estimating distributions as discussed above, there is a recent surge in the use of deep learning-based models to estimate data distribution in the field of machine learning. Two such prospective methods are the Generative Adversarial Networks (GANs) [18] and Variational Autoencoders (VAEs) [26]. Such deep learning-based models adopt unique optimization strategies to model very high-dimensional distributions of a wide range of objects. They convert a purely statistical problem of distribution estimation into an optimization problem (i.e., find the parameter values that minimize some objective function). However, to model the distribution perfectly, deep learning methods need multiple iterations over the data, which can be infeasible *in situ* solution. Recent efforts into the application of such methods in the field of scientific visualization [4, 19, 25] have been to mostly perform post hoc analyses. Bringing in the advantages of such powerful models to an *in situ* environment is an exciting research prospect in the near future.

5.2 Guidelines for Practitioners

In this section, we briefly provide some guidelines for users and practitioners about how appropriate distribution models can be selected and how some of these techniques can be implemented in a simulation. Given a particular task, the first choice is to decide whether univariate distribution models are sufficient or multivariate models will be needed. If multivariate models are necessary, then we recommend using the statistical Copula-based approach. This technique is suitable when several variables are needed to be summarized and can tackle the curse of dimensionality problem that often arises while estimating high-dimensional distributions. If univariate distribution-based models are sufficient, then we found that GMM-based summarization performed best. In Sect. 1.3, we have provided several

statistical tests that can be used to select the appropriate number of Gaussian models when estimating a GMM. Note that using the traditional Expectation-Maximization algorithm for estimating GMM parameters can be costly at times, and hence, if performance is critical, an alternate incremental Gaussian mixture model estimation strategy is suggested in [12]. The use of an incremental algorithm will trade some estimation accuracy but will result in faster parameter estimation.

A majority of the above techniques advocate for the local region-based distribution models. In such modeling, since the data blocks are non-overlapping, the distribution estimation for each data block is independent and hence no additional data communication is required. So, it is straightforward to compute such models. First, the data in each processing node needs a partitioning and then an appropriate distribution model can be used. Users and practitioners are encouraged to consider using EDDA [1], the open-source distribution-based analysis library, which came out of the research done at the Ohio State University and implements building blocks of several of the distribution estimation techniques that have been discussed in this chapter. The library is under development and so some of the advanced techniques might not be readily available. However, we believe this library will be a useful starting point for the practitioners who are interested in using distributions in their analyses.

If the users are interested in conducting the feature analysis in the in situ environment, then additional data communication among computing nodes will be needed. Since a data feature could span across multiple computing nodes, a strategy needs to be developed which will send data distributions to the neighboring processing nodes so that the complete feature can be extracted and analyzed. Sending data distributions to the neighboring blocks is expected to be a cheap operation since the size of distribution parameters is significantly smaller compared to the raw data. Note that this will require new research to come up with a desired and scalable solution. However, we believe that with the present advances made in the distribution-based analysis domain, as discussed throughout this chapter, the strategy of estimating distributions in situ and performing feature analysis post hoc have resulted in promising results and a variety of complex and important visual-analysis tasks were satisfactorily performed.

6 Additional Research Possibilities and Future Scopes

The preceding sections present various in situ distribution-based data modeling techniques for both univariate and multivariate scalar data sets. The applicability of such distribution-based data summaries for solving various domain specific problems is also demonstrated in Sect. 4. In order to study the usefulness of these distribution-based data summaries in the context of a broader set of visualization tasks, we plan to conduct a comprehensive evaluation where comparison among various data reduction techniques such as distribution-based summaries, data compression techniques, and sampling-based reduction approaches will be considered. Besides analyzing scalar data sets, distribution-based data summaries can also be used

for analyzing and visualizing vector field data sets, ensemble data sets, and also particle-based data. Several studies have already been done for summarizing vector fields using distribution-based representations [23, 27]. To generate streamlines from such distribution-based vector data summaries, He et al. adopted a Bayesian framework using particle filtering technique [23]. In another work, to trace the particles accurately using distribution-based vector fields, Li et al. used winding angle of particles trajectories for correctly predicting the particle path using a Bayesian approach [27]. A recent work demonstrated usefulness of distribution-based techniques for in situ particle data reduction [28].

Among other future possibilities, applications of distribution-based data summaries have also been tested for summarizing and analyzing large ensemble data sets. In one approach, Wang et al. [38] first captured the relationship between high and low resolution ensemble data members. Then, for future runs of the simulation using different parameter combinations, the data was summarized in situ using GMM-based data models. During post hoc analysis, the high-resolution data was reconstructed from the GMM-based down-sampled data summaries using the prior knowledge to improve the reconstruction quality. The in situ study was conducted using Nyx cosmology simulation [3]. For more details about this technique, please refer to [38]. Besides statistical super resolution, distribution-based representations of ensemble data can also be used for studying data features which are characterized as a range of data values. Study of such features were done by He et al. [24] using range likelihood trees.

7 Conclusion

In this chapter we have described various methods of in situ distribution-based data summarization techniques, which on one hand can achieve significant data reduction, and on the other hand can also be used as a flexible data product for post hoc visual analysis. We discussed in details the advantages and disadvantages of using different parameter and non-parametric distribution models for data summarization from the perspective of their in situ feasibility. Using a real world large-scale CFD simulation, we discussed the challenges and possible solutions for distribution-based data modeling for both univariate and multivariate cases. Additionally, several important post hoc data analysis and visualization tasks have been briefly discussed which highlight the effectiveness of the in situ generated distribution-based data summaries in solving a wide range of visualization and data analysis problems.

Acknowledgements We sincerely acknowledge the contributions from Ko-Chih Wang, Wenbin He, Cheng Li, Chun-Ming Chen, Kewei Lu, and Tzu-Hsuan Wei. This research was supported in part by the US Department of Energy Los Alamos National Laboratory contract 47145, UT-Battelle LLC contract 4000159447, NSF grants IIS-1250752, IIS-1065025, and US Department of Energy grants DE-SC0007444, DE-DC0012495. We would also like to thank Prof. Jen-Ping Chen from the Department of Mechanical and Aerospace Engineering, Ohio State University for providing access to the TURBO simulation and offering invaluable domain feedback for the in situ

application studies. The in situ experiments used computing resources at the Ohio Supercomputer Center [7]. The Hurricane Isabel data set has kindly been provided by Wei Wang, Cindy Bruyere, Bill Kuo, and others at NCAR. Tim Scheitlin at NCAR converted the data into the Brick-of-Float format described above. The Turbulent Combustion data set is made available by Dr. Jacqueline Chen at Sandia National Laboratories through the US Department of Energy's SciDAC Institute for Ultrascale Visualization. This research was released under LA-UR-20-20838.

References

1. Edda—extreme-scale distribution-based data analysis library. <https://sites.google.com/site/gravityvisdb/edda>
2. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süstrunk, S.: Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.* **34**(11), 2274–2282 (2012). <https://doi.org/10.1109/TPAMI.2012.120>
3. Almgren, A.S., Bell, J.B., Lijewski, M.J., Lukic, Z., Andel, E.V.: Nyx: A massively parallel amr code for computational cosmology. *Astrophys. J.* **765**(1), 39 (2013)
4. Berger, M., Li, J., Levine, J.A.: A generative model for volume rendering. *IEEE Trans. Vis. Comput. Graph.* **1** (2018). <https://doi.org/10.1109/TVCG.2018.2816059>
5. Bilmes, J.: A gentle tutorial on the EM algorithm including gaussian mixtures and baum-welch. Technical report, International Computer Science Institute (1997)
6. Biswas, A., Dutta, S., Shen, H., Woodring, J.: An information-aware framework for exploring multivariate data sets. *IEEE Trans. Vis. Comput. Graph.* **19**(12), 2683–2692 (2013). <https://doi.org/10.1109/TVCG.2013.133>
7. Center, O.S.: Ohio supercomputer center (1987). <http://osc.edu/ark:/19495/f5s1ph73>
8. Center, O.S.: Oakley supercomputer. <http://osc.edu/ark:/19495/hpc0cvqn> (2012)
9. Chen, J.P., Hathaway, M.D., Herrick, G.P.: Prestall behavior of a transonic axial compressor stage via time-accurate numerical simulation. *J. Turbomach.* **130**(4), 041014 (2008)
10. Chen, J.P., Webster, R., Hathaway, M., Herrick, G., Skoch, G.: Numerical simulation of stall and stall control in axial and radial compressors. In: 44th AIAA Aerospace Sciences Meeting and Exhibit. American Institute of Aeronautics and Astronautics (2006). <https://doi.org/10.2514/6.2006-418>, <http://arc.aiaa.org/doi/abs/10.2514/6.2006-418>
11. Dutta, S.: In situ summarization and visual exploration of large-scale simulation data sets. Ph.D. thesis, The Ohio State University (2018). http://rave.ohiolink.edu/etdc/view?acc_num=osu1524070976058567
12. Dutta, S., Chen, C., Heinlein, G., Shen, H., Chen, J.: In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE Trans. Vis. Comput. Graph.* **23**(1), 811–820 (2017). <https://doi.org/10.1109/TVCG.2016.2598604>
13. Dutta, S., Liu, X., Biswas, A., Shen, H.W., Chen, J.P.: Pointwise information guided visual analysis of time-varying multi-fields. In: SIGGRAPH Asia 2017 Symposium on Visualization, SA '17. Association for Computing Machinery, New York, NY, USA (2017). <https://doi.org/10.1145/3139295.3139298>
14. Dutta, S., Shen, H.: Distribution driven extraction and tracking of features for time-varying data analysis. *IEEE Trans. Vis. Comput. Graph.* **22**(1), 837–846 (2016). <https://doi.org/10.1109/TVCG.2015.2467436>
15. Dutta, S., Woodring, J., Shen, H., Chen, J., Ahrens, J.: Homogeneity guided probabilistic data summaries for analysis and visualization of large-scale data sets. In: 2017 IEEE Pacific Visualization Symposium (PacificVis), pp. 111–120 (2017). <https://doi.org/10.1109/PACIFICVIS.2017.8031585>
16. Findley, D.F.: Counterexamples to parsimony and bic. *Ann. Inst. Stat. Math.* **43**(3), 505–514 (1991)

17. Gentle, J.E.: Random Number Generation and Monte Carlo Methods. Springer, New York (2007). <https://doi.org/10.1007/b97336>
18. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc. (2014). <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
19. Han, J., Tao, J., Wang, C.: Flownet: A deep learning framework for clustering and selection of streamlines and stream surfaces. IEEE Trans. Vis. Comput. Graph. 1 (2018). <https://doi.org/10.1109/TVCG.2018.2880207>
20. Hazarika, S., Biswas, A., Dutta, S., Shen, H.W.: Information guided exploration of scalar values and isocontours in ensemble datasets. Entropy **20**(7) (2018)
21. Hazarika, S., Biswas, A., Shen, H.W.: Uncertainty visualization using copula-based analysis in mixed distribution models. IEEE Trans. Vis. Comput. Graph. **24**(1), 934–943 (2018). <https://doi.org/10.1109/TVCG.2017.2744099>
22. Hazarika, S., Dutta, S., Shen, H., Chen, J.: Codda: a flexible copula-based distribution driven analysis framework for large-scale multivariate data. IEEE Trans. Vis. Comput. Graph. **25**(1), 1214–1224 (2019). <https://doi.org/10.1109/TVCG.2018.2864801>
23. He, W., Chen, C., Liu, X., Shen, H.: A bayesian approach for probabilistic streamline computation in uncertain flows. In: 2016 IEEE Pacific Visualization Symposium (PacificVis), pp. 214–218 (2016). <https://doi.org/10.1109/PACIFICVIS.2016.7465273>
24. He, W., Liu, X., Shen, H., Collis, S.M., Helmus, J.J.: Range likelihood tree: a compact and effective representation for visual exploration of uncertain data sets. In: 2017 IEEE Pacific Visualization Symposium (PacificVis), pp. 151–160 (2017). <https://doi.org/10.1109/PACIFICVIS.2017.8031589>
25. He, W., Wang, J., Guo, H., Wang, K., Shen, H., Raj, M., Nashed, Y.S.G., Peterka, T.: Insitunet: deep image synthesis for parameter space exploration of ensemble simulations. IEEE Trans. Vis. Comput. Graph. **26**(1), 23–33 (2020)
26. Kingma, D.P., Welling, M.: Auto-encoding variational bayes (2013). <http://arxiv.org/abs/1312.6114>. Cite arxiv:1312.6114
27. Li, C., Shen, H.W.: Winding angle assisted particle tracing in distribution-based vector field. In: SIGGRAPH Asia 2017 Symposium on Visualization. Association for Computing Machinery, New York, NY, USA (2017). <https://doi.org/10.1145/3139295.3139297>
28. Li, G., Xu, J., Zhang, T., Shan, G., Shen, H., Wang, K., Liao, S., Lu, Z.: Distribution-based particle data reduction for in-situ analysis and visualization of large-scale n-body cosmological simulations. In: 2020 IEEE Pacific Visualization Symposium (PacificVis), pp. 171–180 (2020)
29. Lu, K., Shen, H.W.: A compact multivariate histogram representation for query-driven visualization. In: Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV), LDAV ’15, pp. 49–56 (2015)
30. Rank, J.: Copulas: From Theory to Application in Finance. Bloomberg Financial. Wiley (2007). <https://books.google.com/books?id=133HkvhOHC8C>
31. Razali, N.M., Wah, Y.B.: Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. J. Stat. Model. Anal. **2**(1), 21–33 (2011)
32. Schindler, K., Wang, H.: Smooth foreground-background segmentation for video processing. In: Proceedings of the 7th Asian Conference on Computer Vision—Volume Part II, ACCV’06, pp. 581–590. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11612704_58
33. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**, 591–611 (1965)
34. Sklar, M.: Fonctions de Répartition À N Dimensions Et Leurs Marges. Université Paris 8 (1959)
35. Stauffer, C., Grimson, W.: Adaptive background mixture models for real-time tracking. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, p. 252 (1999). <https://doi.org/10.1109/CVPR.1999.784637>
36. Stephens, M.A.: EDF statistics for goodness of fit and some comparisons. J. Am. Stat. Assoc. **69**(347), 730–737 (1974). <https://doi.org/10.1080/01621459.1974.10480196>

37. Wang, K., Kewei Lu, Wei, T., Shareef, N., Shen, H.: Statistical visualization and analysis of large data using a value-based spatial distribution. In: 2017 IEEE Pacific Visualization Symposium (PacificVis), pp. 161–170 (2017). <https://doi.org/10.1109/PACIFICVIS.2017.8031590>
38. Wang, K., Xu, J., Woodring, J., Shen, H.: Statistical super resolution for data analysis and visualization of large scale cosmological simulations. In: 2019 IEEE Pacific Visualization Symposium (PacificVis), pp. 303–312 (2019). <https://doi.org/10.1109/PacificVis.2019.00043>

Exploratory Time-Dependent Flow Visualization via In Situ Extracted Lagrangian Representations



Sudhanshu Sane and Hank Childs

Abstract This chapter considers exploratory flow visualization of time-dependent vector fields via in situ extraction of Lagrangian representations. The Lagrangian perspective is more capable than the traditional approach of incorporating the increased spatiotemporal data afforded by in situ processing, creating significantly better trade-offs with respect to accuracy and storage. For example, in situ Lagrangian-based flow analysis has delivered the same accuracy as the traditional approach with less than 2% of the storage, or 10X greater accuracy with the same storage. The chapter begins by discussing the Lagrangian frame of reference and how this frame of reference can be used as an in situ operator for data reduction. Next, opportunities for achieving maximum information per byte—where particles are placed, how they are terminated, and how much information to store per particle trajectory are discussed. The chapter then considers post hoc exploration using the Lagrangian representation, as well as the corresponding challenges involved. Finally, the chapter concludes with a qualitative evaluation to demonstrate the efficacy of the technique and a discussion of the current state of the art.

1 Introduction

A flow field describes how a fluid’s velocity varies over time in a two or three dimensional domain. Flow fields are produced by computational fluid dynamics (CFD) simulations and often require scientific visualization to understand, verify, and explore phenomena of interest. Typically, the velocity of a flow field is represented within a simulation code as a time-dependent vector field defined over a discretized mesh. To visualize the vector field, i.e., to perform *flow visualization*, techniques can choose to operate on the vector field data as “steady state” (i.e., ignoring that the vector

S. Sane (✉) · H. Childs
University of Oregon, Eugene, USA
e-mail: ssane@uoregon.edu

H. Childs
e-mail: hank@uoregon.edu

field evolves over time, typically in order to reduce execution time by considering less data) or as “unsteady state” (i.e., recognizing the time-dependent nature of the field, typically at a cost of higher execution time). That said, for unsteady state flow visualization to be accurate, it requires complete spatiotemporal data. Unfortunately, this data rarely exists for post hoc analysis in practice since simulation codes need to perform temporal subsampling to fit their results on disk. Worse, trends in high-performance computing are leading temporal subsampling to become increasingly aggressive, leading unsteady state flow visualization to become increasingly inaccurate.

In situ processing represents a new opportunity to obtain sufficient temporal resolution to perform accurate flow visualizations. Further, when the desired visualizations are known *a priori*, these accurate visualizations can be achieved in a relatively straightforward manner. However, when the desired visualizations are not known *a priori*, the traditional approach for generating flow visualizations cannot take advantage of the increased temporal information—while in situ processing enables access to more data, the traditional approach has no way to take advantage of this data. This use case—accurate, exploratory flow visualization for unsteady state flow—is the motivation for this chapter. The main idea for enabling exploration is one already discussed in the introductory chapter and in other chapters in this book: using in situ processing to generate a reduced form and then doing post hoc exploration on that reduced form. That said, a challenge and opportunity for time-dependent vector field data is fully taking advantage of the increased temporal information provided by in situ processing.

This chapter explores the use of the Lagrangian perspective when using in situ processing to calculate a reduced representation of time-dependent vector fields. This reduced representation can then be used as part of a workflow, illustrated in Fig. 1, to enable exploratory flow visualization. This overall workflow is referred to as in situ *Lagrangian-based flow analysis*. Seminal research on this approach demonstrated improved accuracy and data storage propositions compared to traditional methods for exploratory flow visualization [1]. These improvements are possible because

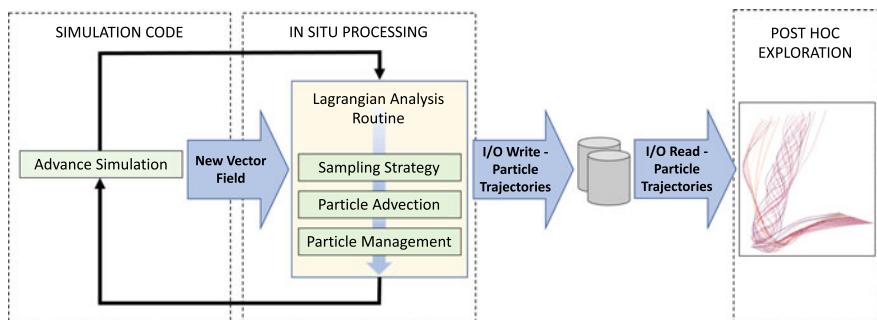


Fig. 1 Diagram showing the operations for in situ Lagrangian-based flow analysis, as well as its relation to the simulation code and post hoc visualization workflow

Lagrangian-based flow analysis is able to take advantage of the increased temporal resolution from in situ processing.

2 Background and Motivation

This background section contains two parts. First, Sect. 2.1 discusses two frames of reference for representing time-dependent vector data: Eulerian and Lagrangian. Next, Sect. 2.2 discusses the limitations of the traditional, Eulerian-based paradigm for visualizing and analyzing time-dependent vector data.

2.1 *Frames of Reference in Fluid Dynamics*

In fluid dynamics, the Eulerian and Lagrangian frames of reference are two ways of looking at fluid motion. In the Eulerian frame of reference, the observer is at a fixed position. In the Lagrangian frame of reference, however, the observer is attached to a fluid parcel and moves through space and time. In computational fluid dynamics, simulations can be designed to employ a fixed mesh (Eulerian), have simulation grid points follow the simulation velocity field (Lagrangian), or use hybrid Eulerian-Lagrangian specifications.

When a velocity field is stored in an Eulerian representation, it is typically done by means of its vector field. A velocity field v is a time-dependent vector field that maps each point $x \in \mathbb{R}^d$ in space to the velocity of the field for a given time $t \in \mathbb{R}$

$$v : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d, \quad x, t \mapsto v(x, t) \quad (1)$$

In a practical setting, the velocity field is typically defined over a fixed, discrete mesh and represents the state of the velocity field at a specific instant of time, i.e., at a specific simulation time and cycle.

In a Lagrangian representation, flow behavior is encoded via its flow map $F_{t_0}^t$. The flow map considers what happens when a massless particle at some starting position, x_0 , and starting time, t_0 , is advected by the velocity field. The mathematical definition of the flow map is the mapping

$$F_{t_0}^t(x_0) : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad t \times t_0 \times x_0 \mapsto F_{t_0}^t(x_0) = x(t) \quad (2)$$

of initial values x_0 to the solutions of the ordinary differential equation

$$\frac{d}{dt}x(t) = v(x(t), t) \quad (3)$$

In a practical setting, the flow map is stored as sets of particle trajectories calculated in the time interval $[t_0, t] \subset \mathbb{R}$. The stored information, encoded in the form of known particle trajectories, represents the behavior of the velocity field over an interval of time.

Finally, the two frames of reference are theoretically equivalent [4]. For the Eulerian representation, particle trajectories can be calculated from the velocity field through integration. For the Lagrangian representation, the velocities at arbitrary locations can be calculated from the particle trajectories through differentiation. That said, their application in practical contexts varies.

2.2 *Traditional Paradigm for Visualization and Analysis of Time-Dependent Vector Fields*

There exist a rich set of flow visualization techniques, including line integral convolution, finite-time Lyapunov exponents (FTLE), pathlines, etc. Each of these techniques uses particle trajectories as building blocks for their respective visualizations.

The traditional approach for calculating particle trajectories is called “particle advection” and uses the Eulerian frame of reference. The input to this operation is Eulerian velocity data, i.e., a vector field defined on a discretized mesh. The trajectory of a given particle is calculated iteratively, advancing the particle by small displacements each step. Each of the steps is referred to as an “advection step” and is calculated by solving an ordinary differential equation (ODE) using the velocity field. There are a variety of numerical methods to solve ODEs; with scientific visualization the most commonly used method is a fourth-order Runge-Kutta method (RK4).

Accuracy is a critical consideration for particle advection, especially for unsteady-state flow. Numerical solvers (RK4 or otherwise) require velocity field evaluations at various locations, typically the current location of the particle as well as nearby locations. These evaluations require interpolation, which introduces error. For unsteady-state flow visualization, the desired evaluations are at arbitrary times, requiring temporal interpolation. Therefore, the more temporal resolution available, the less error.

Unfortunately, trends in supercomputing are reducing temporal resolution, and thus increasing error. As discussed in the introductory chapter, the growing gap between I/O and computing capabilities of modern supercomputers compels the use of temporal subsampling to store data. As a consequence, only a subset of data is practically available to perform particle advection in a post hoc Eulerian settings. For flow visualization, this means interpolation error will be high and results can be misleading.

From an implementation perspective, the traditional Eulerian approach of storing velocity field data on a fixed mesh yields two significant benefits: fast cell location and easy interpolation. This makes the computational cost of the particle advection operation during post hoc flow visualization relatively inexpensive. That said, I/O operations—writing large files to disk representing simulation output and subse-

quently reading those files from disk for post hoc exploratory flow visualization—will remain bottlenecks in their respective workflows.

Overall, the challenges in accuracy and performance jeopardize the ability to perform time-dependent flow visualization under the traditional paradigm, and motivates new approaches.

3 Lagrangian-Based Flow Analysis

This section considers how Lagrangian-based flow analysis operates as a workflow. It is organized into two parts: Sect. 3.1 considers the two distinct phases of computation involved in Lagrangian-based flow analysis, while Sect. 3.2 considers the differences with the traditional, Eulerian workflow.

3.1 Phases of Computation

There are two phases of computation involved in Lagrangian-based flow analysis: *in situ* extraction and post hoc exploration (described in detail in Sects. 4 and 5, respectively). The *in situ* phase involves calculating a Lagrangian representation of the vector field, i.e., tracing basis trajectories (pathlines that can be used subsequently to infer additional pathlines). *In situ* access to the complete spatial and temporal resolution of the simulation velocity field enables the extraction routine to accurately calculate the trajectories that form the stored flow map.

With respect to a Lagrangian representation extracted *in situ*, the following characteristics are desirable:

- It should be possible to compute within *in situ* constraints.
- It should maximize information per byte stored to disk.
- It should support accurate and interactive post hoc exploration.

The major research challenge with Lagrangian-based *in situ* reduction is to achieve all three of these characteristics. Further, the basic framework of the Lagrangian paradigm inherently has no constraints on how particle trajectories should be seeded, terminated, selected, represented, or stored. In particular, selection of particle trajectories requires smart sampling along both spatial and temporal axes. These important areas of flexibility are discussed in Sects. 4.2 and 4.3.

The second phase, i.e., the post hoc exploration phase, involves performing flow visualizations using the stored Lagrangian basis trajectories. In this phase, there are no constraints on the types of flow visualizations performed; the flow visualizations specify seed locations, and new pathlines can be interpolated using the *in situ* extracted data. That said, this interpolation step is non-trivial and also the subject of research. Section 5 discusses the factors involved in the post hoc interpolation phase in more detail.

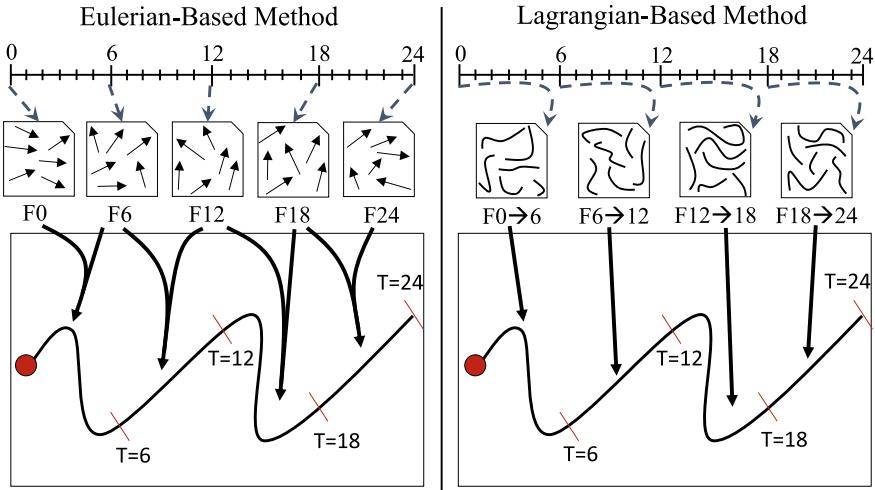


Fig. 2 Notional example showing the differences between an Eulerian-based and a Lagrangian-based method. In this example, a simulation code runs for 24 cycles total and outputs data every 6 cycles. The Eulerian method stores time slices, denoted by F_X , where X denotes the cycle. The Lagrangian method calculates particle trajectories in situ and stores data capturing the behavior of the vector field over an interval of time. These files are denoted by $F_X \rightarrow Y$, where X and Y represent the cycles that begin and end the interval of calculation. The figure illustrates how each technique interpolates the stored data to calculate a new particle trajectory. This image is adapted from a version by Agranovsky et al. [1]

3.2 Differences Between Eulerian and Lagrangian-Based Flow Analysis

The Lagrangian paradigm, unlike the Eulerian paradigm, introduces an encumbrance on the simulation code when it calculates its basis trajectories. The impact of this overhead is an important consideration when assessing the practicality of the method.

To better highlight the differences between the methods, consider the notional example in Fig. 2. There are differences in the type of data stored, the information represented by the data, and how data is interpolated post hoc to calculate a new particle trajectory. In particular, the Eulerian-based method stores a vector field representing a time slice, while the Lagrangian-based method stores particle trajectories representing time intervals. To calculate particle trajectories, Eulerian-based advection solves ordinary differential equations and performs spatial and temporal interpolation. In this case, two time slices (files) are interpolated to advance a particle across an interval of time. For Lagrangian-based advection, a single file is used to advance a particle across an interval of time. Although the notional example shows identical interpolated particle trajectories for both techniques, the actual trajectories are likely to be different. Further, in practice, the Lagrangian trajectory is typically more accurate since the stored Lagrangian representation can encode more information.

Table 1 Differences between the Lagrangian and Eulerian paradigm from a simulation’s perspective. This table is adapted from a version by Agranovsky et al. [1]

	Eulerian	Lagrangian
Saved files contain	Vector fields	Particle trajectories
Saved files represent	Time slices	Time intervals
Reducing I/O, storage	Less time slices	Less particles
Increasing accuracy	More time slices	More particles
Simulation overhead	I/O	I/O + Lagrangian analysis routine computation and memory

tion per byte of storage. Section 6 contains visualizations highlighting the differences in the interpolated trajectories in practical settings.

Table 1 summarizes some differences between the two methods. Although the Lagrangian-based method has an increased encumbrance from running in situ, this cost is offset by multiple benefits: reduced storage requirements, increased accuracy, and improved post hoc performance [1]. For example, the Lagrangian-based method has been demonstrated to be up to 10X more accurate than the Eulerian-based method when considering the same storage, and can provide comparable accuracy for data reductions of up to 64X under sparse temporal settings. Further, depending on the data reduction, the reduced burden on I/O write and read operations can result in performance improvements.

4 In Situ Extraction

This section discusses the in situ extraction phase of Lagrangian-based flow analysis in more detail. This phase involves computing Lagrangian basis trajectories that are built by using the Eulerian approach—advection particles using numerical techniques and the (Eulerian) vector field from the simulation code. Typically, the particle is displaced for an amount of time corresponding to the simulation’s advancement, i.e., if the simulation advances from cycle C to $C + 1$ and goes from time T to $T + \epsilon$, then the particles will advance for ϵ time. There are several factors that influence the outcome of this in situ extraction phase. Specifically, Sect. 4.1 begins with a discussion of the in situ costs and constraints. Next, Sects. 4.2 and 4.3 cover strategies for spatial and temporal sampling of a time-dependent vector field. Finally, Sect. 4.4 discusses options for storage of a Lagrangian representation and the impact it can have on the overall workflow.

4.1 In Situ Costs and Constraints

When considering an in situ routine there are two main costs that must be accounted for: execution time and memory. In situ analysis routines are often allocated a limited resource budget and then required to operate within them. Fortunately, there are several parameters or “knobs” that exist within the Lagrangian framework that can impact (and potentially reduce) these costs. The five main components that contribute to in situ costs are:

- The particle advection workload, i.e., the number of particle trajectories being calculated.
- The frequency and amount of I/O required by the in situ routine.
- Communication costs to exchange information in a distributed memory setting.
- Computation costs of a sampling strategy.
- Memory costs for storing particles and their trajectory information.

The limited memory available places restrictions on (1) the number of particle trajectories that can be calculated, (2) the number of locations along each particle trajectory that can be stored, and lastly, (3) the number of time slices that can be retained in memory in situ. Remaining within in situ constraints requires selecting an appropriate particle advection workload and in-memory particle trajectory representation. In cases where memory usage increases over an interval of computation (e.g., storing many intermediate locations along a particle trajectory), the frequency at which data is moved from memory to disk is pertinent.

Extracting a Lagrangian representation involves computing integral curves in a distributed memory environment. Although several research works have considered preprocessing of the vector field and redistribution of the data, these options are unattractive within an in situ context since the simulation code has already partitioned the data. Thus, using the simulation’s partitioning incurs no additional costs, while repartitioning will increase memory usage and take time. Depending on the distribution of the data and the underlying vector field, the number of particles crossing node boundaries to continue advection could result in an increase of overall execution time. With respect to a sampling strategy (spatial and temporal), the execution time required will vary depending on the sampling algorithm and corresponding implementation. These processes should, ideally, utilize the available hardware acceleration on modern supercomputer compute nodes to produce low-cost, fast techniques that minimize the overall encumbrance on the simulation code.

Overall, in order to not exceed usage of allocated resources, these costs and constraints must be accounted for when designing an in situ Lagrangian extraction routine.

4.2 Spatial Sampling: Seed Placement

Spatially sampling the vector field, i.e., selecting locations to seed basis particles, plays a critical role in determining the quality of the data extracted. In general, a spatial sampling strategy is responsible for directing the following three operations:

- An initial placement of seed particles.
- When a new seed particle should be introduced.
- When an existing seed particle should be terminated.

Seed placement or spatial sampling strategies can be guided by the desire to achieve any of the following objectives in varying orders of priority:

- Maximize information content per byte stored to disk.
- Coverage of the domain, i.e., every region of the domain receives coverage.
- Focus on capturing regions of interest or specific features accurately.
- Minimizing in situ encumbrance, i.e., fast execution times and/or low memory usage.

To work as an effective in situ data reduction operator, however, these objectives are sometimes in tension and require navigating a trade-off. Although maximizing the information per byte stored to disk might seem of highest priority, it is easy to imagine a scenario where this characteristic is compromised for better domain coverage or algorithmic simplicity.

Spatial sampling strategies must consider the distribution of seed particles in both space and time. Given the nature of particle trajectories in unsteady state flow, particles can cluster in regions while leaving voids in other regions. To address this problem, seed particles need to either be reset periodically or added/removed as required to maintain coverage. In the study by Agranovsky et al. [1], seeds are placed at uniform locations in the domain and the particle trajectories are terminated at regular intervals. Although this approach can provide good domain coverage using short trajectories, a uniform distribution of particles is not always the best allocation of resources. In a work by Sane et al. [12], seed particles follow the velocity field and form long trajectories (since post hoc interpolation of long trajectories has been shown to be more accurate [4, 9]), storing locations uniformly along the trajectory. Further, the approach uses a Delaunay triangulation over the seeds to (1) identify locations to add seeds and fill voids and (2) remove seeds in regions of seed clustering. Although this approach provides improved accuracy-storage propositions, the in situ cost of the sampling strategy is high.

There is much research to be done in this space. For example, a strategy might benefit from seed placement guided by information derived from the velocity field (e.g., entropy, curvature, divergence, etc.). Further, derivation of informative scalar fields from a vector field can often be performed in parallel by leveraging hardware acceleration. Figure 3 illustrates a uniform and entropy-guided seed placement for the Double Gyre vector field (a commonly used analytic data set). Multiple research studies in the area of seed placement [13–15] demonstrate the efficacy of field-guided approaches for streamline selection to visualize the velocity field, and these

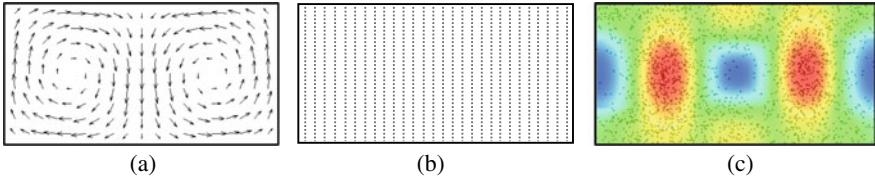


Fig. 3 An illustration of two possibilities for seed placement given the Double Gyre vector field. **a** shows a vector glyph visualization. **b** presents a simple uniform seed placement and **c** colormaps an entropy field derived from the vector field and shows seed placement with a density proportional to the value of entropy

findings may very well translate to in situ Lagrangian-based flow analysis routines. To summarize, field-guided spatial sampling strategies have significant potential to improve information per byte stored in a Lagrangian representation while remaining computationally efficient.

4.3 Temporal Sampling: Curve Approximation

Temporal sampling refers to how much of a single basis trajectory should be stored. A basis trajectory contains the entire route a particle traveled. In practical terms, this route comprises the positions resulting from each advection step. The temporal sampling question, then, is which of these positions along the trajectory should be saved? Saving all of the positions along a trajectory will best capture the underlying vector field, but incurs a large storage cost. Saving fewer positions reduces this storage cost, at a trade-off of reduced accuracy.

For each basis trajectory, there are multiple options for sampling and storing the trajectory. A straightforward strategy, employed by Agranovsky et al. [1], is to save only the start and end points along the pathline computed in situ. Although this strategy can provide data storage optimizations and be sufficient for approximating the pathline, it could be an oversimplification in the event of a long interval of calculation or complex vector field behavior. The algorithm introduced by Sane et al. [12] calculates basis trajectories of variable length and uniformly samples the trajectory. Although their work considers basis flow of variable length, the operations of adding or removing a basis flow can only be performed at predetermined cycles. Alternate strategies might consider various curve simplification techniques, like selecting points along a pathline that minimize its reconstruction error, or using attributes like curvature, winding angle, or linear and angular entropy to guide a temporal sampling strategy. Figure 4 uses notional examples to illustrate a comparison between uniform and attribute-guided curve sampling.

Complex temporal sampling strategies are challenging due to the limited in situ memory available and the higher memory requirements of these strategies, i.e., the requirement to store multiple points along a pathline before selecting a subset. That

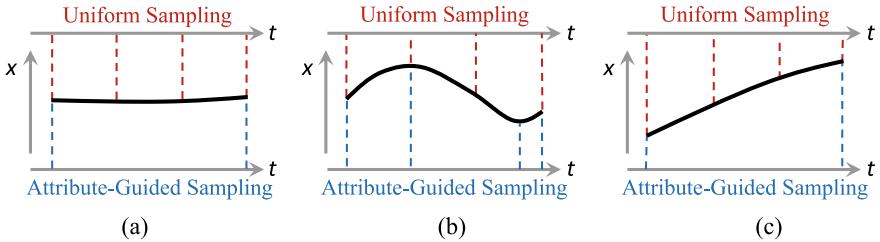


Fig. 4 This figure illustrates two different curve sampling strategies—uniform and attribute-guided—on three notional basis flows. Each basis flow is colored in black, and the data saved is indicated with dotted red lines for uniform sampling, and dotted blue lines for attribute-guided sampling. **a** considers a curve whose x value remains relatively the same across time. In this case, attribute-guided sampling can reduce storage costs and still provide the same information. **b** shows a curve whose x value first increases, then decreases before increasing again. In this case, attribute-guided sampling can place samples where they will best inform the nature of the trajectory. **c** shows a curve whose x value steadily increases across time. Its benefits are similar to (a)—reducing storage costs

said, collective smart temporal sampling across all basis trajectories could enable a higher fidelity reconstruction of the time-dependent vector field while further reducing storage costs.

4.4 Storage Format

Decisions for how to carry out spatial and temporal sampling affect the storage layout for basis trajectories. In turn, this can impact storage and memory usage, I/O times (for in situ writing and post hoc reading), and post hoc execution time (for reconstruction and interpolation).

In general, there are two options for storage: structured and unstructured. When considering a structured data set as output, the seed locations of basis trajectories need to lie along a regular grid. The implicit nature of the grid eliminates the need for storing seed locations explicitly, and promotes a natural organization of data. Trajectory data can be stored in “grid” form, with the values at each grid point being information about the seed that originated at that location. Agranovsky et al. [1] adopt this approach, and store only a single value at each grid location: the ending position of the seed particle. Of course, additional data can be stored at each grid point. For example, a field can be used to indicate whether a particle remained within the bounds of the domain during the interval of calculation. Although the use of a structured storage format enables a fast post hoc exploration workflow, it is limited to the case of spatial sampling along a regular grid. Figure 5 shows direct visualizations of a structured data output, where particle trajectories (represented as line segments) are calculated using the displacement field of a seismology simulation over three intervals of computation.

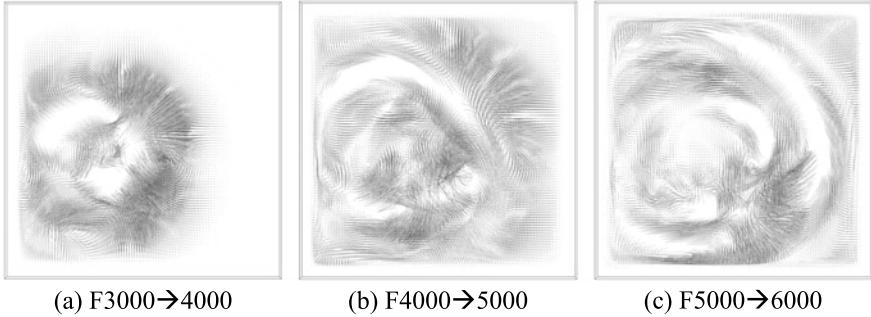


Fig. 5 This figure is an example of directly visualizing the data extracted from an in situ Lagrangian-based flow analysis routine. Each image is a visualization of a Lagrangian representation of the time-dependent vector field produced by a seismic modeling simulation studying seismic wave propagation [10]. For this example, seeds are initially placed along a regular grid and only the end points of the particles are saved at the end of an interval, i.e., a structured grid with an “end point” field is stored. In each visualization, the trajectories (represented as line segments from the grid point to the “end point”) capture the displacement caused by the underlying vector field. For example, a shows trajectories calculated between cycle 3000 and 4000 and represents the earlier stages of the simulation. The progressive propagation of the seismic waves can be perceived when considering all three images

When considering the flexibility that the Lagrangian framework provides, the unstructured storage format is a more natural fit when spatial and temporal sampling are irregular (i.e., adapted to maximize the information per byte). Particle trajectories can be stored as current point locations, lines (two points), or polylines (more than two points), each with additional attributes (for example, ID to identify data points of the same basis trajectory across files) associated with each object. This approach, however, results in a more expensive post hoc exploration process since cell location and interpolation require more elaborate search structures when considering unstructured data. Sane et al. [12] adopted an unstructured storage format to store long particle trajectories, calculated across multiple file write cycles, as line segments with identifiers. Although the adopted approach did not increase in situ costs, post hoc reconstruction required the use of an expensive search structure (Delaunay triangulation) to locate relevant basis trajectories.

5 Post Hoc Exploration

After the in situ generation of basis trajectories, post hoc exploration of the time-dependent vector field can be performed with nearly any flow visualization techniques. These techniques depend on analyzing particle trajectories, and the only difference is in how the trajectories are obtained. Where the traditional Eulerian approach calculates the trajectories via particle advection steps, the Lagrangian approach calculates the trajectories by interpolating between nearby basis trajec-

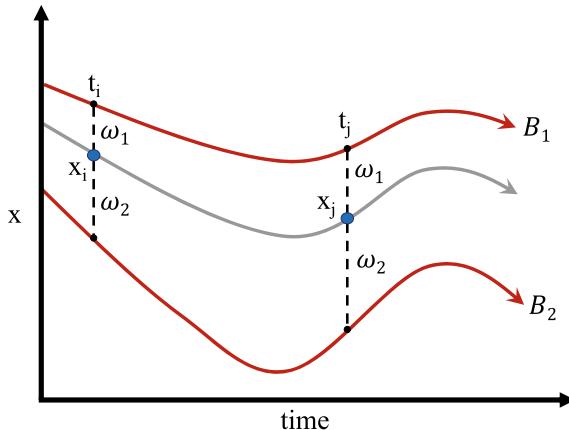


Fig. 6 Interpolation of a new particle trajectory (gray) using previously computed basis trajectories (B_1 and B_2 in red). Each position along the new particle trajectory is calculated by following basis trajectories. At time t_i , weights ω_1 and ω_2 are calculated based on the distance of B_1 and B_2 from x_i respectively. The weights are then used to estimate x_j at time t_j . This process can continue to trace the complete trajectory. This image is adapted from a version by Agranovsky et al. [1]

ries. This interpolation can be thought of as “following” basis trajectories, i.e., using them as a guide to infer where particles in between the trajectories would travel. This section discusses how to interpolate a new particle trajectory, as well as the corresponding search structures that can be used to locate nearby basis trajectories.

To calculate a new particle trajectory starting at a specific location x_i , a “neighborhood” of basis trajectories to follow needs to be identified. Typically, depending on the distances of the neighborhood basis trajectories from x_i and the selected interpolation method, different weights are assigned to each basis trajectory and then used when calculating the next location along the path of the new particle trajectory. Interpolation methods such as barycentric coordinate interpolation, Moving Least Squares interpolation, Shepard’s method, etc., can be used to calculate new particle trajectories [2, 3]. A notional example for interpolation is illustrated in Fig. 6.

The determination of the particle neighborhood is largely dependent on how the basis trajectories are stored (structured or unstructured). When the Lagrangian representation is stored using a structured data set, the neighborhood can be identified as the basis trajectories that are initialized at the grid points of the cell containing the location of the new particle trajectory to be interpolated. When using an unstructured data set, the neighborhood can be identified as the set of basis trajectories within a specific search radius or those that form a convex hull that contains the location of the new particle trajectory. Chandler et al. [6] use a modified k-d tree to perform a radius search and Sane et al. [12] use a Delaunay triangulation to identify a containing cell given a Lagrangian representation stored in an unstructured data set. Further, techniques like binning can be used to accelerate neighborhood identification.

The result of interpolating basis trajectories is a set of points that form a pathline. These computed positions along the pathline are an interval in time apart. To estimate the position of the particle in between interpolated locations, various curve fitting techniques can be used. A simple and straightforward approach to visualize the interpolated positions along a pathline is the use of a C^0 polygonal chain. This approach is used to visualize pathlines interpolated from basis trajectories in Sect. 6. As the size of the interval increases, however, the aesthetic quality of the C^0 representation of the pathline deteriorates and can be improved by using parameter curves. Bujack et al. [4] studied and evaluated the use of multiple parameter curves (cubic Hermite spline, Bézier curve) to represent particle trajectories.

The complexity of a post hoc Lagrangian-based interpolation routine is dependent on the format of the extracted Lagrangian representation of the time-dependent vector field. If basis trajectories are long and span across several simulation cycles storing multiple positions along the way, then pathline interpolation following the same neighborhood of basis trajectories results in more accurate interpolation [4, 12]. Although following short (single interval) basis trajectories is straightforward, changing the neighborhood frequently propagates a local truncation error [4, 9, 11]. There is scope for complex, yet efficient, and accurate post hoc Lagrangian-based interpolation systems to be researched and developed in the future.

6 Efficacy of Lagrangian-Based In Situ + Post Hoc Flow Analysis

For time-dependent flow visualization, the Lagrangian paradigm offers significantly improved accuracy and data storage propositions compared to the Eulerian paradigm under sparse temporal settings. This is possible because the Lagrangian representation of time-dependent vector data is capable of encoding more information per byte. The Lagrangian representation captures the behavior of the underlying vector field over an interval of time. This is in contrast to an Eulerian representation that captures the vector data at a single time slice. Further, in situ access to the complete spatiotemporal resolution of the simulation vector field enables accurate computation of the Lagrangian representation. This section highlights the efficacy of the Lagrangian-based approach for time-dependent flow visualization.

The study by Agranovsky et al. [1] demonstrated the ability of the Lagrangian representation to retain substantially better accuracy relative to the Eulerian method, even with significantly less data. For example, the study demonstrated the Lagrangian-based approach achieving comparable accuracy using a 64X data reduction. When using the same amount of storage as the Eulerian approach, the Lagrangian representation enabled over 10X more accurate reconstruction of the velocity field. Further, the study showed how increasing the interval between storing information to disk is far less detrimental to the Lagrangian-based method than the Eulerian approach. This initial quantitative evaluation was performed by comparing the “end point” of

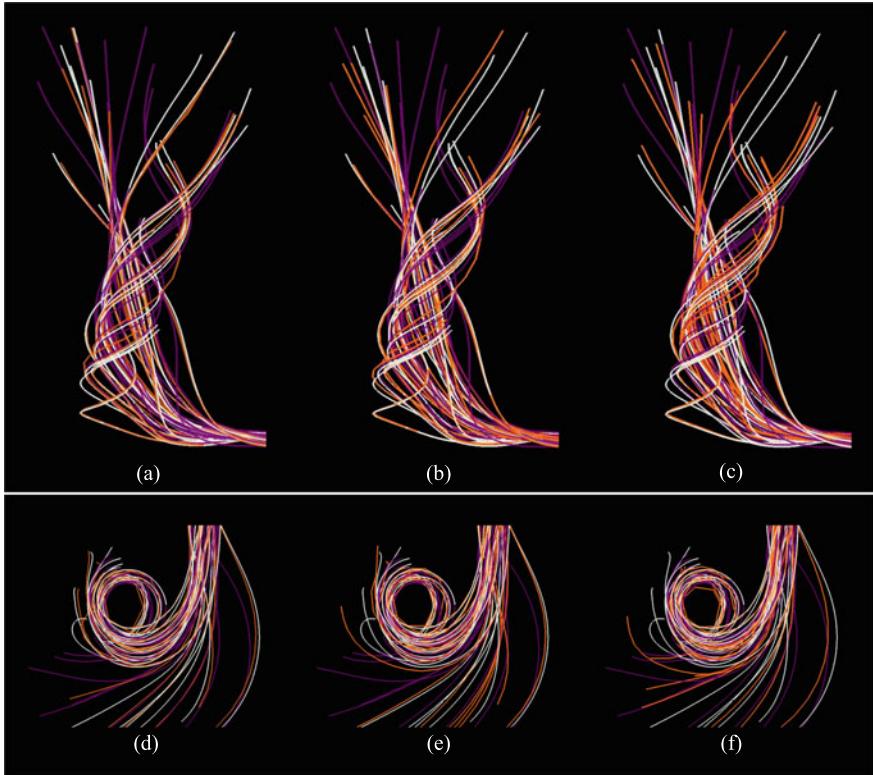


Fig. 7 Visualizations of an F-5 tornado vortex qualitatively compare the accuracy of pathlines traced using two methods under sparse temporal settings: Lagrangian (orange trajectories) and Eulerian (purple trajectories). The ground truth set of trajectories, calculated using every cycle of the simulation is traced in white. The six visualizations present results for varying configurations of how many basis trajectories are stored in the Lagrangian representation. The configurations are: 1:1 ratio of particles to grid points in (a) and (d), 1:2 in (b) and (e), and 1:4 in (c) and (f). With respect to temporal subsampling, every $8/t$ simulation cycle is stored. For all Lagrangian configurations, the method described by Agranovsky et al. [1] is used. For the post hoc visualizations, particles are initially seeded in a rake and trace trajectories that enter the tornado vortex region from the bottom-right (a–c) and top-right (d–f) of the figures. In a and d, the white (ground truth) and orange (Lagrangian) pathlines following very similar trajectories. The Eulerian method pathlines (purple) have diverged from the white trajectories in most instances. b and e show the Lagrangian representation still performing better than the Eulerian method, even though it is using only half as much storage. A similar trend can be observed in c and f where Lagrangian configurations use a quarter of the storage. In these figures Lagrangian accuracy deteriorates as the number of particle trajectories stored reduces from 1:1 to 1:4. That said, in all cases the accuracy of the Lagrangian pathlines remains higher than the corresponding Eulerian pathlines (which are calculated using the full spatial resolution)

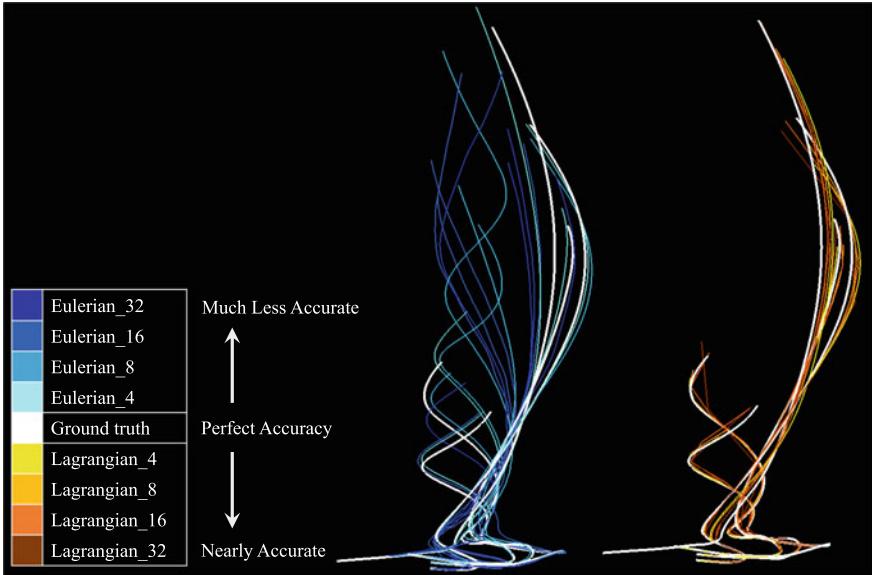


Fig. 8 Pathline visualizations compare the accuracy of pathlines traced using two methods: Eulerian and Lagrangian. Lagrangian_X and Eulerian_X denote configurations that store data every Xth cycle, i.e., the size of the interval. Left: five ground truth pathlines and their corresponding Eulerian pathlines. Overall, as the size of the interval X increases, the accuracy of the pathlines decreases and the trajectories of the Eulerian pathlines diverge from the ground truth. Right: the same five ground truth pathlines and their corresponding Lagrangian pathlines. The Lagrangian trajectories closely follow the ground truth and do not significantly diverge as the value of X increases

interpolated particle trajectories to the ground truth. Further, the study presented the accuracy of in situ Lagrangian-based flow analysis relative to the traditional Eulerian approach. Sane et al. [11] conducted a study using multiple evaluation metrics (“end point”, “every point”, “known interpolated points” of a particle trajectory) to compare the absolute errors of both the methods and observe trends across a range of spatiotemporal configurations. This study too demonstrated the significantly improved accuracy-storage propositions offered by the Lagrangian method in settings of temporal sparsity. To evaluate the benefits of using long basis flow trajectories, Sane et al. [12] conducted a study and found that accuracy-storage propositions could be improved by up to 2X in accuracy while requiring 50% less storage. However, the strategy adopted to maintain domain coverage in this study increased in situ computation costs. Each of these studies considered average error across all particle trajectories considered. Overall, with respect to a quantitative evaluation, studies exploring per particle outcomes, rather than average performance across varying non-analytical data sets, would further understanding of the approach.

To compare the Lagrangian and Eulerian methods qualitatively, Figs. 7, 8, and 9 visualize pathlines from an F-5 tornado weather simulation. The simulation has a base grid resolution of $490 \times 490 \times 280$ and a mature tornado vortex exists in the

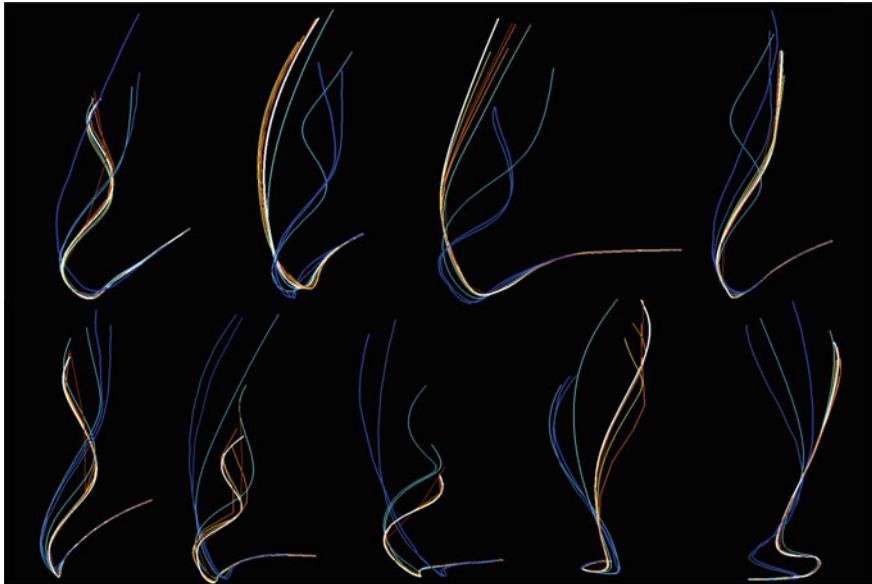


Fig. 9 Sets of pathlines traced from nine different seed locations using multiple Eulerian and Lagrangian configurations. Each set shows a ground truth pathline, and four Eulerian and Lagrangian pathlines each. The color scheme is the same as that described in Fig. 8. Overall, Lagrangian pathlines more closely follow their respective ground truth trajectories and the Eulerian pathlines are less accurate in settings of temporal sparsity, i.e., configurations with large intervals between saving data to disk

domain during the 512 simulation cycles considered for the visualization. The visualization includes ground truth pathlines, that are interpolated using every simulation cycle, and Lagrangian and Eulerian trajectories, that are computed under sparse temporal settings. In Fig. 7, Lagrangian pathline interpolation using the same number of particles as grid points (Fig. 7a) is nearly perfectly accurate, when compared to the ground truth. As the number of particles used is reduced (Fig. 7b and c), although the accuracy of Lagrangian pathline interpolation decreases, it remains more accurate than the Eulerian method. In comparison to the Lagrangian pathline interpolation, the pathlines generated by the Eulerian method trace much less accurate trajectories. Figures 8 and 9 demonstrate the effects of increasing temporal sparsity on the Lagrangian and Eulerian methods. As expected, when the size of the interval increases, the accuracy of the Eulerian pathlines decreases. In nearly every case, only the Eulerian pathline computed using a small interval size remains accurate. In contrast, the Lagrangian pathlines remain accurate and closely follow the ground truth trajectories, irrespective of the interval size.

Overall, these examples demonstrate that the Lagrangian method is capable of significantly improving the ability to perform exploratory time-dependent flow visualization by providing high-integrity reconstructions of the velocity field while requiring less data to be extracted from a simulation.

7 Discussion of State of the Art and Future Work

Over the past decade, Lagrangian methods have been increasingly used for flow visualization. Presenting a post hoc Lagrangian-based advection technique, Hlawatsch et al. [8] explored the use of a hierarchical scheme to construct longer pathlines using previously computed Lagrangian basis trajectories. The constructed pathlines would be more accurate due to being constructed using fewer integration steps. More recently, there has been an increased interest in the use of in situ processing to extract Lagrangian basis trajectories. Agranovsky et al. [1] first demonstrated the benefits of extracting a reduced Lagrangian representation to capture the behavior of a time-dependent vector field. Furthering this research direction, Chandler et al. [6] extracted a Lagrangian representation from an SPH [7] simulation and used a modified k-d tree to accelerate post hoc interpolation. Additionally, Chandler et al. [5] conducted studies to identify correlations between Lagrangian post hoc interpolation error and divergence in the velocity field. Other theoretical error analysis studies and empirical evaluations have been conducted to study the absolute error of Lagrangian-based flow analysis and error propagation during particle trajectory computation [4, 9, 11]. Building upon theoretical work, Sane et al. [12] explored the use of variable duration basis trajectories and proposed interpolation schemes to accurately compute pathlines by following long basis trajectories to reduce error propagation.

There are several future works and research directions when considering in situ Lagrangian-based flow analysis. Questions surrounding whether in situ Lagrangian-based flow analysis is capable of providing improved time-dependent flow visualization in sparse temporal settings has largely been addressed. Questions regarding best practices and the use of Lagrangian representations for specific flow visualization tasks during post hoc analysis, however, remain relatively open. That being said, existing research works provide compelling evidence to support future research efforts on in situ Lagrangian-based flow analysis to enable exploratory time-dependent flow visualization of large vector data.

Acknowledgements This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

1. Agranovsky, A., Camp, D., Garth, C., Bethel, E.W., Joy, K.I., Childs, H.: Improved post hoc flow analysis via lagrangian representations. In: 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV), pp. 67–75. IEEE (2014)
2. Agranovsky, A., Camp, D., Joy, K.I., Childs, H.: Subsampling-based compression and flow visualization. In: Visualization and Data Analysis 2015, vol. 9397, p. 93970J. International Society for Optics and Photonics (2015)
3. Agranovsky, A., Garth, C., Joy, K.I.: Extracting flow structures using sparse particles. In: VMV, pp. 153–160 (2011)

4. Bujack, R., Joy, K.I.: Lagrangian representations of flow fields with parameter curves. In: 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV), pp. 41–48. IEEE (2015)
5. Chandler, J., Bujack, R., Joy, K.I.: Analysis of error in interpolation-based pathline tracing. In: Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers, pp. 1–5. Eurographics Association (2016)
6. Chandler, J., Obermaier, H., Joy, K.I.: Interpolation-based pathline tracing in particle-based flow visualization. *IEEE Trans. Vis. Comput. Graph.* **21**(1), 68–80 (2015)
7. Gingold, R.A., Monaghan, J.J.: Smoothed particle hydrodynamics-theory and application to non-spherical stars. *Mon. Not. R. Astron. Soc.* **181**, 375–389 (1977)
8. Hlawatsch, M., Sadlo, F., Weiskopf, D.: Hierarchical line integration. *IEEE Trans. Vis. Comput. Graph.* **17**(8), 1148–1163 (2011)
9. Hummel, M., Bujack, R., Joy, K.I., Garth, C.: Error estimates for lagrangian flow field representations. In: Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers, pp. 7–11. Eurographics Association (2016)
10. Petersson, N.A., Sjögren, B.: Wave propagation in anisotropic elastic materials and curvilinear coordinates using a summation-by-parts finite difference method. *J. Comput. Phys.* **299**, 820–841 (2015)
11. Sane, S., Bujack, R., Childs, H.: Revisiting the Evaluation of In Situ Lagrangian Analysis. In: Childs, H., Cucchietti, F. (eds.) Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2018). <https://doi.org/10.2312/pgv.20181096>
12. Sane, S., Childs, H., Bujack, R.: An Interpolation Scheme for VDVP Lagrangian Basis Flows. In: Childs, H., Frey, S. (eds.) Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2019). <https://doi.org/10.2312/pgv.20191115>
13. Verma, V., Kao, D., Pang, A.: A flow-guided streamline seeding strategy. In: Proceedings of the conference on Visualization'00, pp. 163–170. IEEE Computer Society Press (2000)
14. Xu, L., Lee, T.Y., Shen, H.W.: An information-theoretic framework for flow visualization. *IEEE Trans. Vis. Comput. Graph.* **16**(6), 1216–1224 (2010)
15. Yu, H., Wang, C., Shene, C.K., Chen, J.H.: Hierarchical streamline bundles. *IEEE Trans. Vis. Comput. Graph.* **18**(8), 1353–1367 (2012)

Workflows and Scheduling

Unlocking Large Scale Uncertainty Quantification with In Transit Iterative Statistics



Alejandro Ribés, Théophile Terraz, Yvan Fournier, Bertrand Iooss,
and Bruno Raffin

Abstract Multi-run numerical simulations using supercomputers are increasingly used by physicists and engineers for dealing with input data and model uncertainties. Most of the time, the input parameters of a simulation are modeled as random variables, then simulations are run a (possibly large) number of times with input parameters varied according to a specific design of experiments. Uncertainty quantification for numerical simulations is a hard computational problem, currently bounded by the large size of the produced results. This book chapter is about using in situ techniques to enable large scale uncertainty quantification studies. We provide a comprehensive description of Melissa, a file avoiding, adaptive, fault-tolerant, and elastic framework that computes in transit statistical quantities of interest. Melissa currently implements the on-the-fly computation of the statistics necessary for the realization of large scale uncertainty quantification studies: moment-based statistics (mean, standard deviation, higher orders), quantiles, Sobol' indices, and threshold exceedance.

A. Ribés (✉)
EDF R&D, Palaiseau, France
e-mail: alejandro.ribes@edf.fr

T. Terraz · B. Raffin
Université Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
e-mail: theophile.terraz@inria.fr

B. Raffin
e-mail: bruno.raffin@inria.fr

Y. Fournier · B. Iooss
EDF R&D, Chatou, France
e-mail: yvan.fournier@edf.fr

B. Iooss
e-mail: bertrand.iooss@edf.fr

1 Introduction

A numerical simulation is a calculation that is run on a computer following a program that implements a mathematical model for a physical system. Nowadays, engineers and scientists often use numerical simulation as a tool, and its use in industries or scientific laboratories is very broad. From the mathematical point of view, discretized differential equations are numerically solved, often using a mesh as spatial support. Popular methods include Finite Difference, Finite Volumes, Finite Elements, or particle-based methods. From the computer science point of view, a numerical simulation consists of a workflow of actions. First, the engineer or scientist prepares the mesh or other spatial discretization such as particles, and she/he defines the initial and boundary conditions. Second, the calculations are run in a computer and generate results that are written to files. Finally, the results are analyzed.

Chapter “In Situ Visualization for Computational Science: Background and Foundational Topics” of this book already presented an introduction to in situ techniques for computational science. We would like to insist on the fact that engineers and scientists are so used to the classical workflow presented above, that they currently do not realize that a major change is occurring in current computer systems. The sizes of the simulations are strongly increasing, and writing files is becoming cumbersome and time-consuming. This bottleneck, in numerous cases, limits the size of the simulations. When executing multiple simulation runs, this problem becomes even more critical.

Multiple simulation runs (sometimes several thousands) are required to compute sound statistics in the context of uncertainty quantification [43]. Taking uncertainties into account when dealing with complex numerical simulations is necessary to assess their robustness, as well as to answer tighter regulatory processes (security, safety, environmental control, health impacts, etc.). Many attempts at treating uncertainty in industrial applications (e.g. automotive and aerospace engine design, nuclear safety, agronomy, renewable energy production) have involved different mathematical approaches from many scientific domains as in metrology, structural reliability, variational analysis, design of experiments, machine learning and global sensitivity analysis [11]. As an example, in aeronautic and nuclear industries, uncertainty quantification approaches are applied on numerical models simulating non destructive testing procedures, in order to evaluate probability of detection curves [26]. Such curves allow the operators to evaluate the performance of their non destructive tests for the detection of harmful defects of the inspected structure, which is particularly important for the system safety.

Current practice consists of performing multiple executions of a classical workflow. All the necessary instances with different sets of input parameters are run, and the results are stored to disk, often called ensemble data, to later read them back from disk to compute statistics. In this context, we are confronted with two important problems. First, the amount of storage needed may quickly become overwhelming, with the associated long read time that makes statistic computing time-consuming. To avoid this pitfall, scientists reduce their study size by running low-resolution sim-

ulations or down-sampling output data in space and time. Second, humans should be able to somehow navigate through the complexity of these large simulation results. Down-sampling output data in space and time, extracting probes, or concentrating on specific features of the ensemble are usually performed to reduce complexity; this introduces strong dependences on a priori ideas related to the behavior of the ensemble.

Novel approaches are required. In situ and in transit processing emerged as a solution to perform data analysis starting as soon as the results are available in the memory of the simulation. The goal is to reduce the data to store to disk and to avoid the time penalty to write and then read back the raw data set as required by the classical post hoc analysis approach. In recent works, we proposed the Melissa framework for the on-line data aggregation of high-resolution ensemble runs [39, 45]. As soon as each available simulation provides the results produced to a set of staging nodes, these nodes process them to update the statistics on a first-come-first-served basis thanks to one-pass algorithms. This in transit processing mode enables us to fully avoid storage of intermediate data on disks. Furthermore, this new approach allows the computation of *ubiquitous statistics*: we compute multidimensional and time-varying statistics, i.e. everywhere in space and time; instead of providing a down-sampled subset, for a limited sample of probes or concentrating on specific features of the ensemble, as usually done.

In the context of this book, we would like to remark that this chapter does not present any in situ visualization system but an example of how in situ techniques can be used for the statistical analysis of large quantities of data; which is defined in chapter “In Situ Visualization for Computational Science: Background and Foundational Topics” as “use cases beyond exploratory analysis.” As a matter of fact, uncertainty quantification for numerical simulations is a hard computational problem, currently bounded by the large size of the produced results. This chapter is about using in situ techniques to unleash large scale uncertainty quantification (UQ) studies.

In the following, Sect. 2 presents the general methodology for dealing with UQ studies; Sect. 3 introduces the iterative statistics necessary to perform on-line UQ; Sect. 4 briefly describes the MELISSA platform which implements the statistics introduced in Sect. 3; Sect. 5 uses a fluid mechanics example to illustrate the realization of a large scale UQ study; finally, a short conclusion and a bibliography section end up the chapter.

2 Uncertainty Management Methodology

2.1 Introduction

A general framework has been proposed in order to deal with various uncertainties that arise in numerical simulations [3, 11]. The uncertainty management generic

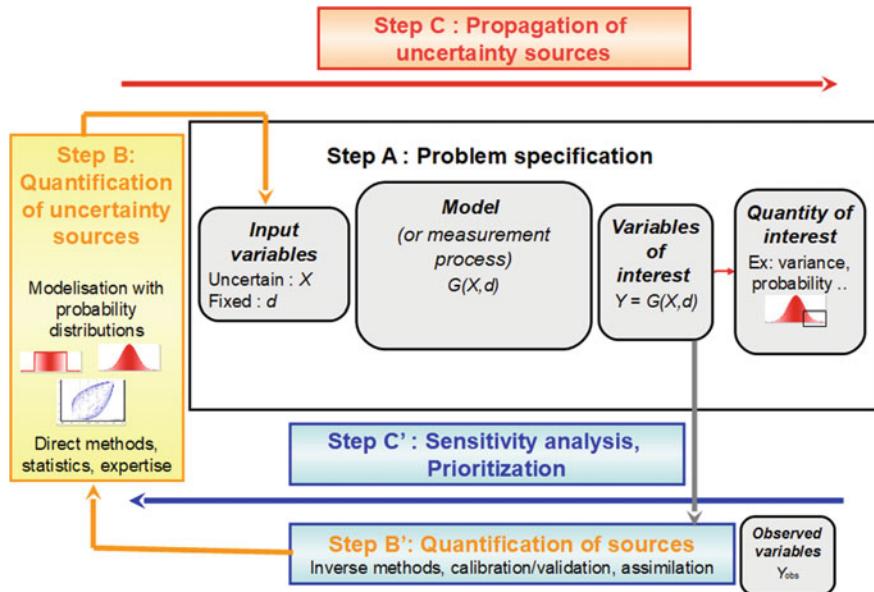


Fig. 1 The methodology of uncertainty management in numerical simulation

methodology is schematized in Fig. 1. Based on a probabilistic modeling of the model input variables, it consists of the following steps:

- Step A: specify the random inputs X , the deterministic inputs d , the numerical model G (analytical, complex computer code or experimental process), the variable of interest (model output) Y and the quantity of interest on the output (central dispersion, its distribution, probability to exceed a threshold, ...). The fundamental relation writes:

$$Y = G(X, d) = G(X),$$

with $X = (X_1, \dots, X_p) \in \mathbb{R}^p$.

- Step B: quantify the sources of uncertainty. This step consists in modeling the joint probability density function (pdf) of the random input vector by direct methods (e.g. statistical fitting, expert judgment).
- Step B': quantify the sources of uncertainty by indirect methods using some real observations of the model outputs. The calibration process aims to estimate the values or the pdf of the inputs while the validation process aims to model the bias between the model and the real system.
- Step C: propagate uncertainties to estimate the quantity of interest. With respect to this quantity, the computational resources and the CPU time cost of a single model run, various methods will be applied as linear-based analytical formula, geometrical approximations, Monte Carlo sampling strategies, metamodel-based techniques.

- Step C': analyze the sensitivity of the quantity of interest to the inputs in order to identify the most influential inputs (useful if one wants to reduce the output uncertainty) and to rank the uncertainty sources.

2.2 *Quantiles of Simulation Outputs*

Quantiles are important order statistics for outlier detection or computation of non parametric prediction and tolerance intervals. Then, in the context of uncertainty quantification analysis of computer models, quantile estimation is one of the key steps. Low or high-order quantiles are often required, especially in industrial safety studies [11, 22, 32].

Standard approaches deal with the problem of quantile estimation of scalar outputs [8, 16, 17]. Let us consider a N -sample (Y_1, \dots, Y_N) of independent and identically distributed random variables from an unknown distribution $f_Y(y)$. We look for an estimator \hat{q}_α of the α -quantile q_α defined by:

$$\mathbb{P}(Y \leq q_\alpha) = \alpha , \quad (1)$$

which is sometimes written as

$$q_\alpha = \inf\{y | \mathbb{P}(Y \leq y) \geq \alpha\} . \quad (2)$$

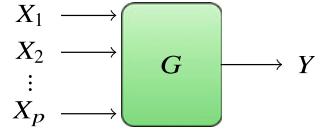
However, simulation models most often return spatial fields varying over time. For example, recent studies have considered quantiles of one-dimensional functional outputs (temporal curves) [33, 36, 38, 40], that demonstrates users' interest in computing these functional quantiles.

2.3 *Sensitivity Analysis via Sobol' Indices*

Sensitivity studies are an important application of uncertainty quantification in numerical simulation [43]. The objective of such studies can be broadly viewed as quantifying the relative contributions of individual input parameters to a simulation model, and determining how variations in parameters affect the outcomes of the simulations. In this context, multi-run studies treat simulations as black boxes that produce outputs when a set of parameters is fixed (Fig. 2). Global sensitivity analysis is an ensemble of techniques that deal with a probabilistic representation of the input parameters [21, 42] to consider their overall variation range.

Variance-based sensitivity measures, also called Sobol' indices [44], are popular among methods for global sensitivity analysis because they can deal with nonlinear responses. They decompose the variance of the output, Y , of the simulation into

Fig. 2 A simple solver G taking p input parameters X_1 to X_p , and computing a scalar output Y



fractions, which can be attributed to random input parameters or sets of random inputs.

For example, with only two input parameters X_1 and X_2 and one output Y (Fig. 2), Sobol' indices might show that 60% of the output variance is caused by the variance in the first parameter, 30% by the variance in the second, and 10% due to interactions between both. These percentages are directly interpreted as measures of sensitivity. If we consider p input parameters, the Sobol' indices can identify parameters that do not influence or influence very slightly the output, leading to model reduction or simplification. Sobol' indices can also measure the effect of interactions in non-additive systems.

Mathematically, the first and second order Sobol' indices [44] are defined by:

$$S_i = \frac{\text{Var}(\mathbb{E}[Y|X_i])}{\text{Var}(Y)}, \quad S_{ij} = \frac{\text{Var}(\mathbb{E}[Y|X_i X_j])}{\text{Var}(Y)} - S_i - S_j, \quad (3)$$

where X_1, \dots, X_p are p independent random variables. In Eq. (3), S_i represents the first order sensitivity index of X_i while S_{ij} represents the effect of the interaction between X_i and X_j . Higher-order interaction indices ($S_{ijk}, \dots, S_{1\dots p}$) can be similarly defined. The total Sobol' indices express the overall sensitivity of an input variable X_i :

$$ST_i = S_i + \sum_{j \neq i} S_{ij} + \sum_{j \neq i, k \neq i, j < k} S_{ijk} + \dots + S_{1\dots p}. \quad (4)$$

The previous formula applies for a scalar output Y . Some authors have proposed the generalization of the concept of Sobol' indices for multidimensional and functional data [14] by synthesizing all the sensitivity information of the multidimensional output in a single sensitivity value. Few authors have considered the estimation of Sobol' indices at each output cell (see [31] for an overview on this subject) and this estimation has always been applied to small models. Applications of these techniques on environmental assessment can be found for example in [18, 29] for spatial outputs and in [28, 30] for spatio-temporal outputs. All these works have shown that obtaining temporal/spatial/spatio-temporal sensitivity maps leads to powerful information for the analysts. Indeed, the parameter effects are localized in time or space, and can be easily examined in relation with the studied physical phenomena.

3 In Transit Statistics

Computing statistics from N samples classically requires $O(N)$ memory space to store these samples. But if the statistics can be *computed in one-pass* (also called iterative, on-line or even parallel [34]), i.e. if the current value can be updated as soon as a new sample is available, the memory requirement goes down to $O(1)$ space. With this approach, not only simulation results do not need to be saved, but they can be consumed in any order, loosening synchronization constraints on the simulation executions.

There also exist multi-pass algorithms for the computation of statistics, where P passes are necessary. In this case, the on-line processing system would need to have access to the same data P times thus forcing the data to be stored till the last pass is finished. This is not feasible for large scale use cases. These approaches, along with the classical ones requiring $O(N)$ memory space, are avoided in on-line applications.

3.1 Moment-Based Statistics: Mean, Std, Higher Orders

One-pass variance algorithms were proposed in [9, 12, 47]. Numerically stable, one-pass formulas for arbitrary centered statistical moments and co-moments are presented in [5, 34]. Reference Pébay et al. [34] also contains update formulas for higher order moments (skewness, kurtosis and more). These works set the base for a module of parallel statistics in the VTK scientific visualization toolkit [35]. In this context, the one-pass algorithms enable computation of partial results in parallel before performing a reduction to get the final result. These iterative statistics were used for computing large scale parallel statistics for a single simulation run either from raw data files [7], compressed data files [24] or in situ [6]. More recently Lampitella et al. [25] proposed a general update formula for the computation of arbitrary-order, weighted, multivariate central moments.

In Melissa, we iteratively compute the moments of a random variable Y as

$$\mu_{(k),\mathcal{S}}(Y) = \mu_{(k),\mathcal{S}} = \mu_{(k),\mathcal{S}_1} + \frac{1}{n} (y^k - \mu_{(k),\mathcal{S}_1}) \quad (5)$$

for $k = 1, 2, 3$ and 4 , where $\mathcal{S} = \mathcal{S}_1 \cup \{y\}$ and $n = \text{card}(\mathcal{S})$. Then from these moments, we compute the mean, variance, skewness and kurtosis:

$$\begin{aligned} \text{Mean} &= \mu_{(1)}, \\ \text{Variance} &= \frac{n}{n-1} (\mu_{(2)} - \mu_{(1)}^2), \\ \text{Skewness} &= \frac{\mu_{(3)} - 3\mu_{(1)}\mu_{(2)} + 2\mu_{(1)}^3}{(\mu_{(2)} - \mu_{(1)}^2)^{1.5}}, \end{aligned}$$

$$\text{Kurtosis} = \frac{\mu_{(4)} - 4\mu_{(1)}\mu_{(3)} + 6\mu_{(1)}^2\mu_{(2)} - 3\mu_{(1)}^4}{(\mu_{(3)} - 3\mu_{(1)}\mu_{(2)} + 2\mu_{(1)}^3)^2} .$$

3.2 Sobol' Indices

The information contained in this section can be found in a previously published article [45]. We include it here for self-completeness. Thus the reader can find the description, in this book chapter, of all iterative methods currently implemented in Melissa.

In order to compute Sobol' indices, we use the so-called pick-freeze scheme that uses two random independent and identically distributed samples of the model inputs [19, 23, 44]. One-pass iterative Sobol' indices formulas directly derive from the iterative variance (presented in Sect. 3.1) and iterative covariance [34]. Note that another iterative computation of Sobol' indices has been introduced in [15] for the case of a scalar output.

Our goal is to compute in transit the Sobol' indices of each input parameter X_i (Fig. 2). We explain below the so-called pick-freeze scheme that uses two random independent and identically distributed samples of the model inputs [19, 23, 44].

We first define the p variable input parameters of our study as a random vector, with a given probabilistic law for each parameter. We then randomly draw two times n sets of p parameters, to obtain two matrices A and B of size $n \times p$ (each row is a set of parameters for one simulation):

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,p} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,p} \end{pmatrix}; B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,p} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,p} \end{pmatrix} .$$

For each $k \in [1, p]$ we define the matrix C^k , which is equal to the matrix A but with its column k replaced by column k of B . Each row of each matrix is a set of input parameters:

$$C^k = \begin{pmatrix} a_{1,1} & \cdots & a_{1,k-1} & b_{1,k} & a_{1,k+1} & \cdots & a_{1,p} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{i,1} & \cdots & a_{i,k-1} & b_{i,k} & a_{n,k+1} & \cdots & a_{i,p} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,k-1} & b_{n,k} & a_{n,k+1} & \cdots & a_{n,p} \end{pmatrix} .$$

Then, a study consists in running the $n \times (p + 2)$ simulations defined by the matrices A , B and C^k for $k \in [1, p]$. For each matrix M with n rows, $\forall i \in [1, n]$, let M_i be the i th row of M , and $M_{[i]}$ the matrix of size $i \times p$ built from the i first lines of M .

For example :

$$C_i^k = (a_{i,1} \cdots a_{i,k-1} b_{i,k} a_{i,k+1} \cdots a_{i,p})$$

and

$$C_{[i]}^k = \begin{pmatrix} a_{1,1} \cdots a_{1,k-1} & b_{1,k} & a_{1,k+1} \cdots a_{1,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i,1} \cdots a_{i,k-1} & b_{i,k} & a_{i,k+1} \cdots a_{i,p} \end{pmatrix}.$$

Let Y_i^A be the result of $G(A_i)$, and $Y^A \in \mathbb{R}^n$ the vector built from component i of Y_i^A , $\forall i \in [1, n]$. We define Y_i^B and Y^B in the same way, as well as $Y_i^{C^k}$ and $Y^{C^k} \forall k \in [1, p]$. Let $\text{Var}(x)$ be the unbiased variance estimator, and $\text{Cov}(x, y)$ the unbiased covariance estimator, as defined in [34]. First order Sobol' indices S_k can be estimated by the following formula, called Martinez estimator [2]:

$$S_k(f, A, B) = \frac{\text{Cov}(Y^B, Y^{C^k})}{\sqrt{\text{Var}(Y^B)} \sqrt{\text{Var}(Y^{C^k})}}, \quad (6)$$

while total order Sobol' indices ST_k are estimated by:

$$ST_k(f, A, B) = 1 - \frac{\text{Cov}(Y^A, Y^{C^k})}{\sqrt{\text{Var}(Y^A)} \sqrt{\text{Var}(Y^{C^k})}}. \quad (7)$$

Since variances and covariances can be updated iteratively, first order and total Sobol' indices can be computed from these formulas. The covariance update formula between two random variables X and Y writes [34]:

$$\text{Cov}_{\mathcal{S}} = \text{Cov}_{\mathcal{S}_1} + \frac{n-1}{n} [x - \mu_{(1), \mathcal{S}_1}(X)] [y - \mu_{(1), \mathcal{S}_1}(Y)] \quad (8)$$

where $\mathcal{S} = \mathcal{S}_1 \cup \{x, y\}$, $n = \text{card}(\mathcal{S})$ and the mean update $\mu_{(1)}$ comes from Eq. (5).

There are many other estimators than those of Eqs. (6) and (7) (see for example [37]) relying on the matrices A , B and C^k to compute the variance and the covariance with different formulas. We use the Martinez estimator because it provides an asymptotic confidence interval [2], which is very simple to express, and is easy to compute in an iterative fashion. In addition, it has been shown to be unbiased and one of the most numerically stable estimators.

3.3 Order Statistics: Quantiles

The classical estimator of the α -quantile y_α of the random variable Y is the empirical quantile, based on the notion of order statistics [10]. Essentially, we associate with the independent and identically distributed sample (Y_1, \dots, Y_N) the ordered sample

$(Y_{(1)}, \dots, Y_{(N)})$ in which $Y_{(1)} \leq \dots \leq Y_{(N)}$. The empirical estimator is then:

$$\hat{q}_\alpha = Y_{(\lfloor \alpha N \rfloor + 1)}, \quad (9)$$

where $\lfloor x \rfloor$ is the integer part of x .

For an iterative statistical estimation, the Robbins-Monro estimator [41] consists of updating the quantile estimate $q_\alpha(n)$ at each new observation Y_{n+1} with the following rule:

$$q_\alpha(n+1) = q_\alpha(n) - \frac{C}{n^\gamma} (\mathbb{1}_{Y_{n+1} \leq q_\alpha(n)} - \alpha), \quad (10)$$

with $n = 1 \dots N$, $q_\alpha(1) = Y_1$ an independent realization of Y , $\hat{q}_\alpha = q_\alpha(N)$, $\mathbb{1}_x$ the indicator function, C a strictly positive constant and $\gamma \in]0, 1]$ the step of the gradient descent of the stochastic algorithm. Under several hypotheses with $\gamma \in]0.5, 1]$, this algorithm has been shown to be consistent and asymptotically normal. A fine tuning of the constant C is important; several numerical tests have shown that a value of C of the order of the dispersion of Y (for example its standard deviation or an interquantile interval) would be satisfactory [20].

Asymptotically (where N is large), a value $\gamma = 1$ is known to be optimal. However, in practical studies, N is often not large. For example, in nuclear safety studies (see for example [8, 22]), $\alpha = 0.95$ and N is in the order of several hundreds of simulated values. In this case and as we look for γ values that can work for different distributions of Y (which are unknown in practice), we propose to define γ as a function of n . Indeed, one can observe that a good γ value for a certain type of probability distribution produces bad results for another type of distribution (for example, $\gamma = 0.6$ gives good results for a normal distribution and incorrect quantile estimates for a uniform one). We then use the following heuristic formula for γ :

$$\gamma(n) = 0.1 + 0.9 \frac{n-1}{N-1}. \quad (11)$$

The idea is to have strong mixing properties at the beginning of the algorithm (with small γ), then to slow down the potential variation of the quantile estimation all along the iterations of the algorithm.

Slightly different linear profiles can be proposed as $\gamma(n) = 0.5[1 + (n-1)/(N-1)]$. Several tests on simple analytical functions (where the true quantile can be known) have been performed in order to calibrate and validate these γ -profiles [20, 39]. Other algorithmic developments are currently under study to further improve the robustness of the Robbins-Monro estimate.

3.4 Probability of Threshold Exceedance

If y_{crit} is a safety output value, a classical failure probability estimation problem occurs:

$$p_f = \mathbb{P}(Y > y_{\text{crit}}) , \quad (12)$$

well-known in structural reliability [4, 27]. For this issue, without loss of generality, we can turn to:

$$p_f = \mathbb{P}(Y < 0) .$$

Computing a failure probability can be seen as a direct problem of uncertainty propagation [32].

If the failure domain is defined by $\mathcal{D}_f = \{x \in \chi \subseteq \mathbb{R}^p \mid G(x) \leq 0\}$, the probability that the event – failure occurs is given by

$$p_f = \mathbb{P}(G(X) \leq 0) = \int_{\mathcal{D}_f} f_X(x) dx = \int_{\chi} \mathbb{1}_{G(x) \leq 0} f_X(x) dx = \mathbb{E}[1_{G(X) \leq 0}] , \quad (13)$$

where f_X is the joint probability density function of X . One of the goals of a structural reliability study is to provide an estimate of p_f and the uncertainty involved. The complexity of models and large potential number of input variables means that, in general, we cannot calculate the exact probability of failure. The evaluation of the integral in formula (13) is the subject of numerous mathematical techniques, laid out in an abundant array of international scientific literature [11, 43]. The use of Monte Carlo simulation methods is the most common. The naive Monte Carlo estimator is

$$\hat{p}_f = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{G(x^{(i)}) \leq 0\}} , \quad (14)$$

where the $x^{(i)}$ are n independent and identically distributed random vectors simulated according to f_X . This is an unbiased estimate of the quantity of interest, for which it is possible to control the precision via its variance and the provision of a confidence interval (thanks to the central limit theorem).

Of course, as this estimation is based on an expectation, the trivial iterative algorithm for the mean (see Sect. 3.1) applies to compute p_f on the fly.

4 The Melissa Framework

Melissa (Modular External Library for In Situ Statistical Analysis) proposes a new approach to compute statistics at large scale by avoiding storage of the intermediate results produced by multiple parallel simulation runs. The key enabler is the use of iterative formulations for the required statistics. This allows for updating statistics

on-the-fly each time new simulation results are available. To manage the simulation runs as well as the in transit computation of iterative statistics, we developed a full framework built around an elastic and fault tolerant parallel client/server architecture. The benefits of this framework are multiple:

- **Storage saving:** no intermediate files are generated. Melissa fully avoids storage of intermediate data on disks.
- **Time saving:** simulations run faster when sending data to the server than when writing their results to disk. Statistics are computed while simulations are running, saving the time of post hoc statistic computing that, in addition, requires time to read back simulation results once all are performed.
- **Ubiquitous:** performance and scalability gains enable computing ubiquitous multidimensional and time varying statistics, i.e. everywhere in space and time, instead of providing statistics for a limited sample of probes as usually done with post hoc approaches to reduce the amount of temporary data storage.
- **Adaptive:** simulations can be defined, started or interrupted on-line according to past runs behavior or the statistics already computed.
- **Elasticity:** Melissa enables the dynamic adaptation of compute resource usage according to availability. Simulations are independent and connect dynamically to the parallel server when they start. They are submitted as independent jobs to the batch scheduler. Thus, the number of concurrently running simulations providing data to the server can vary during the course of a study to adapt to the availability of compute resources.
- **Fault tolerance:** Melissa's asynchronous client/server architecture supports a simple yet robust fault tolerance mechanism. Only some lightweight bookkeeping and a few heartbeats are required to detect issues and restart the server or the simulations, with limited loss of intermediate results.

4.1 *Melissa Architecture*

Melissa is an open source framework¹ that relies on a three tier architecture (Fig. 3). The *Melissa Server* aggregates the simulation results and updates iterative statistics as soon as a new result is available. The *Melissa clients* are the parallel simulations, providing their outputs to the server. *Melissa Launcher* interacts with the batch scheduler and server, for creating, launching, and supervising the server and clients. We present in this section an overview of the Melissa architecture. Please refer to [45] for more details.

¹ <https://melissa-sa.github.io>.

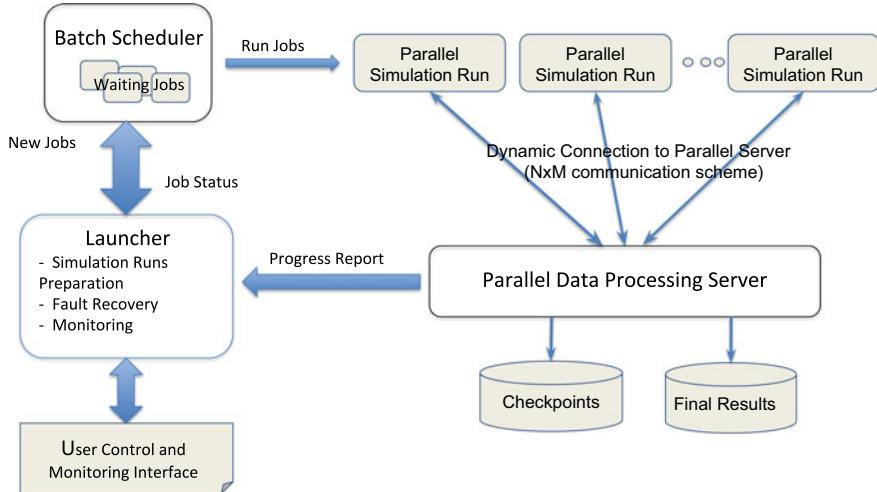


Fig. 3 Melissa three tier architecture. The launcher oversees the execution in tight link with the batch scheduler. The job scheduler regulates the number of simulation jobs to run according to the machine availability, leading to an elastic resource usage. The parallel server, started first, processes incoming data as soon as received from the connected simulations. A fault tolerance mechanism automatically restarts failing simulation runs or a failing parallel server

4.1.1 Melissa Server

Melissa Server is parallel and runs on several nodes. The number of nodes required for the server is driven by (1) memory needs and (2) data pressure. The amount of memory needed for each computed statistic field is of same order as the size of the output field of one simulation (number of timesteps \times the number of cells or points in the mesh). The number of server nodes should also be large enough to process incoming data without stalling the simulations.

4.1.2 Dynamic Connection to Melissa Server

When a simulation starts, it dynamically connects to the Melissa Server. Each simulation process opens individual communication channels to each necessary server process for enabling a $N \times M$ data redistribution. Every time new results are available, simulation processes push their results toward the Melissa Server.

Melissa is designed to keep intrusion into the simulation code minimal. Melissa provides 3 functions to integrate in the simulation code through a dynamic library. The first function (Initialize) allocates internal structures and connects the simulation to the server. At each timestep, the second function (Send) sends the simulation data to its corresponding Melissa Server processes. The third function (Finalize) disconnects the simulation and releases the allocated structures.

4.1.3 Melissa Launcher

Melissa Launcher takes care of generating the parameter sets, requesting the batch scheduler to start the server and the clients, and tracking the progress of the running server and clients jobs. It first submits to the batch scheduler a job for the Melissa Server. Then, the launcher retrieves the server node addresses (the server is parallelized on several nodes) and submits the simulation jobs. Each simulation is submitted to the batch scheduler as a standalone job, making Melissa very elastic, i.e. capable of adapting to a varying compute resource availability. Simulations can be submitted all at once or at a more regulated pace depending on the machine policy for job submissions.

4.1.4 Fault Tolerance

The Melissa asynchronous client/server architecture leverages the iterative statistics computations to support a simple yet robust fault tolerance mechanism. Melissa supports detection and recovery from failures (including straggler issues) of Melissa Server and simulations, through heartbeats and server checkpointing. Melissa Launcher communicates with the server and the batch scheduler to detect simulation or server faults. As every simulation runs in a separate job, the failure of one simulation does not impact the ongoing study: Melissa launcher simply restarts it and the server discard already processed messages. Please refer to [45] for a complete description of this fault tolerance system.

5 An Illustrative Example

5.1 A Large Scale Study

We used *Code_Saturne* [1], an open-source computational fluid dynamics tool designed to solve the Navier-Stokes equations, with a focus on incompressible or dilatable flows and advanced turbulence modeling. *Code_Saturne* relies on a finite volume discretization and allows the use of various mesh types, using an unstructured polyhedral cell model, allowing hybrid and non-conforming meshes. The parallelization [13] is based on a classical domain partitioning using MPI, with an optional second (local) level using OpenMP.

5.1.1 Use Case

We validated our implementation on a fluid mechanics demonstration case simulating a water flow in a tube bundle (Fig. 4). The mesh is composed of 6 002 400 hexahedra.

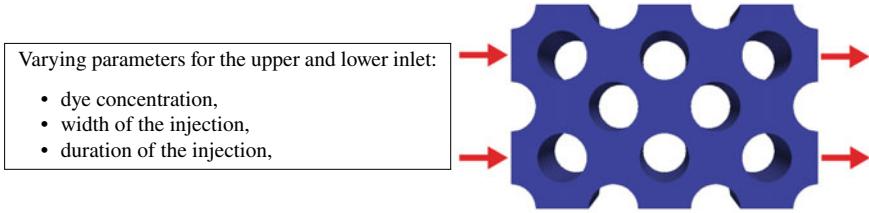


Fig. 4 Use case: water flows from the left, between the tube bundle, and exits to the right with 6 varying parameters

We generate a multi-run sensitivity study by simulating the injection of a tracer or dye along the inlet, with 2 independent injection surfaces, each defined by three varying parameters (Fig. 4). The solved scalar field representing dye concentration could be replaced by temperature or concentration of chemical compounds in actual industrial studies.

To initialize our multi-run study, we first ran a single 1000 timesteps simulation, to obtain a steady flow. Each simulation consisted of 100 timesteps starting from this steady flow, with different parameter sets.

This study ran a total of 80 000 simulations for computing ubiquitous variances, covariances, Sobol' indices, and the 5th, 25th, 50th, 75th and 95th percentiles on the 6M hexahedra and 100 timesteps. Sobol' index computations rely on the pick-freeze method that requires to run groups of simulations with non-independent parameter sets. These correlated simulations are not used for the quantiles that are computed from the remaining independent 20 000 simulations.

The study took a total 260 000 CPU hours for the simulations and 11 112 CPU hours for the server (4% of the total CPU time). Melissa Server processed on-line a cumulated total of 288 TB of data coming from the simulations.

5.2 Ubiquitous Statistic Interpretation

In this section we interpret the ubiquitous statistics computed during the experiments. By ubiquitous statistics we mean the statistics of multidimensional and time varying physical quantities, i.e. statistics everywhere in space and time. Using ParaView we have chosen a timestep and performed a slice on a mid-plane of the mesh presented in Fig. 4. This slice is aligned with the direction of the fluid. The chosen timestep belongs to the last temporal part of the simulation (80th timestep over 100). This operation reduces the ubiquitous statistics to 2D spatial maps, thus allowing us to generate the images for this section. We remark that this is just a choice for illustration purposes, and any other visualisation pipeline can be applied to the ubiquitous statistics.

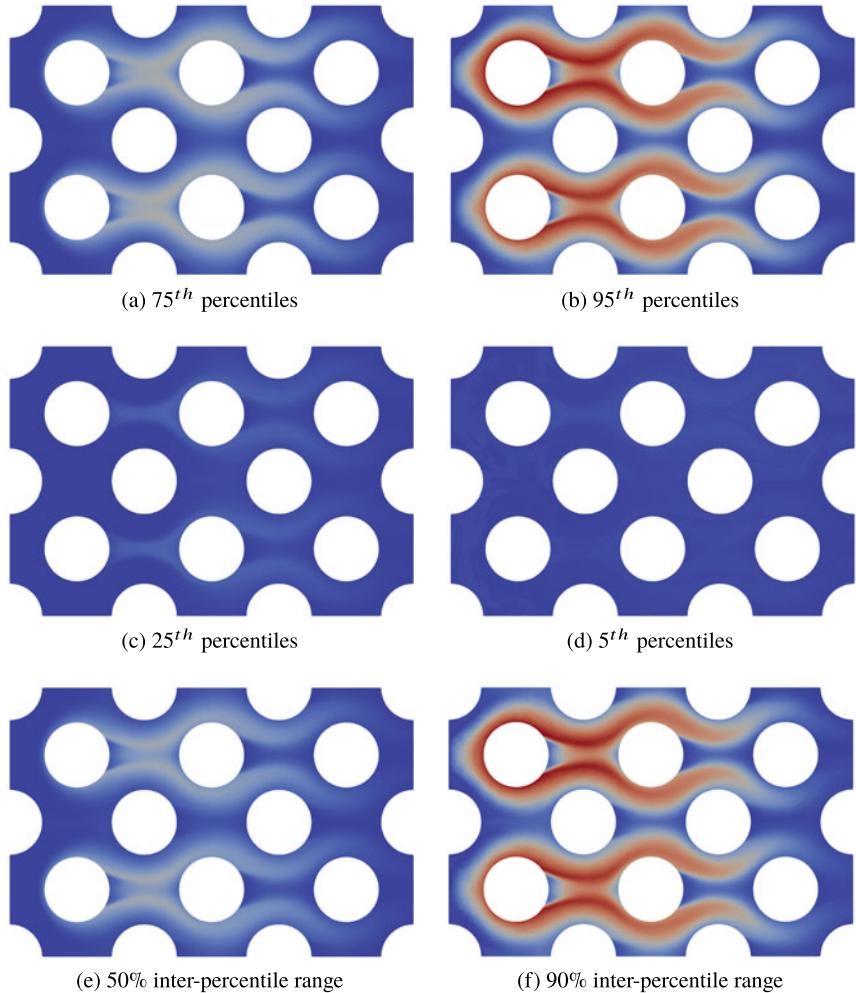


Fig. 5 Percentiles and inter-percentile ranges maps on a slice of the mesh at timestep 80. The top two lines correspond to percentiles while the bottom line corresponds to inter-percentile ranges. All maps share the same scale

5.2.1 Quantiles

Figure 5 presents six spatial maps extracted from the ubiquitous quantiles. We concentrate in percentiles because they are easily interpreted. We recall that a percentile is not a per cent but a type of quantile: a 100-quantile. Our system can also calculate 4-quantiles, the so-called quartiles that are popular measures in statistics, or any other kind of quantile.

On the four top panels of Fig. 5, Fig. 5a–d, we present the 75th, 95th, 25th, 5th percentiles, respectively. On the two bottom panels the interpercentile ranges containing 50% and 90% of the samples are shown. Interpercentile ranges are easily computed from percentiles by subtraction: the 50% interpercentile range corresponds to the 75th percentile minus the 25th percentile; the 90% interpercentile range corresponds to the 95th percentile minus the 5th percentile. In Fig. 5 each column shows an interpercentile map on the bottom and the percentile maps that served for its calculation above it. Looking at these maps an analyst can deduce several things:

1. Extreme high percentile maps such as 95th, Fig. 5b, give an idea of the distribution of the upper bounds of all simulations. In our use case, we can assess which spatial areas contain low quantities of contaminant. Indeed areas coloured in blue necessarily contain low contaminant concentrations for any simulation in the multi-run study. Extreme low percentile maps, such as 5th, has also a direct interpretation in the opposite sense.
2. Interpercentile-range maps, such as Fig. 5e or f, are maps that show the spatial variability of statistical dispersion. Indeed, scalar interpercentile ranges are non-parametric measures of statistical dispersion, which means that no a priori knowledge about the distribution of the data is needed. This characteristic makes these ranges both general and robust. Visualising a map of such a measure of dispersion enables understanding of how the data distribution is spatially concentrated. In our use case the low percentile maps used to calculate the interpercentile maps are mainly close to zero for all cells of the mesh, which makes these maps resemble the higher percentiles maps. However, this is in general not true. Moreover, if we visually compare Fig. 5a, e we realise that both maps are different.

The maps shown in Fig. 5 are static and 2D but we recall that we calculate ubiquitous percentiles, thus 3D and time dependent data is available. Figure 6 shows the temporal evolution of a probe positioned in the mesh using ParaView. At a specific location, a temporal evolution of all computed quantiles can be performed. In Fig. 6b this evolution is plotted for the 95th, 75th, 25th and 5th percentiles. The vertical line

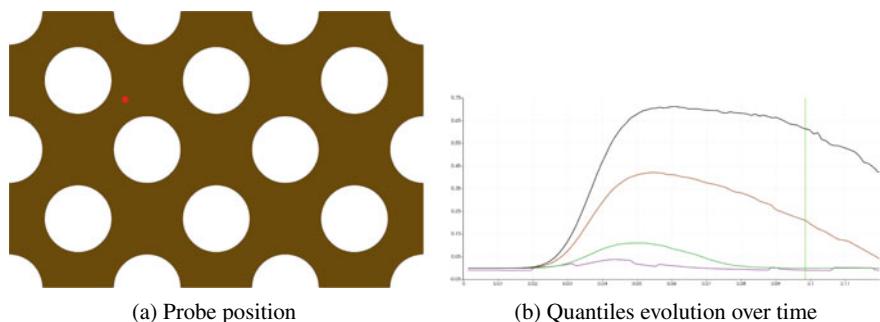


Fig. 6 A probe in a cell of the mesh allows an extraction of the temporal evolution of percentiles at a specific spatial location

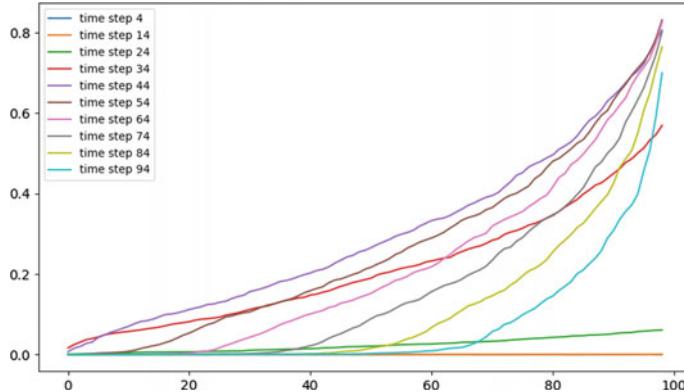


Fig. 7 Percentile functions of the dye concentration at different time steps of the simulation. Vertical axis represents dye concentration. Horizontal axis represents percentiles. Each curve corresponds to different time steps of the simulation. All curves have been extracted from the probe position shown in Fig. 6a

indicates the position of the current time step (80th time step). This figure clearly shows how the output variability of the ensemble study depends on time. Indeed, all simulations contain no dye for the first 15 time-steps, which is the time the dye takes to propagate from the top injector to the spatial location of the probe. After this point, we observe a moment where the variability of the dye concentration is the highest before a general decrease.

Figure 6b can be seen as the evolution of a Tukey boxplot [46] over time. In fact, the 25th and 75th percentiles correspond to the 1st and 3rd quartiles thus delimit the central box of the plot, while the 5th and 95th quantiles can be a choice for the whiskers. Using this analogy, we can easily observe that the dispersion of the dye concentration on the whole ensemble moves over time. Furthermore, the distribution of this quantity is not symmetrical and its asymmetry is evolving over time.

Finally, Fig. 7 shows a different representation of the evolution of the dye concentration at a fixed probe (the same than in Fig. 6). At different regularly sampled time steps, the quantile functions of the concentration values are plotted (as a function of the order of the quantiles, between 0% and 100%). One can first observe zero-valued quantile functions for the first time steps (time steps 4 and 14). Indeed, at the probe, the dye concentration is zero during the first times of the injection. Then, from time step 24 to time step 44, all the values of the quantile functions regularly increase. It means that the dye concentration values homogeneously increase from 0, reaching a maximal value close to 0.82 for the 100%-order quantile. At the end of the simulation time, from time step 54 to time step 94, the quantile functions are regularly displaced to the right. The values close to zero disappear and the concentration of strong values becomes more and more important. As a conclusion, thanks to the quantile functions, this graph allows to finely and quantitatively analyze the temporal evolution of this dye concentration phenomena. We remark that, because

we have calculated the ubiquitous percentiles, it is possible to obtain Figs. 6 and 7 for any location on the simulation domain.

5.2.2 Sobol' Indices

In this section we interpret the Sobol' indices computed during the experiments. Figure 8 presents six spatial first order Sobol' maps extracted from the ubiquitous indices. Looking at these figures an analyst can deduce several things:

1. The **width of the injections** influence locations far up or down each injector (Fig. 8c, d). This is because the injectors are located in the center of the inlet segments, as depicted by the arrows in Fig. 4. The parameter controlling the aperture of an injector makes a thinner or wider aperture around a central position. When the aperture's width is small, all dye is injected on the center while very few will go up and down. On the opposite, when the injector is at its maximal aperture, dye can easily flow to extreme vertical locations. In any case, the dye always flows in the center of the injectors which makes this parameter not influential around these areas.
2. The Sobol' maps for the **duration of the injection** (Fig. 8e, f) can be understood by thinking about the temporal evolution of the simulation. When the simulations are just started, they all inject dye and the duration represents the time at which this injection is stopped. Figure 8 represents a state near the end of the simulation. Thus, it is not surprising that this parameter does not influence the right part of the domain (where all simulations were injecting dye) while it strongly influences the left side (where some simulations already stopped injecting while others continued).
3. Finally, the **dye concentration** mostly influences the areas where the other parameters have less influence, that is to say the center of the top and bottom channels, and the right side of the flow (Fig. 8a, b).

We recommend co-visualizing Sobol' indices with the variance of the whole model output; Fig. 8g shows this variance map for the Sobol' indices presented in Fig. 8. One of the reasons of this co-visualization is that $Var(Y)$ appears as a denominator in Eq. (3), consequently when $Var(Y)$ is very small or zero, the Sobol' indices have no sense due to numerical errors or can even produce a zero division. Furthermore, it is not conceptually interesting to try to understand which input parameters influence low variance areas of the simulation once we know that “not much happens” in these areas.

5.3 Combining Sobol' Indices

The sum of Sobol' indices should be 1 and they partition the total variance of the outputs. These two characteristics allow Sobol' indices to be grouped (added) or sub-

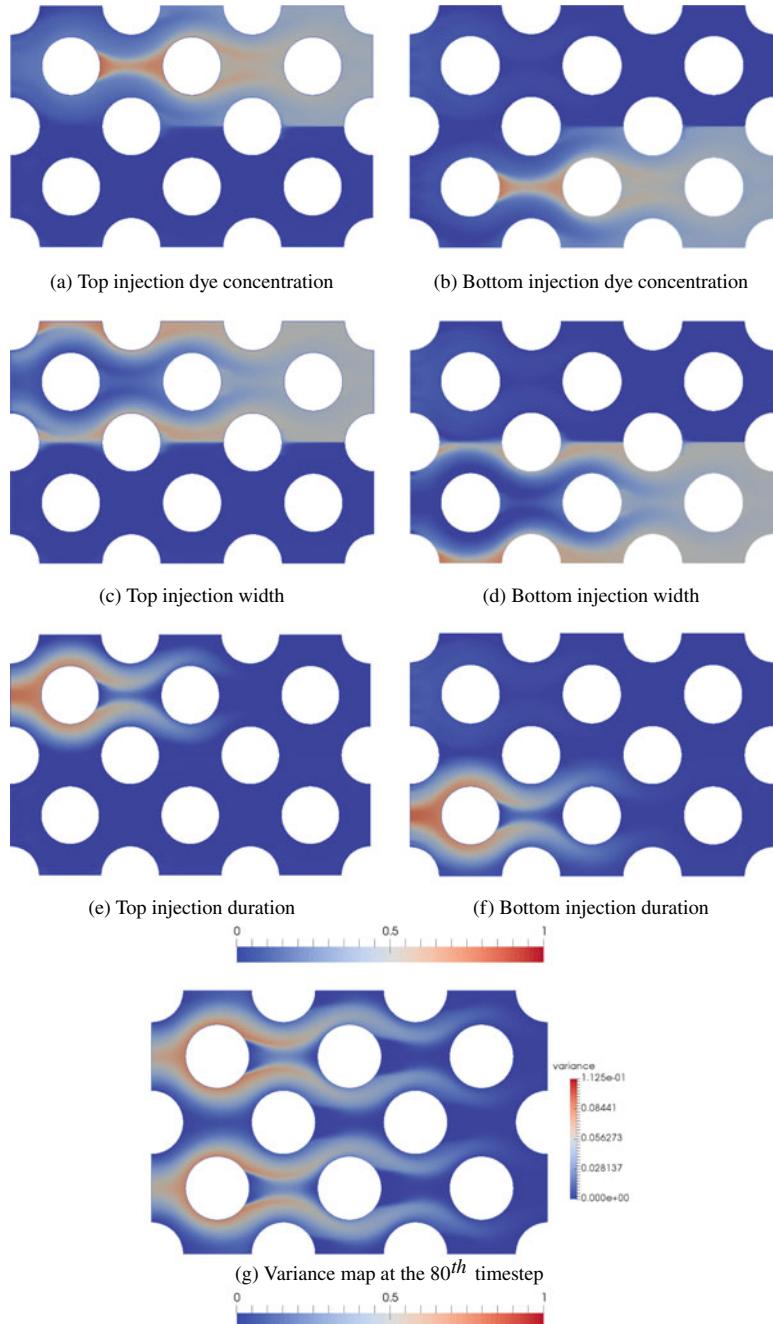


Fig. 8 First order Sobol' index maps on a slice of the mesh at timestep 80. The left column corresponds to the Sobol' indices for the upper injector while right corresponds to the bottom injector. All maps are scaled between zero (blue) and one (red)

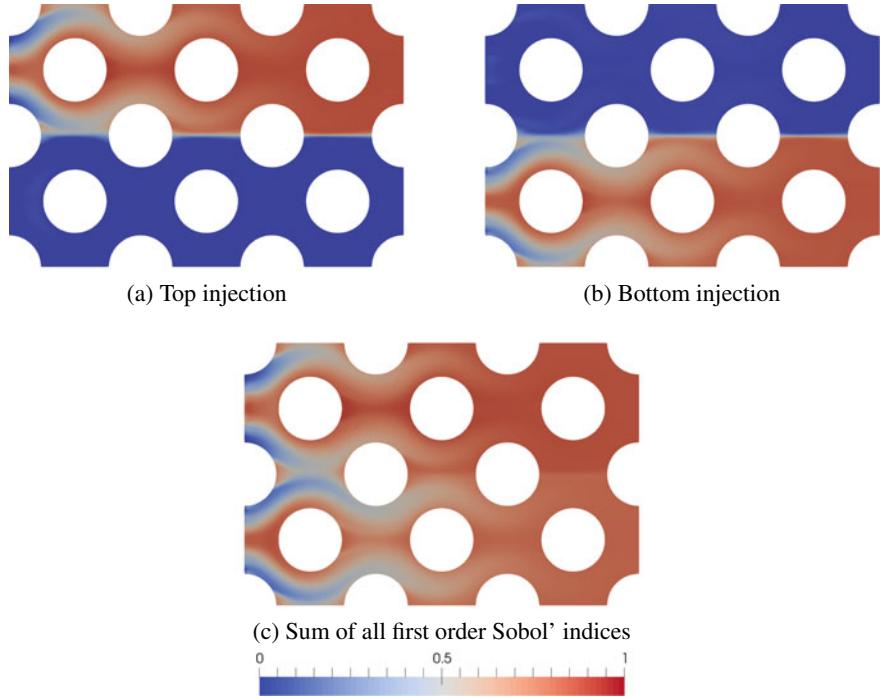


Fig. 9 Addition of first order Sobol' index maps on a slice of the mesh at timestep 80. The left panel corresponds to the addition of the Sobol' indices for the upper injector while right corresponds to addition for the bottom injector. All maps are scaled between zero (blue) and one (red)

tracted. If the indices are added, the resulting “Sobol group index” keeps normalised between zero and one. We show two examples of the use of such groups:

1. Figure 9a corresponds to the sum of the Sobol' indices for the upper injector while Fig. 9b corresponds to the sum for the bottom injector. We clearly observe that the three parameters that define the behavior of the upper injector have no influence in the lowest half part of the simulation domain. This phenomenon can also be observed by simultaneously looking at the Sobol' maps of the three parameters for the upper injector (Fig. 8a, c, e). However, the grouped Sobol map is clearer and gives a straightforward answer to the question “What is the influence of the upper injector?” Respectively, the parameters of the bottom injector do not influence the upper part of the simulation as directly observed on Fig. 9b (or looking at Fig. 8b, d, f).
2. We also calculate $S_1 + \dots + S_n$, with $n = 6$, shown in Fig. 9c, which gives us an indication of the importance of the interactions between parameters. Red areas of Fig. 9c indicate that the first order Sobol indices are responsible of the behaviour of the simulation. On the contrary, when this sum is near zero (blue in the map) this means that the kind of relationship between input parameters and model outputs

is complex and involves cross-effects among several parameters. We observe that the areas where interactions among Sobol' indices are important concentrates spatially around the injectors.

6 Conclusion

Dealing with uncertainty quantification may require executing from thousands to millions of runs of the same simulation, making it an extremely compute-intensive process that will fully benefit from Exascale machines. However, the large amount of data generated is a strong I/O bottleneck if the intermediate data is saved to disk. The proposed approach, implemented in the Melissa framework, demonstrates that combining one-pass statistics algorithms with an elastic and fault-tolerant in transit data processing architecture drastically reduces the amount of intermediate data stored, making it possible to compute ubiquitous statistics.

Melissa currently allows the in transit execution of large scale uncertainty quantification studies. This open-source software² currently implements the computation of the statistical tools necessary for this purpose: moment-based statistics (mean, standard deviation, higher orders), quantiles, Sobol' indices, and threshold exceedance. Future work will include the integration of Melissa with the OpenTurns uncertainty quantification software to consolidate and broaden their respective capabilities.

References

1. Archambeau, F., Méchitoua, N., Sakiz, M.: Code_saturne: a finite volume code for the computation of turbulent incompressible flows. *Int. J. Finite Vol.* **1** (2004)
2. Baudin, M., Boumhaout, K., Delage, T., Iooss, B., Martinez, J.M.: Numerical stability of sobol' indices estimation formula. In: Proceedings of the 8th International Conference on Sensitivity Analysis of Model Output (SAMO 2016). Le Tampon, Réunion Island, France (2016)
3. Baudin, M., Dutfoy, A., Iooss, B., Popelin, A.: Open TURNS: an industrial software for uncertainty quantification in simulation. In: Ghanem, R., Higdon, D., Owhadi, H. (eds.) Springer Handbook on Uncertainty Quantification, pp. 2001–2038. Springer (2017)
4. Bect, J., Ginsbourger, D., Li, L., Picheny, V., Vazquez, E.: Sequential design of computer experiments for the estimation of a probability of failure. *Stat. Comput.* **22**, 773–793 (2012)
5. Bennett, J., Grout, R., Pébay, P., Roe, D., Thompson, D.: Numerically stable, single-pass, parallel statistics algorithms. In: Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, pp. 1–8. IEEE (2009)
6. Bennett, J.C., Abbasi, H., Bremer, P.T., Grout, R., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., et al.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, pp. 1–9. IEEE (2012)
7. Bennett, J.C., Krishnamoorthy, V., Liu, S., Grout, R.W., Hawkes, E.R., Chen, J.H., Shepherd, J., Pascucci, V., Bremer, P.T.: Feature-based statistical analysis of combustion simulation data. *IEEE Trans. Vis. Comput. Graph.* **17**(12), 1822–1831 (2011)

² <https://melissa-sa.github.io/>.

8. Cannamela, C., Garnier, J., Iooss, B.: Controlled stratification for quantile estimation. *Ann. Appl. Stat.* **2**, 1554–1580 (2008)
9. Chan, T.F., Golub, G.H., LeVeque, R.J.: Updating formulae and a pairwise algorithm for computing sample variances. In: COMPSTAT 1982 5th Symposium held at Toulouse 1982, pp. 30–41. Springer (1982)
10. David, H., Nagaraja, H.: Order Statistics, 3rd edn. Wiley, New-York (2003)
11. de Rocquigny, E., Devictor, N., Tarantola, S. (eds.): Uncertainty in Industrial Practice. Wiley (2008)
12. Finch, T.: Incremental calculation of weighted mean and variance. Technical report. University of Cambridge (2009)
13. Fournier, Y., Bonelle, J., Moulinec, C., Shang, Z., Sunderland, A., Uribe, J.: Optimizing code_saturne computations on petascale systems. *Comput. Fluids* **45**(1), 103–108 (2011). 22nd International Conference on Parallel Computational Fluid Dynamics (ParCFD 2010)ParCFD
14. Gamboa, F., Janon, A., Klein, T., Lagnoux, A.: Sensitivity analysis for multidimensional and functional outputs. *Electron. J. Stat.* **8**(1), 575–603 (2014)
15. Gilquin, L., Arnaud, E., Prieur, C., Monod, H.: Recursive estimation procedure of sobol' indices based on replicated designs. Submitted, <http://hal.univ-grenoble-alpes.fr/hal-01291769> (2017)
16. Glynn, P.W.: Importance sampling for monte carlo estimation of quantiles. In: Proceedings of Second International Workshop on Mathematical Methods in Stochastic Simulation and Experimental Design, pp. 180–185. Publishing House of Saint Petersburg University (1996)
17. Hesterberg, T., Nelson, B.: Control variates for probability and quantile estimation. *Manag. Sci.* **44**, 1295–1312 (1998)
18. Higdon, D., Gattiker, J., Williams, B., Rightley, M.: Computer model calibration using high-dimensional output. *J. Am. Stat. Assoc.* **103**, 571–583 (2008)
19. Homma, T., Saltelli, A.: Importance measures in global sensitivity analysis of non linear models. *Reliab. Eng. Syst. Saf.* **52**, 1–17 (1996)
20. Iooss, B.: Estimation itérative en propagation d'incertitudes : réglage robuste de l'algorithme de Robbins-Monro. Preprint, <https://hal.archives-ouvertes.fr/hal-02511787> (2020)
21. Iooss, B., Lemaître, P.: A review on global sensitivity analysis methods. In: Meloni, C., Dellino, G. (eds.) Uncertainty management in Simulation-Optimization of Complex Systems: Algorithms and Applications, pp. 101–122. Springer (2015)
22. Iooss, B., Marrel, A.: Advanced methodology for uncertainty propagation in computer experiments with large number of inputs. *Nucl. Technol.* **205**, 1588–1606 (2019)
23. Janon, A., Klein, T., Lagnoux, A., Nodet, M., Prieur, C.: Asymptotic normality and efficiency of two sobol index estimators. *ESAIM: Probab. Stat.* **18**, 342–364 (2014)
24. Lakshminarasimhan, S., Jenkins, J., Arkatkar, I., Gong, Z., Kolla, H., Ku, S.H., Ethier, S., Chen, J., Chang, C.S., Klasky, S., et al.: Isabela-qa: query-driven analytics with isabela-compressed extreme-scale scientific data. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 31. ACM (2011)
25. Lampitella, P., Inzoli, F., Colombo, E.: Note on a formula for one-pass, parallel computations of arbitrary-order, weighted, multivariate central moments (2015)
26. Le Gratiet, L., Iooss, B., Browne, T., Blatman, G., Cordeiro, S., Goursaud, B.: Model assisted probability of detection curves: new statistical tools and progressive methodology. *J. Nondestruct. Eval.* **36**, 8 (2017)
27. Lemaire, M., Chateauneuf, A., Mitteau, J.C.: Structural Reliability. Wiley (2009)
28. Marrel, A., De Lozzo, M.: Sensitivity analysis with dependence and variance-based measures for spatio-temporal numerical simulators. In: Stochastic Environmental Research and Risk Assessment, In Press (2017)
29. Marrel, A., Iooss, B., Jullien, M., Laurent, B., Volkova, E.: Global sensitivity analysis for models with spatially dependent outputs. *Environmetrics* **22**, 383–397 (2011)
30. Marrel, A., Perot, N., Mottet, C.: Development of a surrogate model and sensitivity analysis for spatio-temporal numerical simulators. *Stoch. Environ. Res. Risk Assess.* **29**, 959–974 (2015)
31. Marrel, A., Saint-Geours, N.: Sensitivity analysis of spatial and/or temporal phenomena. In: Ghanem, R., Higdon, D., Owhadi, H. (eds.) Springer Handbook on Uncertainty Quantification. Springer (2017)

32. Morio, J., Balesdent, M.: Estimation of Rare Event Probabilities in Complex Aerospace and Other Systems. Woodhead Publishing (2016)
33. Nanty, S., Helbert, C., Marrel, A., Pérot, N., Prieur, C.: Uncertainty quantification for functional dependent random variables. *Comput. Stat.* (2016). <https://doi.org/10.1007/s00180-016-0676-0>
34. Pébay, P.: Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Sandia Report SAND2008-6212, Sandia National Laboratories **94** (2008)
35. Pébay, P., Thompson, D., Bennett, J., Mascarenhas, A.: Design and performance of a scalable, parallel statistics toolkit. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp. 1475–1484. IEEE (2011)
36. Popelin, A.L., Iooss, B.: Visualization tools for uncertainty and sensitivity analyses on thermal-hydraulic transients. In: Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013). Paris, France (2013)
37. Prieur, C., Tarantola, S.: Variance-based sensitivity analysis: theory and estimation algorithms. In: Ghanem, R., Higdon, D., Owhadi, H. (eds.) Springer Handbook on Uncertainty Quantification. Springer (2017)
38. Ribés, A., Pouderoux, J., Popelin, A.L., Iooss, B.: Visualizing statistical analysis of curves datasets in Paraview. In: 2014 IEEE Conference on Visual Analytics Science and Technology (VAST). Paris, France (2014)
39. Ribés, A., Terraz, T., Fournier, Y., Iooss, B., Raffin, B.: Large scale in transit computation of quantiles for ensemble runs. Unpublished Technical Report (2019). [arXiv: 1905.04180](https://arxiv.org/abs/1905.04180)
40. Ribés, A., Pouderoux, J., Iooss, B.: A visual sensitivity analysis for parameter-augmented ensembles of curves. *J. Verif. Valid. Uncertain. Quantif.* December 2019 **4**(4), 041007 (2020, February 11). <https://doi.org/10.1111/1.4046020>
41. Robbins, H., Monro, S.: A stochastic approximation method. *Ann. Math. Stat.* **22**, 400–407 (1951)
42. Saltelli, A., Chan, K., Scott, E. (eds.): Sensitivity Analysis. Wiley Series in Probability and Statistics. Wiley (2000)
43. Smith, R.: Uncertainty quantification. SIAM (2014)
44. Sobol, I.: Sensitivity estimates for non linear mathematical models. *Math. Model. Comput. Exp.* **1**, 407–414 (1993)
45. Terraz, T., Ribés, A., Fournier, Y., Iooss, B., Raffin, B.: Melissa: large scale in transit sensitivity analysis avoiding intermediate files. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17). Denver (2017)
46. Tukey, J.W.: Exploratory Data Analysis, vol. 2. Reading, Mass (1977)
47. Welford, B.: Note on a method for calculating corrected sums of squares and products. *Technometrics* **4**(3), 419–420 (1962)

Decaf: Decoupled Dataflows for In Situ Workflows



Orcun Yildiz, Matthieu Dreher, and Tom Peterka

Abstract In situ workflows manage the coordination and communication in a directed graph of heterogeneous tasks executing simultaneously in an high-performance computing system. The communication through the graph can be modeled as a dataflow, and Decaf is a software library for managing the dataflow for in situ workflows. Decaf includes a Python API to define a workflow, creating a complete stand-alone system, but the dataflow design also allows Decaf to support the communication needs of other workflow management systems, because a science campaign may be composed of several workflow tools. Decaf creates efficient parallel communication channels over MPI, including arbitrary data transformations ranging from simple data forwarding to complex data redistribution. Decaf provides three building blocks: (i) a lightweight data model that enables users to define the policies needed to preserve semantic integrity during data redistribution, (ii) flow control designed to prevent overflows in the communication channels between tasks, and (iii) a data contract mechanism that allows users to specify the required data in the parallel communication of the workflow tasks. Decaf has been used in a variety of applications. Two examples are highlighted. The first case is from materials science, where the science campaign consists of several workflow tools that cooperate, and Decaf supports these tools as the dataflow layer. The second problem is motivated by computational cosmology, where the in situ workflow consists of three parallel tasks: synthetic particle generation, Voronoi tessellation, and density estimation.

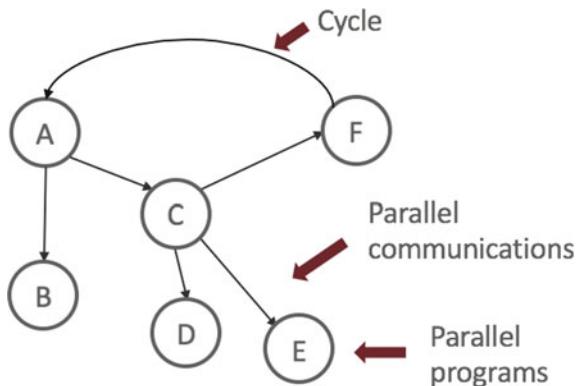
O. Yildiz (✉) · T. Peterka

Argonne National Laboratory, 9700S. Cass Ave., Lemont, IL 60439, USA
e-mail: oyildiz@anl.gov

T. Peterka
e-mail: tpeterka@mcs.anl.gov

M. Dreher
Formerly of Argonne National Laboratory, Lemont, USA

Fig. 1 Workflow graph, where the nodes are tasks and the edges represent information exchanged between the tasks. Tasks are parallel programs such as simulation, analysis, and visualization



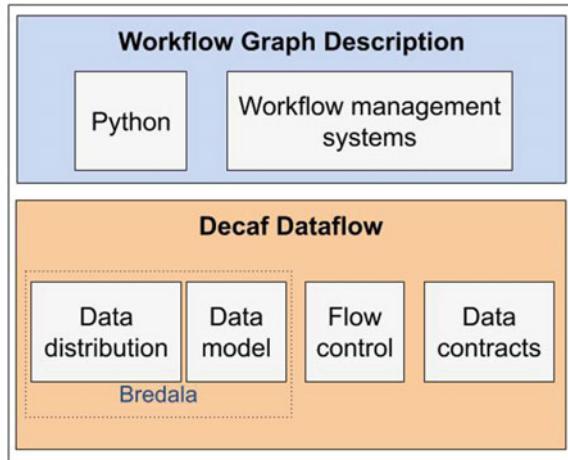
1 Introduction

Computational science often involves a workflow of interconnected tasks, several of them being executed on a supercomputer, for example, simulation, analysis, visualization, and user interaction. The workflow can be modeled as a directed graph, which is illustrated in Fig. 1, where the nodes of the graph are tasks and the edges represent information exchanged between the tasks. The graph can have cycles where feedback loops exist between tasks so that the result of a task can be used to modify another one (e.g., computational steering).

Dataflow is defined as the information exchanged between tasks in a workflow. Traditionally, tasks exchanged data through files. However, the growing mismatch between high-performance computing (HPC) rate and I/O bandwidth motivates shifting from file-oriented workflow models to *in situ workflows*, where dataflows are through memory or the supercomputer interconnect, avoiding the storage I/O bottleneck.

Decaf is a dataflow library for *in situ workflows*. Figure 2 shows an overview of Decaf and its components. Decaf is designed as a library separate from the workflow engine because a science campaign may consist of several workflow tools that need to cooperate, and this design allows one dataflow library to support multiple workflow tools simultaneously. However, several challenges need to be addressed as a result of this design choice. The dataflow needs to manage the data exchange between heterogeneous programming and data models because simulation and analysis tasks are often developed independently. Workflows can be instantiated with different numbers of resources (e.g., MPI ranks) per task, and the dataflow has to transform and redistribute data during the exchange between tasks. The dataflow runtime needs to support any generic directed graph because workflow graphs come in different shapes and sizes, ranging from a simple pipeline of two or three tasks to more complex topologies including fan-in, fan-out, and cycles. To meet these challenges, Decaf is equipped with the following features:

Fig. 2 Overview of Decaf dataflow library and its components



- A decoupled dataflow, a hybrid of tight and loose coupling of tasks through an optional intermediate set of resources that can be used to transform or redistribute data between tasks
- A flow control library designed to manage disparate data rates between tasks
- A data contract mechanism that allows users to specify the required data in the parallel communication of the tasks
- A high-level Python API for the workflow graph definition so that Decaf can be used as a stand-alone workflow system when it is not used in conjunction with other workflow tools
- A design that allows composition of different workflow systems so that the workflow can be defined in a completely different software tool with Decaf providing only the dataflow capability
- A message-driven runtime for executing workflow tasks when all messages for a task have been received, which supports any directed graph topology, including cycles
- Support for the MPMD (multiple program multiple data) capability of MPI to run multiple executables in the same job launch, a feature available on most supercomputing and cluster platforms

The remainder of this chapter is organized as follows. Section 2 gives background for in situ workflows and presents related work. Section 3 describes Decaf's design and key features. Section 4 illustrates the capabilities of Decaf. Section 5 gives a summary and briefly describes future work.

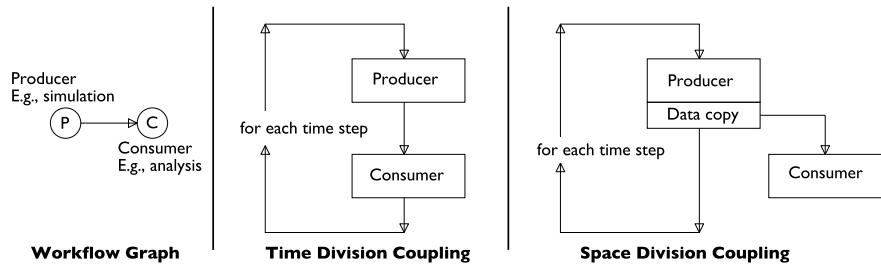


Fig. 3 Time division and space division in situ coupling of a producer and consumer task [17]. In time division coupling, these tasks execute sequentially on the same resource; in space division coupling, they execute concurrently on separate resources

2 Background and Related Work

This section first provides a brief background on types of in situ workflows with respect to the coupling mode between the workflow tasks. Then, it presents related work by categorizing the workflows according to their coupling modes.

2.1 Types of In Situ Workflows

Figure 3 illustrates the different in situ coupling modes when two tasks are connected in the workflow graph. In time division, the two tasks, producer and consumer, share the same resources and run sequentially in time. For example, a simulation (producer) and analysis (consumer) task may take turns operating on the same time step of data. The simulation computes the data for one time step and then waits while the analysis task processes the data, which then waits while the simulation generates the next step.

In space division, the producer and consumer run on different resources concurrently in time. Using the same simulation-analysis example, the simulation computes the first time step, copies or sends it to the consumer, and then computes the second time step while the analysis code processes the first time step. Both modes are used in practice, and there are time-space tradeoffs between them.

2.2 In Situ Workflow Runtimes

In situ tools have been developed by various communities with performance, programmability, and portability tradeoffs highlighted by Dorier et al. [13]. In this section, these tools are characterized according to the type of coupling between their tasks.

2.2.1 Time Division

Scientific visualization is one of the main targets for in situ analytics, motivated by the need for high bandwidth and temporal resolution. Yu et al. [35] integrated a particle volume renderer synchronously in a combustion code (S3D). The renderer processing required 10% of the total execution time to produce images. Current production visualization tools include ParaView [3] and VisIt [2] for postprocessing. Both have an in situ library, respectively Catalyst [21] and Libsim [34], to process simulation data. The goal of these libraries is to convert the internal data structures of the simulation to VTK data structures. The libraries then execute the rendering pipeline, usually synchronously, although other configurations are possible.

2.2.2 Space Division

Other in situ tools originate in the I/O community. Originally designed for I/O staging, these tools have subsequently coupled analysis and visualization applications together with simulations. ADIOS [26] is a common interface for multiple I/O methods. Initially developed to store data efficiently in parallel, it now provides methods to share data between codes and transform data in situ along the I/O path [6].

Several frameworks using the ADIOS interface can transfer data asynchronously to dedicated resources. DataStager [1] can schedule data movement while the simulation is in its computational phase, avoiding unnecessary network contention. FlexPath [8] provides a publisher/subscriber model to exchange data between parallel codes having different numbers of processes ($M \times N$ communication pattern).

DataSpaces [9] implements a distributed shared-memory space with a publisher/subscriber interface for external applications. DataSpaces indexes data based on a space-filling curve. Data are then distributed among data servers based on their index. The index is used both for pushing data into DataSpaces and for retrieving data efficiently from it.

Damaris [11, 14] splits the global MPI communicator of a simulation to dedicated cores or nodes [12] in order to run the analysis concurrently with the application. The framework was used to stage I/O and for in situ visualizations. Functional partitioning [25] also uses dedicated cores for I/O with a FUSE interface.

HDF5 DSM [5] connects a simulation code to ParaView by using the HDF5 interface. The simulation and the ParaView servers can run in separate jobs. Both jobs read and write data to the shared virtual HDF5 file layer; steering is also possible.

2.2.3 Hybrid Approaches

Time division and space division each have pros and cons, motivating flexibility in analytics placement [36]. Several solutions now support both time and space division. FlexIO [37] allows time division tasks on the same compute cores or space division on dedicated cores. Bennett et al. [4] combined computations in time division followed

by other processing on dedicated nodes using DART servers [10] to transfer data, applying several analysis algorithms to the combustion code S3D.

FlowVR [19] couples parallel codes to form a graph of tasks. Each task is usually a separate executable. Communication between parallel codes is managed by a daemon running on each node. Decaf also adopts a hybrid approach. Similar to FlowVR, Decaf composes multiple executables to form a workflow. However, its execution model is designed for in situ HPC environments, and it does not rely on a sepeate daemon to manage the graph execution.

3 Design

This section provides an overview of Decaf and its main components. Decaf's main building block is a dataflow, which consists of a producer, a consumer, and a communication object. Decaf implementation is defined in two parts. First, the user defines the dataflow using Decaf's Python scripting interface, which produces a JSON file describing the layout. Second, the user implements the producer and consumer of that dataflow using Decaf's C/C++ API. Besides the dataflow component, Decaf provides three building blocks: (i) a data model to preserve semantic integrity during data redistribution, (ii) a flow control to manage disparate rates between tasks, and (iii) a data contract mechanism to specify the required data in the parallel communication of the tasks.

3.1 Decaf Dataflow

A dataflow is the association of a producer, a consumer, and a communication object to exchange data between the producer and consumer. Producers and consumers, called nodes in Decaf, are parallel codes such as simulation and analytics. For instance, Gromacs [33], a molecular dynamics simulation, typically generates atom positions, and scientists use tools such as VMD [24] to visualize and analyze the atom positions. In an in situ context, Gromacs and VMD communicate directly through memory. Data structures between the two codes are often different, and some data transformation might be required. This can be as simple as converting data units (from nanometers to angstroms) to more complex data rearrangements such as mapping a simulation data structure to a VMD data structure.

These data transformations can require extensive computations. To address these needs, Decaf dataflow includes an intermediate staging area with computational resources. This staging area is called a *link*, which is an intermediate parallel program transforming data between a producer and a consumer.

Figure 4 shows the organization of a simple dataflow from producer to link to consumer. Advanced workflow graphs can be obtained by combining multiple such dataflows. The producer, link, and consumer are all MPI programs, each with its own

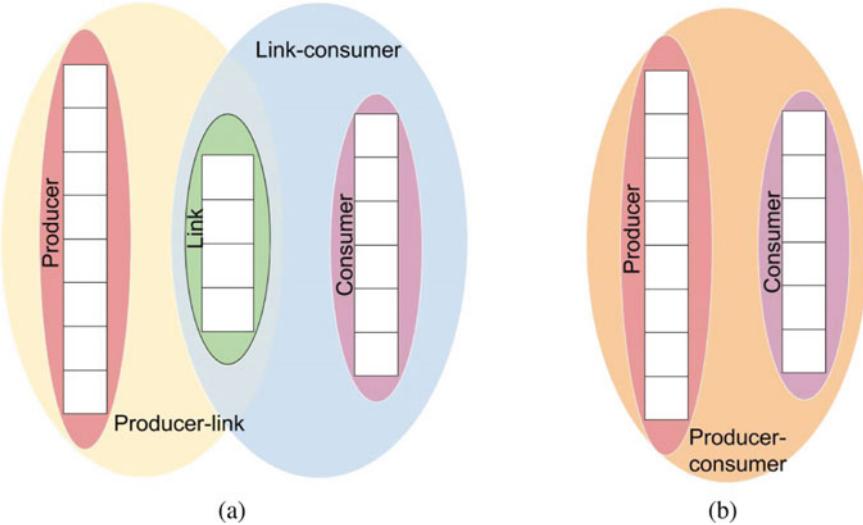


Fig. 4 **a** Decaf forms 5 communicators for a dataflow: one communicator each for producer, consumer, and link, plus two more communicators for the overlap between producer-link and link-consumer. **b** The link is optional; when not used, there are only 3 communicators

MPI communicator. The dataflow creates two additional communicators: producer-link and link-consumer. Section 3.2 explains how to describe such graphs and how the runtime executes them. Section 3.4 describes how data are exchanged through the communicators.

The link resources are optional; one can disable the link when no data manipulations are required between the producer and consumer. In that case, the user does not assign any resources to the link, and the runtime creates only one communicator directly between the producer and the consumer.

3.2 Workflow Graph Description and Runtime Execution

A Decaf workflow is the composition of multiple dataflows. For instance, to create a pipeline of three tasks, the user declares two dataflows, where the second task is both consumer in the first dataflow and producer in the second dataflow. Decaf does not apply any constraints on the graph topology. In particular, the user can define cycles in the graph for computational steering.

```
w = nx.DiGraph()

# Task declaration
w.add_node('node0', start_proc=0, nprocs=4)
w.add_node('node1', start_proc=7, nprocs=2)
w.add_node('node2', start_proc=11, nprocs=1)

# Dataflow declaration
w.add_edge('node0', 'node1', start_proc=4, nprocs=3,
           func='dflow', path=mod_path,
           prod_dflow_redist='count',
           dfollow_con_redist='count')
w.add_edge('node1', 'node2', start_proc=9, nprocs=2,
           func='dflow', path=mod_path,
           prod_dflow_redist='count',
           dfollow_con_redist='count')

wf.workflowToJson(w, mod_path, "linear3.json")
```

Listing 1 Code sample to generate a 3-node pipeline.

The user describes the workflow graph in a Python script. Listing 1 presents the code generating a three-task pipeline. First, the user describes individual tasks with a name and a set of resources (MPI ranks). Second, the user describes the dataflow linking two tasks (producer, consumer), a set of resources, and the choice of the redistribution strategy. The same task can be used by several dataflows as producer or consumer. Once all the tasks are described, the user calls `workflowToJson` to generate a JSON file. That file contains an intermediate representation of the workflow read by the Decaf runtime upon initialization. The graph description is done prior to the launch of the workflow.

In a Decaf workflow, every task is an MPI program. Each task can be a separate program, or all tasks may be combined in a single program. Decaf relies on the MPMD capability of MPI to launch all the programs. With this method, `MPI_COMM_WORLD` is shared among all the executables. Based on the information provided by the JSON file, Decaf creates either three or five communicators per dataflow (Fig. 4). This method requires existing codes such as simulations only to replace their equivalent of `MPI_COMM_WORLD` by the communicator provided by Decaf.

3.3 Structure of Task Code

To minimize the required code modifications for integration of user codes, Decaf provides a simple put/get API that allows tasks to send/receive data to/from the rest of the workflow. Listing 2 presents a typical program.

After the initialization of the runtime (Sect. 3.2), the code loops over incoming data. The `get(in_data)` call waits until data are received on all inbound links or a termination message is received, whereby the code exits the loop. Incoming data are processed and sent by calling `put(out_data)`. The `terminate()` call signals to the runtime to exit the application. This API is available in both C and C++.

```

int main(int argc,
        char** argv)
{
    // define the workflow
    Workflow workflow;
    make_wflow_from_json(workflow, "linear3.json");

    // initialize MPI and Decaf
    MPI_Init(NULL, NULL);
    Decaf* decaf = new Decaf(MPI_COMM_WORLD, workflow);

    // run the task
    vector<pConstructData> in_data;
    while (decaf->get(in_data))
    {
        // process the input data
        // send the results outbound
        pConstructData out_data;
        decaf->put(out_data);
    }
    // end the task
    decaf->terminate();

    // cleanup
    delete decaf;
    MPI_Finalize();
}

return 0;
}

```

Listing 2 A typical program constructs the Decaf object and executes a node task. The task waits for all its inputs to be satisfied and then processes the received data. It can safely do so in an infinite loop because the Decaf `get()` function returns false upon termination.

3.4 Data Model and Data Redistribution in the Dataflow

The key question is how to maintain a valid data model that preserves the original semantics when data are redistributed between M producer and N consumer processes. One major challenge is that none of these processes has a complete view of the data decomposition; hence, considerable effort is required to match decomposition between producer and consumer. Also, different scenarios require different logic, and extra hints from the user can simplify the required computation. For example, if only one producer rank has a particular dataset and only one consumer rank wants to read it, then the underlying dataflow must reconcile this information. With this goal in mind, Bredala [16] is a library designed to build a data model with enough information to preserve the semantics of data during data redistribution in situ dataflows.

Decaf uses both the data model and the redistribution components of Bredala. The data model serves as the data interface to exchange data between tasks within the workflow. For each dataflow, Decaf creates two redistribution components: one between the producer and the link and one between the link and the consumer. When calling `put()`, Decaf passes the data to the corresponding redistribution component that manipulates and transmits the data to its destination. When calling `get()`, Decaf receives data on the corresponding redistribution component and transmits it to the task. These data model and redistribution components are detailed in the following subsections.

```

int gid;
float pos[3*NB_POS];

// Declaring the fields
SimpleFieldi fieldGid(gid);
// One semantic item consists of 3 floats (e.g., position in a 3D space)
ArrayFieldf fieldPos(pos, 3*NB_POS, 3);

pConstructData container;
container->appendData("gid", fieldGid,
                      DECAF_NOFLAG, DECAF_SHARED,
                      DECAF_SPLIT_DEFAULT,
                      DECAF.Merge_Keep_First);
container->appendData("pos", fieldPos,
                      DECAF_POS, DECAF_PRIVATE,
                      DECAF_SPLIT_DEFAULT,
                      DECAF.Merge_Append);

```

Listing 3 Code sample to generate a simple data model. The data model is composed of a global identifier and an array of positions.

3.4.1 Data Model Description

Bredala protects the semantic integrity of a data model during split and merge operations using the notion of a semantic item. This is the smallest subset of data that contains all the fields of the original data structure while preserving its semantics. In Bredala, the split and merge functions are embedded within the data model. These functions divide a data model into subdata models and merge subdata models into a larger data model safely.

Listing 3 gives an example to describe a simple data model with a global identifier and an array of positions. First, each field is declared with its type (e.g., Simple/Array) and its semantic information if any. For instance, the last argument in the *fieldPos* declaration indicates that this position is composed of three floating points. This semantic information will be used by Bredala for preserving the semantics of this field during data redistribution. Then, each field is pushed in a container (*pConstructData*). Besides the name and the field variables, the user provides a semantic flag, a visibility flag, and flags for split and merge policies.

The semantic flag indicates whether a field has extra semantic meaning in the data model. For instance, the flag *DECAF_POS* indicates that the field *pos* represents the positions of the semantic items. Then, this information can be used to redistribute data spatially. The visibility flag indicates whether the value of the field is shared by all the items in the data model (e.g., a global identifier with *DECAF_SHARED* flag) or when this value is different for each group (e.g., different position values with *DECAF_PRIVATE* flag). The split and merge flags tell Bredala which method to use during the split and merge operations. We will see an example of this in the next section.

3.4.2 Data Redistribution

Bredala provides redistribution components to transfer a data model from M to N processes. The components use the semantic information provided by the data model to extract individual semantic items and send them to the appropriate destination. Bredala provides a safe way to access, extract, and merge semantic items within a data model using three redistribution strategies: round-robin, contiguous, and block redistribution (Fig. 5). The round-robin strategy performs the data redistribution item by item in a round-robin fashion. The contiguous strategy redistributes all the items while preserving their order. The block redistribution strategy divides a global domain into subdomains and attributes each item to a particular subdomain.

In each redistribution strategy, data redistribution consists of three main steps: (1) splitting the data model on the producer side; (2) transferring these data models between M producer and N consumer processes over MPI; and (3) merging the received data models on N consumer processes. Splitting and merging are performed based on the user-defined policies in the data model. For instance, the DECAF_SPLIT_DEFAULT policy in Listing 3 splits the array of positions in chunks of three floating-point values based on the semantics of the field *pos*. The policy MERGE_APPEND_VALUES concatenates the two arrays of position values.

3.5 Flow Control Library

In space division mode, workflow tasks run independently on dedicated resources and exchange data asynchronously. However, tasks may run at different rates. For instance, a consumer might not be able to process the incoming data from a producer fast enough, overflowing or stalling the dataflow between producer and consumer. To mitigate such scenarios, Decaf provides a flow control library, Manala [20], to control the flow of messages between producers and consumers.

Figure 6 illustrates the operation of Manala, which manages the message queue between producers and consumers. First, Manala intercepts the messages from producers and stores them in a storage hierarchy consisting of main memory, shared remote memory (i.e., distributed in-memory key/value store), and the file system. The hierarchical design allows Manala to buffer as many messages as possible before blocking producers. Then, Manala selects the messages that need to be forwarded to the consumers depending on the flow control policy defined by the user. Manala provides four different flow control policies: *all*, *some*, *latest*, and *hybrid*. The *all* policy applies FIFO flow control, Decaf’s default policy when Manala is not used, forwarding all messages to consumers in the order they have been sent by producers. The *some* policy also preserves the order of the messages, but this time Manala stores one out of every x messages received and drops the rest of them. In the *latest* policy, Manala forwards the most recent data to the consumers and drops the older data. The *hybrid* policy combines the *some* and *latest* policies by forwarding one out of every x messages received among the most recent data. These different policies

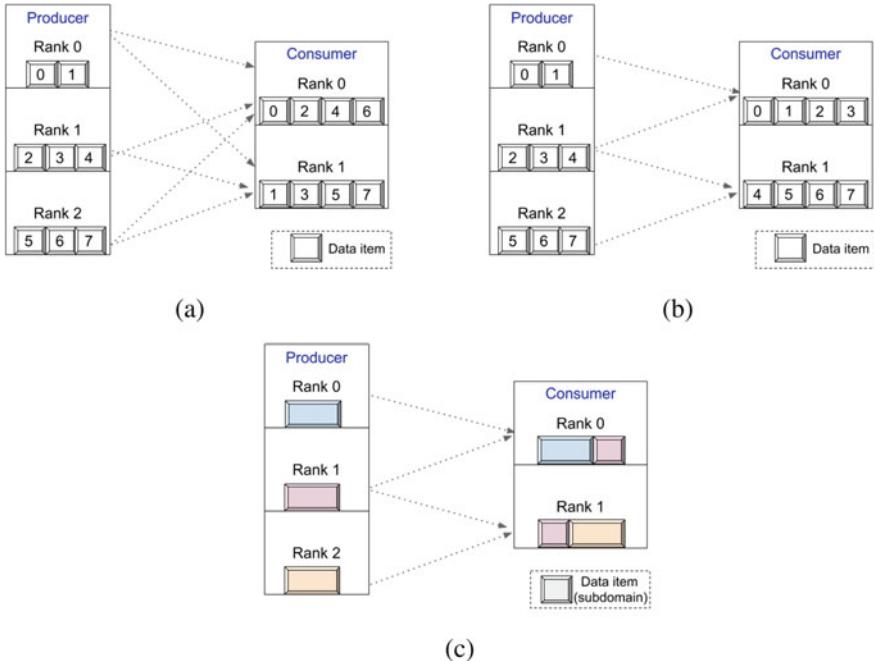


Fig. 5 Redistribution strategies available in Bredala. **a** The round-robin strategy performs the data redistribution item by item in a round-robin fashion. **b** The contiguous strategy redistributes all the items while preserving their order. **c** The block redistribution strategy divides a global domain into subdomains and attributes each item to a particular subdomain

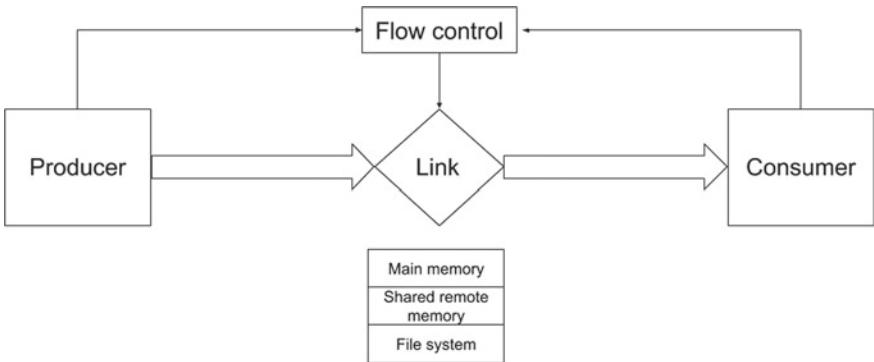


Fig. 6 Manala, a flexible flow control library, controls the flow of messages between the producer and the consumer by using a storage hierarchy consisting of main memory, shared remote memory, and the file system

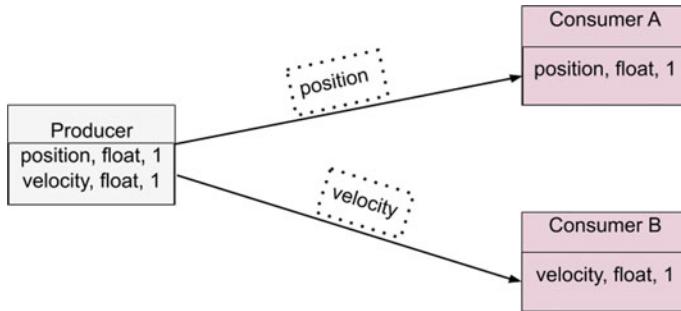


Fig. 7 Sample data contract mechanism between one producer and two different consumers. The dashed-line rectangles represent the data fields forwarded in each dataflow

provide flexibility in managing the flow of messages in the workflow in order to meet application requirements.

3.6 Data Contract Mechanism

Decaf decouples data management between producers and consumers by handling the exchange of messages between these tasks. Traditionally, the data to be sent to different consumers are usually hard-coded in producers. This requires modifying the producer code every time the consumer changes, which is inefficient. To address this problem, Decaf provides a mechanism [27] where users can specify a data contract between a producer and consumer of data. This data contract mechanism allows the Decaf runtime to automatically extract the data from producers depending on the requirements of the consumers, minimizing required code changes. Moreover, the same producer can be coupled to different consumers with different data requirements, without specializing the producer task for each consumer. Without data contracts, a single producer would have to send the superset of data required by all the consumers. Doing so, however, would create redundant data transfers over the network, wasting resources.

In the contract, the data model is represented as a set of fields. The field is a triplet, $(name, type, periodicity)$, where periodicity is the frequency at which the producer generates the field. The user specifies the set of fields that the producer makes available for output, as well as the data fields needed for input to the consumer. Then, Decaf's data contract mechanism generates the matching list to deduce which data fields need to be forwarded in the dataflow between the producer and the consumer. Figure 7 illustrates a simple data contract example between one producer and two different consumers. Listing 4 shows the necessary additions to describe such a contract in the workflow graph description.

```

# Contract declaration in workflow tasks
...
producer.addOutput('position', 'float', 1)
producer.addOutput('velocity', 'float', 1)
...
consumerA.addInput('position', 'float', 1)
consumerB.addInput('velocity', 'float', 1)

# Contract declaration in dataflows
...
edgeA.addInput('position', 'float', 1)
edgeA.addOutput('position', 'float', 1)
edgeA.setForwardFields(True)
...
edgeB.addInput('velocity', 'float', 1)
edgeB.addOutput('velocity', 'float', 1)
edgeB.setForwardFields(True)

```

Listing 4 Required modifications in the Python API for the declaration of data contracts.

4 Science Drivers

Two science use cases are highlighted below: molecular dynamics and cosmology. In the molecular dynamics example, the steering capabilities of Decaf and its interoperability with other *in situ* libraries to create heterogeneous workflows are illustrated. In the cosmological example, the scalability of Decaf is tested, and its capacity to handle large, complex data structures is demonstrated.

4.1 Molecular Dynamics

The first example is steering a molecular dynamics simulation to trigger a biological mechanism. In the FepA protein, a channel in the periplasm of Gram-negative bacteria provides an entryway where compounds can traverse the membrane of the cell. This mechanism is of interest to biologists because drugs pass through these channels, and steering the simulation allows a biologist to push a complex within a channel and accelerate the traversal process.

Workflows supporting computational steering pose several challenges. First, a steering workflow needs to support cycles in the workflow graph, which can potentially generate deadlocks. Second, the steering process must be performed asynchronously to avoid blocking the simulation. Third, because multiple types of interactions may be needed (force feedback, visualization, density computation), Decaf must be able to integrate different tools from different communities into a single workflow.

In this experiment, a steering workflow is implemented with Decaf to guide the iron complex toward the FepA channel, based on the implementation proposed by Dreher et al. [18]. Figure 8 summarizes the workflow. First, Decaf API calls are integrated into Gromacs [33], a molecular dynamics simulation code, to add external forces and expose atom positions to the rest of the workflow. Second, in a link (LinkMorton), a Morton index [30] is computed for each atom particle. Third, a

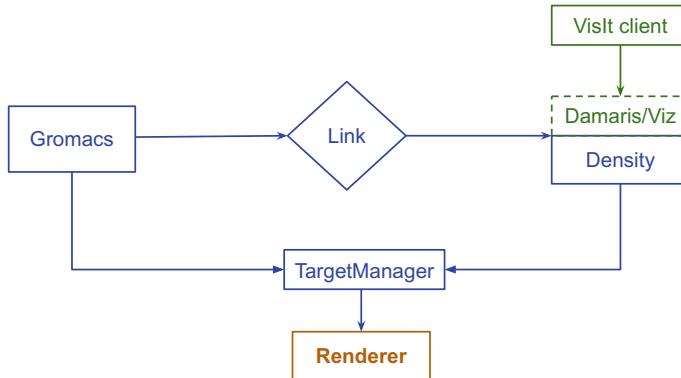


Fig. 8 Steering workflow with Decaf (blue), FlowVR (brown), and Damaris/Viz connected to the VisIt client (green). The steering part is managed by Decaf while visualizations are handled by Damaris and FlowVR. Decaf and FlowVR tasks and communications are in space division mode (blue arrows) while Damaris execution is executed in time division mode (green arrow) during the execution of Density

3D density grid (Density) is computed. Fourth, the force (TargetManager) to guide the system is computed. The trajectory for the complex to follow is defined by a list of target positions, provided by the user, forming a path. A path-finding algorithm [23] based on the previously computed 3D density grid guides the iron complex from one target to the next.

At each step of the workflow, the fields of the data model are modified. First, the atom positions and their indices from the simulation are sent to the link. Second, the Morton indices are added, and data are reorganized to compute the density grid. Third, the Morton indices are removed from the data model (no longer necessary), and a 3D grid is appended to the data model. Then, the result is sent to TargetManager, which broadcasts forces to the simulation.

For this application, FlowVR and Damaris/Viz [14] are used to perform in situ visualization. First, the workflow is connected to the renderer of Dreher et al. [18] based on Hyperballs [7] to visualize the molecular model and the state of the steering system (Fig. 9a). This visualization is preferred by biologists to guide the complex because the user can navigate within the molecular structure and track the iron complex. This visualization is performed in space division mode. Second, Damaris/Viz is used in time division mode to visualize the density grid with VisIt (Fig. 9b). The visualization of the density grid shows the biologists where low-density areas are located, which are good candidates for the iron complex to go through. Since no tools support both visualizations, several visualization packages had to be combined in the same workflow. Without Decaf, the biologists would normally perform such visualizations separately, resulting in redundant data transfers among these tools and loss of productivity.

The following tests are conducted on Froggy, a 138-node cluster from the Ciment infrastructure. Each compute node is equipped with 2 eight-core processors, Sandy

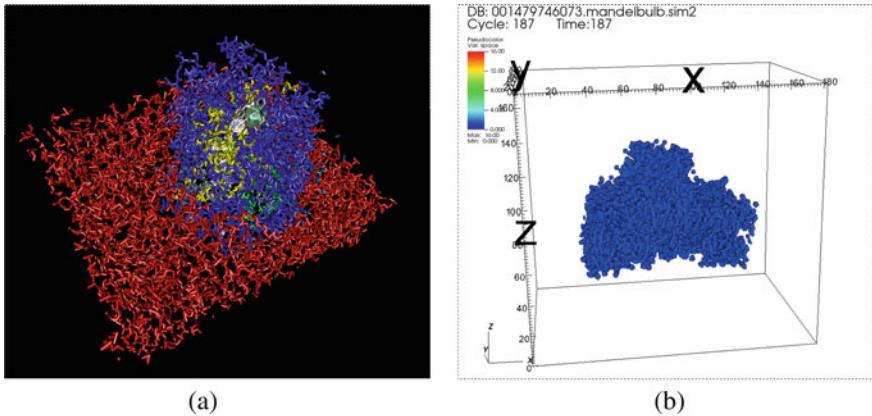


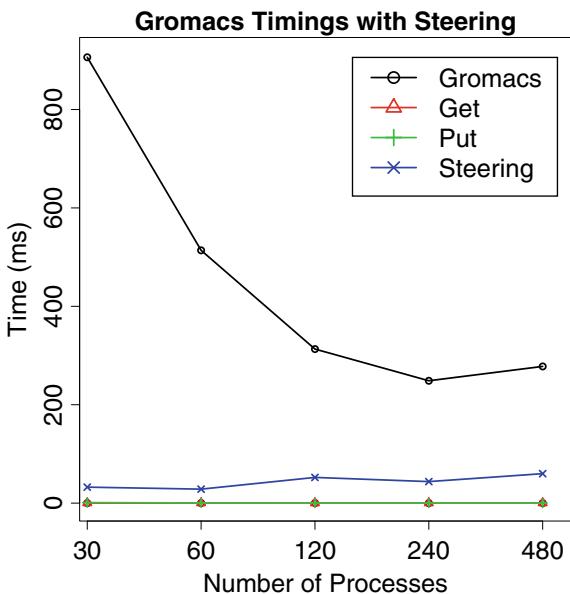
Fig. 9 **a** Visualization of the molecular system with FlowVR performed in space division mode, as depicted in Fig. 8. **b** Visualization of the density grid with Damaris/Viz using VisIt in time division mode

Bridge-EP E5-2670 at 2.6GHz with 64 GB of memory. Nodes are interconnected through an FDR InfiniBand network. FlowVR 2.1 and Gromacs 4.6 are compiled with Intel MPI 4.1.0. For all experiments, the molecular model of the FepA is used, which is composed of about 70,000 atoms. For all scenarios, the atom positions are extracted from the simulation every 100 iterations.

An experiment was conducted to study the impact of the steering pipeline on Gromacs performance. The workflow is created as described in Fig. 8. Each task runs in space division mode, where one node behaves as a staging node, and the remaining nodes are simulation nodes. The Density (four cores) and TargetManager (one core) tasks are hosted on one node. The Density task is limited to four cores since the density grid is relatively small ($170 \times 80 \times 180$). Gromacs runs on each simulation node using 15 cores per node. The LinkMorton task runs on the remaining core on each simulation node to preserve the data locality. MPI is configured to bind each MPI rank to a given core in order to avoid process migration and reduce performance variability.

Figure 10 presents the timing of the Gromacs execution and the steering process for 100 simulation iterations. Steering indicates the accumulated execution time of the LinkMorton, Density, and TargetManager tasks. The steering process has a time budget of 100 simulation iterations to complete before blocking the simulation. At 480 cores, the simulation requires on average 277 ms to compute 100 iterations. In all cases, the time spent by Gromacs in Decaf operations represents less than 0.1% of the allocated time budget.

Fig. 10 Time decomposition of Gromacs performance modified with Decaf and connected to the steering pipeline without visualization. The measured time is the average over a full I/O cycle (100 iterations)



4.2 Cosmology

The second use case is the analysis of dark matter tracer particles from a Vlasov-Poisson N-body cosmology code. This workflow converts particle positions to the deposition of particle density onto a regular 2D and 3D grid, using a Voronoi tessellation as an intermediate step. For high dynamic range data such as dark matter particles, computing the Voronoi tessellation first produces more accurate density estimates than less expensive methods that compute the density directly from the particle data [31]. The data model produced by the cosmology code, a set of particles, is transformed into the intermediate data model of the Voronoi tessellation, an unstructured polyhedral mesh, before being converted into a 2D or 3D regular grid of density scalar values.

The N-body cosmology code is HACC [22]. The tessellation and density estimator codes are built on the Tess library [32], which in turn is built on the DIY data-parallel programming model [28]. The three tasks are coupled with a link between each task in a 3-node linear workflow graph using Decaf. Decaf enables exchange of complex data structures among these tasks, transparent to such task codes. Normally, this would be done manually by the user by performing the data transformation for each particular data type in the task code. For instance, the link between the simulation and tessellation nodes rearranges the particles from the structure of arrays format produced by HACC to an array of structures format required by Tess.

All three tasks are parallel MPI programs that scale to large numbers of MPI processes. The intermediate tessellation has a large memory footprint, approximately

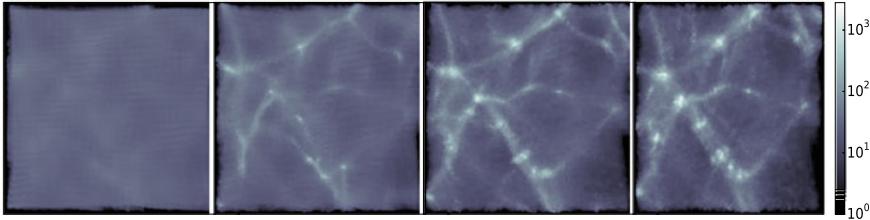


Fig. 11 (Left to right) Output of cosmology analysis workflow at time steps 10, 30, 60, and 100

15 times as large as the simulation data, making it necessary to compute the tessellation on a separate set of compute nodes from the simulation or density estimation. The density image (Fig. 11) is a fraction of the simulation data size; but because of the large memory footprints of the analysis tasks and the desire to perform analysis simultaneously with simulation, all the nodes and links are configured in space division mode to use separate resources. In the following experiment, all tasks and links have the same number of processes.

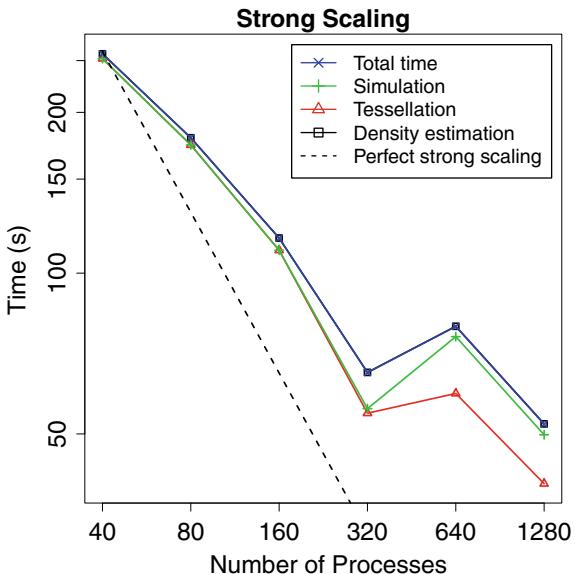
The following tests were conducted on the IBM Blue Gene/Q Vesta machine at the Argonne Leadership Computing Facility at Argonne National Laboratory. Vesta is a testing and development platform consisting of 2K nodes, each node with 16 cores (PowerPC A2 1.6GHz), 16 GB of RAM, and 64 hardware threads. Eight MPI processes per compute node are used. The Clang compiler, based on LLVM version 3.9, is used to compile the code with -O3 optimization.

Figure 12 shows the performance of a strong-scaling test in log-log scale. In this test, there are 128^3 particles estimated onto a 512^2 2D output grid. The vertical axis is the time to simulate a total of 100 time steps and to compute the tessellation and density estimation every 10 time steps. The horizontal axis is the total number of MPI processes for the entire workflow. The curves in the plot include total (end-to-end) time as well as the time for each of the three tasks.

The overall strong-scaling efficiency is approximately 50% from 40 to 320 processes. After 320 processes, the diminishing number of particles per process combined with the increasing imbalance between processes reduces the scalability of the tessellation, which in turn affects the rest of the workflow. For example, at 1,280 processes, the final time step of the simulation produces only 16 particles in the least-loaded process, and 1,972 particles in the most-loaded one. The load imbalance as dark matter particles cluster into halos is expected; using a k-d tree instead of a regular grid of blocks to perform the tessellation [29] can mitigate this situation.

The tasks overlap in time (space division) so that the total time is only slightly longer than that of the longest task, which is density estimation, but much less than the sum of the parts. This is especially true over the total of multiple time steps, effectively hiding the analysis time from the simulation.

Fig. 12 Strong scaling of the cosmology simulation-tessellation-density estimation workflow. Good scaling efficiency is shown until the small number of particles per process and load imbalance of the tessellation reduce efficiency. The time of the analysis tasks is effectively overlapped with the simulation time



5 Conclusion

This chapter introduces Decaf, middleware for coupling in situ tasks to form workflows. Decaf makes it easy to modify existing simulation or analytics codes with its simple API. It is also simple to describe the workflow in Python as a directed graph where nodes are tasks and edges are communication channels. Decaf does not impose any constraints on the graph topology and can manage graphs with cycles. The Decaf runtime handles building the communication channels and the data exchanges. Decaf protects the semantic integrity of data during parallel redistribution with its lightweight data model. Decaf's data contract mechanism allows the Decaf runtime to automatically extract the data from producers depending on the requirements of consumers. Decaf is also equipped with a flow control library to manage disparate rates between tasks.

Decaf's capabilities are demonstrated with science use cases. The first is a steering scenario merging a molecular dynamics simulation, user analytic codes, and visualizations with FlowVR and Damaris/Viz workflow tools. In the second example, Decaf's scalability up to 1,280 cores is studied with the analysis of dark matter that involves the exchange of complex data structures.

Currently, Decaf uses a static Python configuration file to define the workflow. In future work, bringing elasticity to Decaf will be explored, since resources may need to be redistributed among the tasks of the graph at runtime to accommodate for changes in requirements [15].

The Decaf project is available in open source.¹

Acknowledgements This work is supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Laura Biven. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

1. Abbasi, H., Wolf, M., Eisenhauer, G., Klasky, S., Schwan, K., Zheng, F.: Datastager: scalable data staging services for petascale applications. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, HPDC '09, pp. 39–48. ACM, New York, NY, USA (2009). <https://doi.org/10.1145/1551609.1551618>. <http://doi.acm.org/10.1145/1551609.1551618>
2. Ahern, S., Brugger, E., Whitlock, B., Meredith, J.S., Biagas, K., Miller, M.C., Childs, H.: Visit: Experiences with Sustainable Software (2013). [arXiv:1309.1796](https://arxiv.org/abs/1309.1796)
3. Ahrens, J., Geveci, B., Law, C.: 36 paraview: an end-user tool for large-data visualization. The Visualization Handbook, p. 717 (2005)
4. Bennett, J., Abbasi, H., Bremer, P.T., Grout, R., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., Pebay, P., Thompson, D., Yu, H., Zhang, F., Chen, J.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–9 (2012). <https://doi.org/10.1109/SC.2012.31>
5. Biddiscombe, J., Soumagne, J., Oger, G., Guibert, D., Piccinelli, J.G.: Parallel computational steering and analysis for HPC applications using a ParaView interface and the HDF5 DSM virtual file driver. In: Kuhlen, T., Pajarola, R., Zhou, K. (eds.), Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2011). <https://doi.org/10.2312/EGPGV/EGPGV11/091-100>
6. Boyuka, D., Lakshminarasimham, S., Zou, X., Gong, Z., Jenkins, J., Schendel, E., Podhorszki, N., Liu, Q., Klasky, S., Samatova, N.: Transparent I Situ data transformations in ADIOS. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 256–266 (2014). <https://doi.org/10.1109/CCGrid.2014.73>
7. Chavent, M., Vanel, A., Tek, A., Levy, B., Robert, S., Raffin, B., Baaden, M.: GPU-accelerated atom and dynamic bond visualization using hyperballs: a unified algorithm for balls, sticks, and hyperboloids. *J. Comput. Chem.* **32**(13), 2924–2935 (2011)
8. Dayal, J., Bratcher, D., Eisenhauer, G., Schwan, K., Wolf, M., Zhang, X., Abbasi, H., Klasky, S., Podhorszki, N.: Flexpath: Type-based publish/subscribe system for large-scale science analytics. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 246–255 (2014). <https://doi.org/10.1109/CCGrid.2014.104>
9. Docan, C., Parashar, M., Klasky, S.: DataSpaces: an interaction and coordination framework for coupled simulation workflows. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), pp. 25–36. ACM, New York, NY, USA (2010). <https://doi.org/10.1145/1851476.1851481>
10. Docan, C., Parashar, M., Klasky, S.: Enabling high-speed asynchronous data extraction and transfer using dart. *Concurr. Comput. Pract. Exp.* **22**(9), 1181–1204 (2010). <https://doi.org/10.1002/cpe.1567>. <http://dx.doi.org/10.1002/cpe.1567>
11. Dorier, M., Antoniu, G., Cappello, F., Snir, M., Orf, L.: Damaris: how to efficiently leverage multicore parallelism to achieve scalable, Jitter-free I/O. In: CLUSTER—IEEE International Conference on Cluster Computing. IEEE (2012)

¹ <https://github.com/tpeterka/decaf>.

12. Dorier, M., Antoniu, G., Cappello, F., Snir, M., Sisneros, R., Yildiz, O., Ibrahim, S., Peterka, T., Orf, L.: Damaris: addressing performance variability in data management for post-petascale simulations. *ACM Transactions on Parallel Computing (ToPC)* (2016)
13. Dorier, M., Dreher, M., Peterka, T., Wozniak, J.M., Antoniu, G., Raffin, B.: Lessons learned from building in situ coupling frameworks. In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp. 19–24. ACM (2015)
14. Dorier, M., Sisneros Roberto, R., Peterka, T., Antoniu, G., Semeraro Dave, B.: Damaris/Viz: a nonintrusive, adaptable and user-friendly in situ visualization framework. In: *LDAV—IEEE Symposium on Large-Scale Data Analysis and Visualization*, Atlanta, USA (2013). <https://hal.inria.fr/hal-00859603>
15. Dorier, M., Yildiz, O., Peterka, T., Ross, R.: The challenges of elastic in situ analysis and visualization. In: *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp. 23–28 (2019)
16. Dreher, M., Peterka, T.: Bredala: Semantic data redistribution for in situ applications. In: *CLUSTER—IEEE International Conference on Cluster Computing*. IEEE (2016)
17. Dreher, M., Peterka, T.: Decaf: Decoupled dataflows for in situ high-performance workflows. Technical report, Argonne National Lab. (ANL), Argonne, IL (USA) (2017)
18. Dreher, M., Prevoteau-Jonquet, J., Trellet, M., Piuzzi, M., Baaden, M., Raffin, B., Férey, N., Robert, S., Limet, S.: ExaViz: a flexible framework to analyse, steer and interact with molecular dynamics simulations. *Faraday Discuss. Chem. Soc.* **169**, 119–142 (2014). <https://doi.org/10.1039/C3FD00142C>. <https://hal.inria.fr/hal-00942627>
19. Dreher, M., Raffin, B.: A flexible framework for asynchronous in situ and in transit analytics for scientific simulations. In: *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Chicago, USA (2014). <https://hal.inria.fr/hal-00941413>
20. Dreher, M., Sasikumar, K., Sankaranarayanan, S., Peterka, T.: Manala: a flexible flow control library for asynchronous task communication. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 509–519. IEEE (2017)
21. Fabian, N., Moreland, K., Thompson, D., Bauer, A., Marion, P., Geveci, B., Rasquin, M., Jansen, K.: The paraview coprocessing library: a scalable, general purpose In Situ visualization library. In: *2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 89–96 (2011). <https://doi.org/10.1109/LDAV.2011.6092322>
22. Habib, S., Pope, A., Finkel, H., Frontiere, N., Heitmann, K., Daniel, D., Fasel, P., Morozov, V., Zagaris, G., Peterka, T., et al.: HACC: simulating sky surveys on state-of-the-art supercomputing architectures. *New Astron.* (2015)
23. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968). <https://doi.org/10.1109/TSSC.1968.300136>
24. Humphrey, W., Dalke, A., Schulten, K.: VMD—Visual molecular dynamics. *J. Mol. Graph.* **14**, 33–38 (1996)
25. Li, M., Vazhkudai, S.S., Butt, A.R., Meng, F., Ma, X., Kim, Y., Engelmann, C., Shipman, G.: Functional partitioning to optimize end-to-end performance on many-core architectures. In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’10*, pp. 1–12. IEEE Computer Society, Washington, DC, USA (2010). <https://doi.org/10.1109/SC.2010.28>. <http://dx.doi.org/10.1109/SC.2010.28>
26. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., Parashar, M., Samatova, N., Schwan, K., Shoshani, A., Wolf, M., Wu, K., Yu, W.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurr. Comput.: Pract. Exp.* **26**(7), 1453–1473 (2014). <https://doi.org/10.1002/cpe.3125>. <http://dx.doi.org/10.1002/cpe.3125>
27. Mommessin, C., Dreher, M., Raffin, B., Peterka, T.: Automatic data filtering for in situ workflows. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 370–378. IEEE (2017)
28. Morozov, D., Peterka, T.: Block-Parallel Data Analysis with DIY2 (2016)

29. Morozov, D., Peterka, T.: Efficient delaunay tessellation through K-D tree decomposition. In: Proceedings of SC16. IEEE Press (2016)
30. Morton: A computer oriented geodetic data base and a new technique in file sequencing. Technical report Ottawa, Ontario, Canada (1966)
31. Peterka, T., Croubois, H., Li, N., Rangel, E., Cappello, F.: Self-adaptive density estimation of particle data. *SIAM J. Sci. Comput.* **38**(5), S646–S666 (2016)
32. Peterka, T., Morozov, D., Phillips, C.: High-Performance computation of distributed-memory parallel 3D Voronoi and Delaunay Tessellation. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 997–1007. IEEE Press (2014)
33. Pronk, S., Pall, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M.R., Smith, J.C., Kasson, P.M., van der Spoel, D., Hess, B., Lindahl, E.: Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **29**(7), 845–854 (2013)
34. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, EGPGV '11, pp. 101–109. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2011). <https://doi.org/10.2312/EGPGV/EGPGV11/101-109>. <http://dx.doi.org/10.2312/EGPGV/EGPGV11/101-109>
35. Yu, H., Wang, C., Grout, R., Chen, J., Ma, K.L.: In situ visualization for large-scale combustion simulations. *Comput. Graph. Appl. IEEE* **30**(3), 45–57 (2010)
36. Zheng, F., Abbasi, H., Cao, J., Dayal, J., Schwan, K., Wolf, M., Klasky, S., Podhorszki, N.: In-situ i/o processing: a case for location flexibility. In: Proceedings of the Sixth Workshop on Parallel Data Storage, PDSW '11, pp. 37–42. ACM, New York, NY, USA (2011). <https://doi.org/10.1145/2159352.2159362>. <http://doi.acm.org/10.1145/2159352.2159362>
37. Zheng, F., Zou, H., Eisenhauer, G., Schwan, K., Wolf, M., Dayal, J., Nguyen, T.A., Cao, J., Abbasi, H., Klasky, S., Podhorszki, N., Yu, H.: FlexIO: I/O middleware for location-flexible scientific data analytics. In: 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), pp. 320–331 (2013). <https://doi.org/10.1109/IPDPS.2013.46>

Parameter Adaptation In Situ: Design Impacts and Trade-Offs



Steffen Frey, Valentin Bruder, Florian Friess, Patrick Gralka, Tobias Rau,
Thomas Ertl, and Guido Reina

Abstract This chapter presents a study of parameter adaptation in situ, exploring the resulting trade-offs in rendering quality and workload distribution. Four different use cases are analyzed with respect to configuration changes. First, the performance impact of load balancing and resource allocation variants on both simulation and visualization is investigated using the MegaMol framework. Its loose coupling scheme and architecture enable minimally invasive in situ operation without impacting the stability of the simulation with (potentially) experimental visualization code. Second, Volumetric Depth Images (VDIs) are considered: a compact, view-dependent intermediate representation that can efficiently be generated and used for post hoc exploration. A study of their inherent trade-offs regarding size, quality, and generation time provides the basis for parameter optimization. Third, streaming for remote visualization allows a user to monitor the progress of a simulation and to steer visualization parameters. Compression settings are adapted dynamically based on predictions via convolutional neural networks across different parts of images to achieve high frame rates for high-resolution displays like powerwalls. Fourth, different performance prediction models for volume rendering address offline scenarios (like hardware acquisition planning) as well as dynamic adaptation of parameters and load balancing. Finally, the chapter concludes by summarizing overarching approaches and challenges, discussing the potential role that adaptive approaches can play in increasing the efficiency of in situ visualization.

1 Introduction

Visualization algorithms typically expose a variety of parameters. These parameters either influence the content of the visualization (i.e., what can be seen), change its quality or influence the workload distribution. Content can be controlled via camera

S. Frey (✉)
University of Groningen, Groningen, The Netherlands
e-mail: s.d.frey@rug.nl

V. Bruder · F. Friess · P. Gralka · T. Rau · T. Ertl · G. Reina
University of Stuttgart, Stuttgart, Germany

parameters, a transfer function, filtering, or selection. Quality, on the other hand, can be influenced via sampling density (like ray step length and image resolution), optimization criteria like early termination, level-of-detail rendering, or compression settings. With this, quality and content parameters typically define the total workload. How this workload is distributed across parallel and distributed architectures can be specified via different schemes, considering various levels of granularity, and different communication patterns. Performance and/or memory (storage) requirements of visualization algorithms result from a complex interplay of parameter settings regarding context, quality, and distribution (e.g., cf. Bruder et al. [5]). In practice, algorithms may strive to achieve the highest possible quality in a given amount of time or to minimize the time required for a given workload, under the condition of additional user-definable constraints.

This chapter addresses two challenges mentioned in the introduction of this book. First, different approaches to reducing data sizes are described with a focus on studying the impact of their different parameter settings. Second, cost models are implicitly or explicitly used to adequately allocate resources, distribute computations, or adjust the total workload.

In general, to make an informed decision, the interdependency of parameters and result needs to be understood. Two use cases are covered with this focus: Sect. 2 discusses the performance impact of different load balancing and resource allocation schemes on both the simulation and the visualization. As a basis for this, it covers architectural changes to the MegaMol visualization framework to support in situ operations and describes MegaMol's loose coupling scheme between simulation code and visualization. Section 3 describes a view-dependent volume abstraction that allows for a limited degree of interaction. It discusses the impact of several parameters on representation quality, size, and generation time.

Ideally, evaluation data and gained insight provide the basis to develop a model that will allow predictions and thus serve as a basis for optimization. Another two use cases demonstrate this strategy: Sect. 4 covers a model of the impact of image compression settings on the resulting image quality and size. These predictions are used to optimize image transmission in real time for interactive remote visualization and an example is presented in which high-resolution imagery from HPC resources is streamed to a powerwall. Last, Sect. 5 presents different performance prediction models for volume rendering. The models are used to dynamically optimize rendering parameters, balance load distribution and inform hardware acquisition.

Lastly, Sect. 6 summarizes the overarching approaches that span individual use cases, outlines challenges, and concludes by discussing the potential benefit of adaptive approaches in increasing the efficiency of in situ visualization.

2 Impact of Simulation Load Balancing and Resource Allocation

Simulation software utilizes load balancing to distribute workloads as evenly as possible across compute nodes. Visualization software that runs in situ is affected by the resulting domain decomposition. Load balancing that is optimal for simulation codes can have unpredictable effects on visualization performance, since the optimization goal generally differs between simulation and analysis. As an example, molecular dynamics simulations are affected by the number of particle interactions that occur, usually influenced by a cutoff radius. Typical visualization tasks benefit from an even distribution of particle numbers instead. The following use case [20] first investigates the performance impact of two different load balancing schemes in the simulation software ls1-MarDyn [18] on visualization using MegaMol [13].

2.1 *MegaMol In Situ Loose Coupling*

MegaMol, originally conceived to be an OpenGL-accelerated workstation-centric framework, required several changes to properly operate on HPC systems. The first change consisted in adding the ability to render efficiently on CPU-only nodes. Therefore, the open source ray tracing engine OSPRay [23] was integrated together with a new internal data communication paradigm into MegaMol [21]. Note that a running MegaMol instance is always a runtime composition of interconnected modules that load, process, or render data and acquire and pass it on to their neighbors. Traditionally, visualizations using GPU rendering are generated via a strategy that immediately produces output (in form of OpenGL commands) per rendering module. Thus, data is acquired from available sources, then it is processed, rendered, and composited implicitly using the current OpenGL context (frame buffer and depth buffer).

The CPU rendering path, on the other hand, processes the data first, but then gathers all information that is necessary for rendering in the single existing OSPRay instance and renders the final image in one pass. This results in a cross-module retained mode strategy that employs data transport patterns across several layers of modules, a setup that was not necessary for the original GPU-based use cases. Note that, regardless of the rendering backend, in the case of distributed data rendering, a compositing step across nodes will be executed. In a MegaMol instance, the gathering of data in a central place is realized by chaining modules that process and generate geometry. This daisy chaining allows for composition of global scenes without requiring infrastructure or data management modules and does not generate additional programming overhead. Furthermore, the scalability of the CPU rendering has been demonstrated to offer increased rendering performance for medium-sized data sets in comparison with naive GPU ray casting [21].

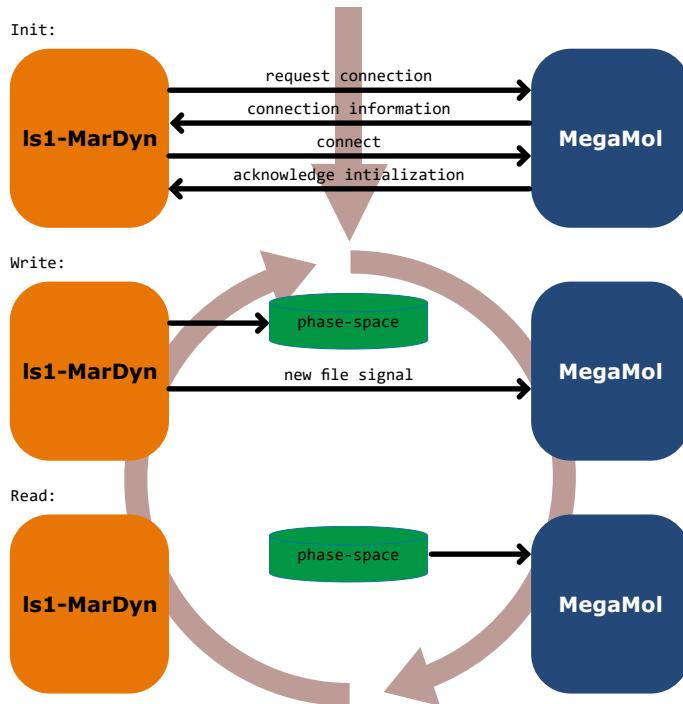


Fig. 1 Workflow within *MegaMol* in situ coupling. It starts with an initialization step that establishes communication between the simulation *Is1-MarDyn* and *MegaMol*. Then the scheme enters a loop in which checkpoints are consecutively shared with the visualization

This software-defined approach also enables off-screen rendering. In addition to the compositing of data-distributed rendering output over MPI, the ability to transport generated or composited images off site to allow for remote rendering has been added. This allows for a loose coupling scheme between simulation and visualization software for in situ rendering and analysis purposes (Fig. 1).

Data—such as the phase-space of the simulation—is communicated via shared memory (a reserved space in the node’s volatile memory), while simulation and visualization are synchronized via signals over a socket (using the Lua-based (remote) scripting interface of *MegaMol* [12]). This loosely-coupled in situ scheme differs from strategies that integrate simulation and visualization in a single executable, for example when using well-known frameworks like *Ascent* [15]. In these frameworks, transient effects or bugs that could cause the (potentially experimental) visualization to fail also lead to an interruption of the simulation, and, depending on the frequency of checkpointing, waste of computation time and results. Therefore, an in situ setup around this worst-case scenario has been designed that separates the MPI worlds of simulation and visualization completely.

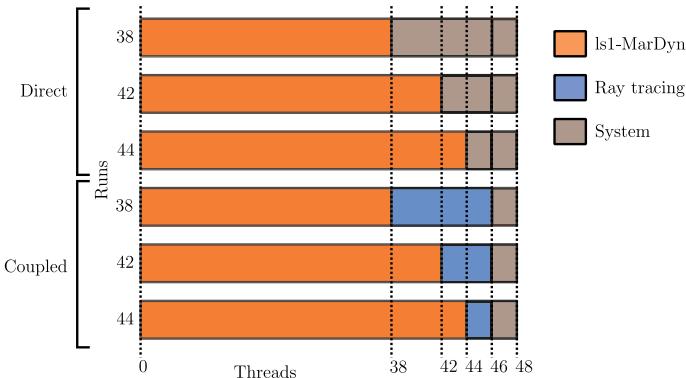


Fig. 2 Applied thread distribution per node between OS, simulation, and rendering for the measurements. For the baseline measurements without an attached rendering framework, the rendering threads are allocated for the OS, as well

In MegaMol’s loosely-coupled setting, data is shared as a copy, and the simulation can continue the computation loop while the visualization is rendering. In heavily compute-bound scenarios, such as molecular dynamics, for example, the additional memory requirements are not an issue. The communication within the MPI world of the visualization is restricted to the sharing of camera parameters and images within a compositing tree driven by IceT [17]. Only if global rendering effects are enabled, data from the simulation will be shared between MegaMol ranks.

2.2 Workload Distribution

In order to test MegaMol’s in situ coupling scheme and to evaluate performance implications of static resource allocation, a molecular dynamics simulation was conducted and visualized using ls1-MarDyn [18]. To comprehensively assess performance, renderings of the scene have been obtained from 256 different camera angles. Accordingly, the output images could also be employed for post-hoc analysis, similar to the approach of O’Leary et al. [19]. The tested workload distributions are depicted in Fig. 2, averaging over all camera configurations. The runs were performed on TACC’s Stampede2 that provides 48 cores per node, whereas two cores have been reserved for the OS. Three test cases are considered with configurations of 38 + 8 cores, 42 + 4 cores, and 44 + 2 cores for simulation and rendering respectively. For the scenarios without an attached rendering, the “rendering cores” were left unallocated for the OS (background work such as communication, I/O, etc.).

Looking at respective results in Fig. 3, it shows that enabling the visualization (denoted as the *coupled* scenario, circles) has only a small impact on the performance of the simulation with respect to the baseline (only simulation, denoted as *direct*, triangles). The captions *DOM* and *DIF* refer to the load balancing scheme of the

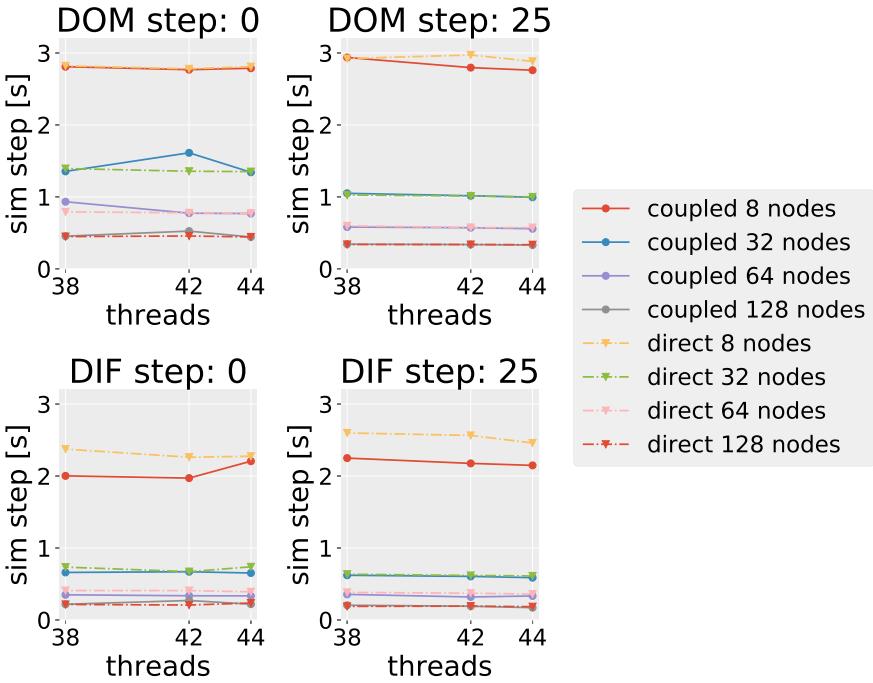


Fig. 3 Graph of wallclock time per simulation timestep over the number of available threads (median over all camera configurations). *DOM* denotes a static domain decomposition, *DIF* denotes a diffusion-based load balancer. *Coupled* values represent simulation running in situ with visualization, *direct* represents simulation code only. This image is adapted from our previous work [20]

simulation. Here, DOM is a static, equal-volume domain decomposition. DIF is a diffusion-based domain decomposition. Additionally, the measurements were taken for system states at different points in (simulation) time. While *step 0* describes the system's initial state, *step 25* represents the system after $25 \cdot 10^5$ iterations.

Given the relatively small number of particles per node resulting from the domain decomposition, the visualization also does not benefit significantly from a resource allocation of more than four threads per node (this can also be seen directly in Fig. 4 that is described below in more detail).

2.3 Load Balancing

The plots in Fig. 4 show that the rendering also benefits from the rebalancing of the simulation domains. This is the case despite the simulation balancing for a target that is not fully correlated to overall particle count. Additionally, the plots show that at higher node counts the performance gain by allocating four additional cores for

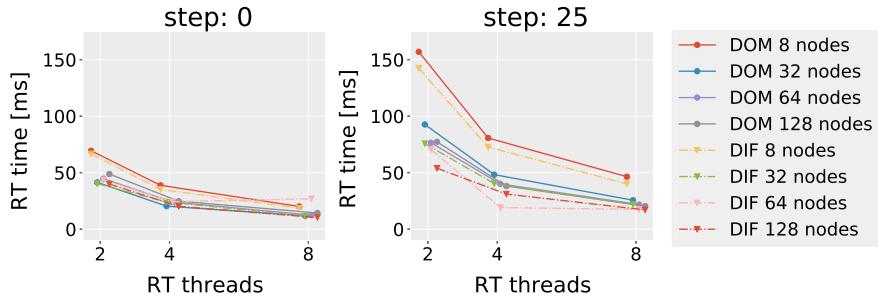


Fig. 4 Plot of ray tracing performance over the number of threads allocated for this task (median over all camera configurations). Among others, the plots show that using more than four cores only leads to small performance improvements in some cases. Dynamic load balancing of the simulation (DIF, compared to static DOM) has a positive impact also on rendering in the considered scenario. This image is adapted from our previous work [20]

the rendering is not justified. The conclusion is that in this specific case, having a rendering framework run in situ with a compute-bound simulation software has a negligible performance hit on the simulation (and even benefits from load balancing). In general, loosely-coupled in situ setups can be considered to be promising with their advantage of a low integration overhead at the cost of fine-tuned optimizations towards specific simulation software. Finally, simulation load balancing can impact visualization performance in a way that is hard to anticipate a priori. Investigating this more closely across various cases could help to better understand occurring effects.

3 Volumetric Depth Images

Volumetric Depth Images (VDIs) are a compact, view-dependent intermediate representation for in situ visualization that can efficiently be generated and used for post hoc exploration from arbitrary new views [10]. Among others, VDIs have been used for the in situ visualization of CFD simulations [1, 7]. In this use case, the focus is on different trade-offs in the generation of VDIs, their representation itself, and rendering properties [9].

3.1 VDI Generation and Rendering

VDIs offer a compact view-dependent representation of volumetric data that can be explored interactively from an arbitrary new view. For a specified camera configuration and transfer function, VDIs represent an intermediate abstraction of color-mapped volume samples—consisting of RGB values c and opacity T —clustered along viewing rays originating from the respective camera position. A VDI can effi-

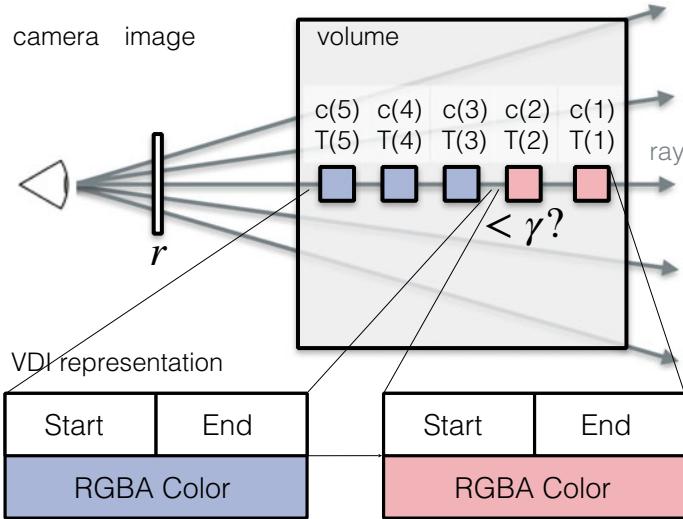
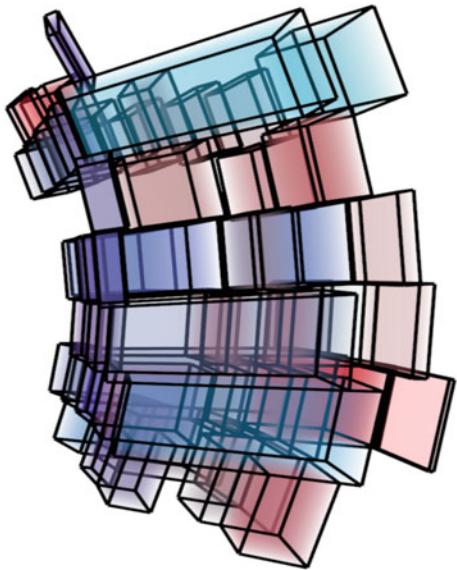


Fig. 5 VDI generation and data structure: samples are clustered along rays, only storing composited color and depth. This image is adapted from our previous work [10]

ciently be generated during volumetric raycasting by partitioning the samples along rays according to their similarity (Fig. 5). We consider two input parameters in this context. First, the resolution r depicts an image resolution of r^2 and consequently r^2 rays. Second, the threshold γ determines the maximum color difference that allows consecutive samples to be merged into an existing cluster along the ray. All colors are normalized to $[0; 1]$ and differences are computed using the Euclidean distance between pre-multiplied colors. The resulting partitions are stored as lists of elements called supersegments that contain the bounding depth pair (*Start*, *End*) and partial color accumulation values (*RGBA Color*). Empty (i.e., fully transparent) supersegments are not stored explicitly. Note that an extension to VDIs—called Space-Time VDIs—has been developed that additionally exploits inter-ray and inter-frame coherence, and introduces delta encoding for further data reduction [8]. However, this study concentrates on the representation in its original form as described above.

The storage of color alongside their respective depth values basically allows for a depth-aware reconstruction of the volume data for rendering. Each (quadrilateral) pixel in the image plane can be conceptually extruded along the ray with the used camera parameters, forming a truncated pyramid. The clustered segments along rays can then be rendered as frustum segments which represent the spatial depth range of the respective segment along a ray. For illustration, Fig. 6 shows an actual rendering of a low-resolution VDI featuring black edges around the frustums. Consistent with γ , color and opacity are constant within a frustum segment, and determined during VDI generation. For rendering, these frustums are eventually depth-ordered and composited, taking into consideration the variable segment length to ensure correct attenuation.

Fig. 6 Frustums reconstructed from sample clusters for rendering (image adapted from our previous work [10])



3.2 Parameters and Output Characteristics

In general, significant size reductions can be achieved, and renderings generated from views only moderately deviating from the initial view are typically quite faithful to original raycasting. However, results can potentially deviate noticeably for strongly different directions. This degradation of quality heavily depends on the chosen parameters for building the VDIs. A lower resolution r naturally leads to lower rendering quality but requires less space, while a lower value for γ leads to a more selective clustering and, with this, a higher depth resolution (resulting in both increased quality and storage requirements). Three different types of performance indicators are considered for quantifying the result: storage cost σ , rendering quality q , and generation time τ . Here, the compressed size (via bzip2) of the VDI representation is used for quantifying the storage cost σ . Rendering quality q is evaluated with images from different camera configurations using PSNR (peak signal-to-noise ratio, a full-reference metric that is computed on the basis of the mean squared error). For this, direct volumetric raycasting with the original data generates corresponding reference images. The implementation extends a standard CUDA volume raycaster, and kernel times τ required for VDI generation are obtained using a NVIDIA GTX980.

Generally, it is expected that input parameters γ and r yield higher quality q and require more storage space σ when they are decreased and increased, respectively. However, the rate at which they do so effectively depends on characteristics of the underlying volume data. It is evaluated here using a time step from a supernova simulation (432^3 scalar values on a uniform grid), with camera view angles ranging from 10° to 60° (in steps of 10°) around the y-axis to define characteristic extents

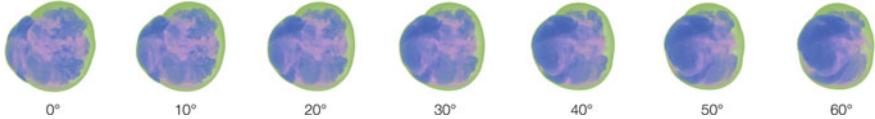


Fig. 7 Supernova (time step 20) for different angles used in the evaluation. This image is adapted from our previous work [9]

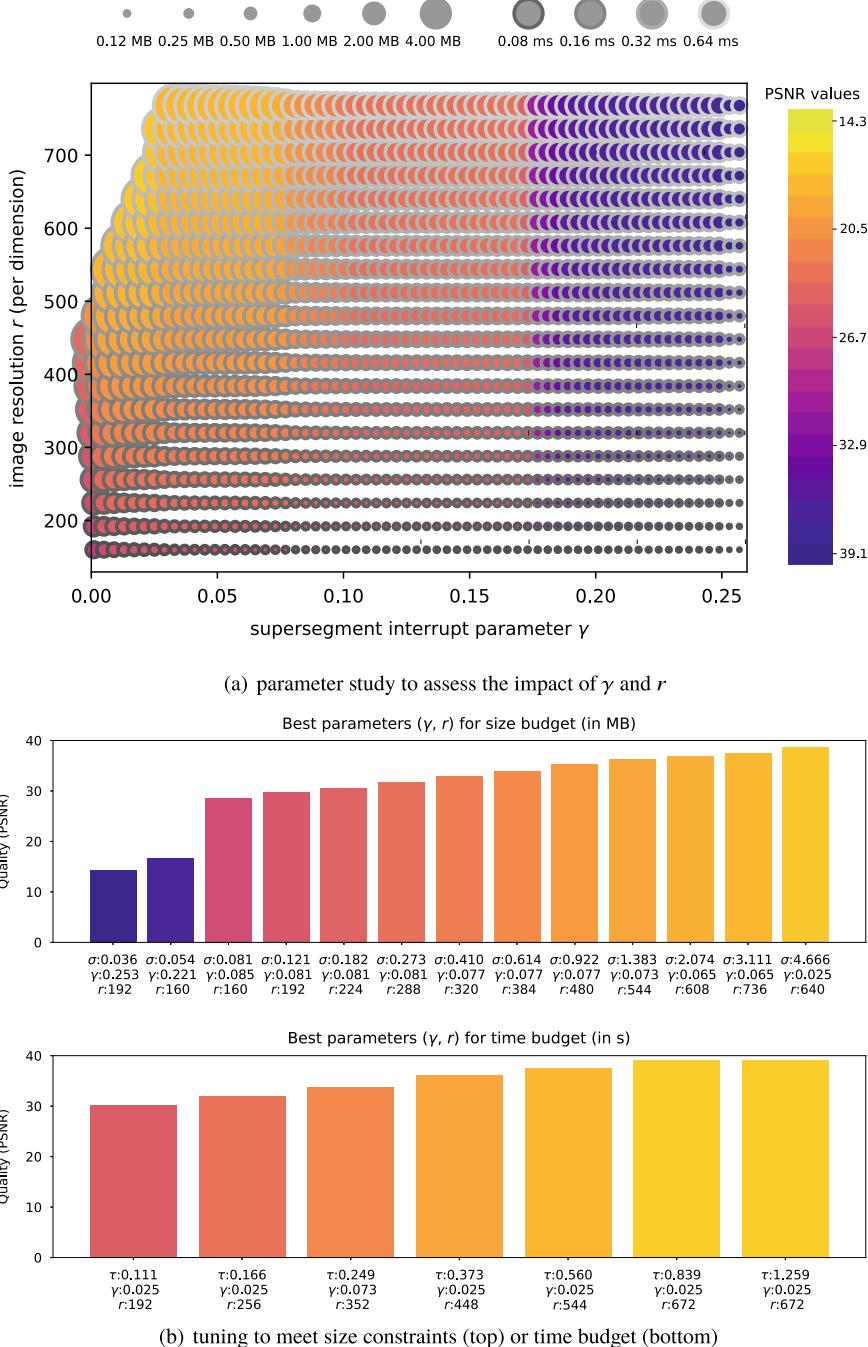
of deviation for visual exploration w.r.t. a given task (cf. Fig. 7). As indicated above, the two input parameters impacting the VDI generation are resolution $r \in [160, 768]$ (step size 32) and threshold $\gamma \in [0.001 - 0.26]$ (step size 0.004).

Results are shown in Fig. 8a. Parameters r and γ behave as expected (i.e., smaller r and larger γ worsen image quality, but yield smaller data sizes). Furthermore, the generation time primarily depends on the resolution r , naturally increasing with higher resolution as more rays are sent for VDI generation. The plot also shows that some parameter combinations seem to be better suited than others. For instance, parameter configuration ($r = 768, \gamma = 0.25$) yields approximately the same data size yet much worse quality when compared to ($r = 608, \gamma = 0.11$). On this basis, an auto-tuning approach chooses the best parameter combination for given constraints.

3.3 Evaluation Results: Auto-Tuning Toward Target Characteristic

An offline auto-tuning approach based on the measurement series described previously is used to optimize the performance of VDIs for different constraints, based on their compressed data size and generation time. The best parameter setting pair (γ, r) is determined according to utility function v . Arbitrary utility functions v can be specified to transform the value-tuple of performance indicators (σ, q, τ) into a single scalar for minimization (i.e., smaller values are better). Generally, the definition of v depends on the application use case and certain constraints induced by the overall setup. Currently, all measurements are conducted prior to the tuning. An integrated scheme could also involve adaptive refinement towards the respective goal, which has both the potential to improve accuracy and to decrease the overall computation time of the tuning (at least when tuning with one specific v that is known and fixed beforehand).

Figure 8(b, top) shows the results of a parameter study where the auto-tuning approach considers different size limitations σ_{target} , while optimizing the achieved quality. This figure shows the respective results for a range of different storage constraints (each limit is $1.5 \times$ larger than the previous one). Figure 8(b, bottom) explores generation time (τ_{target}) as the main quantity of interest. This is important for in situ scenarios in which simulation or measurement output data is produced rapidly, and the visualization needs to keep up with the data being generated. The

**Fig. 8** (Continued)

◀Fig. 8 (Continued) **a** Results of the parameter study for a range of values for γ and r . The chart depicts generation time τ (edge brightness of circle), storage space σ (circle area) and quality q (fill color of circle). Missing values are caused by VDI proxy geometry exceeding GPU memory for rendering. **b** Best parameter setting pair (γ, r) with which the specified size limitation σ_{target} (top) and time budget τ_{target} (bottom) can be undercut with maximum quality (both bar height and color depict PSNR quality). This image is adapted from our previous work [9]

image shows results for a range of time budget constraints (each limit is $1.5 \times$ larger than the previous one). In both cases, γ and r develop towards higher quality, whereas the rate at which they do so depends on the structure and characteristics of the data.

Finally, Table 1 captures how well the tuning results achieved on the basis of one time step transfer to another time step within the same series. In this example, the parameters are tuned based on a certain budget σ_{target} for storage cost. The example captured in Table 1 shows that the transfer would be feasible, i.e., the optimization target still holds when looking at another time step. Furthermore, rendering quality q and generation time τ also show similar outcomes for the same parameter configuration across these different time steps. The fact that similar performance is achieved when using a configuration that was optimized for a different time step indicates that parameter transfer works well overall in this example. However, no guarantees can be given without explicitly checking whether the constraints hold in a specific case. Therefore, when optimizing for a time series, it might be beneficial to not just base this on one but a couple of (favorably characteristic) time steps. For example, one could simply use the parameter setting yielding the best quality overall with no constraint violations across all considered time steps.

4 High-Resolution Streaming

Remote visualization via streaming allows a user to monitor the progress of a simulation and interactively steer visualization parameters. This chapter describes the dynamic adaptation of compression settings based on predictions by Convolutional Neural Networks (CNNs) [11]. CNNs have been successfully used to predict the quality of images [2, 14, 16]. In this use case, a CNN predicts the size and quality of image tiles for different compression settings in order to achieve high frame rates in different environments for high-resolution displays like powerwalls. One of the main challenges for interactive exploration is to maintain a low latency between the user input and the displayed result. The natural approach to reduce the latency of an image stream is to use strong compression, which significantly reduces the size of each image. This consequently decreases the quality of the output, losing fine details or introducing artifacts, and may greatly impact the user experience. To achieve the best possible quality for a given transfer budget in this use case, the image is split into smaller equally-sized tiles and then a CNN is used to predict (1) the size of each tile after the encoding and (2) the quality in terms of similarity to the original

Table 1 Cross-evaluation of results for the different parameter sets obtained from tuning for time steps 20 and 40 of the Supernova (for σ_{target}). In particular, it can be seen that the desired storage budget σ still holds after the transfer. In addition, rendering quality q and generation time τ also yield comparable results when using the same parameter settings across these different time steps

Source	γ	r	σ	q	τ	Source	γ	r	σ	q	τ
$\sigma_{\text{target}} = 0.08 \text{ MB}$											
40 (from 40)	0.077	160.0	0.075	27.777	0.078	40 (from 40)	0.149	256.0	0.165	30.474	0.154
20 (from 40)	0.077	160.0	0.100	28.915	0.077	20 (from 40)	0.149	256.0	0.159	29.656	0.155
40 (from 20)	0.085	160.0	0.072	27.617	0.078	40 (from 20)	0.081	224.0	0.144	29.724	0.124
20 (from 20)	0.085	160.0	0.077	28.546	0.076	20 (from 20)	0.081	224.0	0.159	30.565	0.125
Source	γ	r	σ	q	τ	Source	γ	r	σ	q	τ
$\sigma_{\text{target}} = 0.41 \text{ MB}$											
40 (from 40)	0.073	352.0	0.367	32.892	0.250	40 (from 40)	0.069	544.0	0.885	35.617	0.517
20 (from 40)	0.073	352.0	0.570	33.754	0.249	20 (from 40)	0.069	544.0	1.520	36.464	0.516
40 (from 20)	0.077	320.0	0.299	32.276	0.211	40 (from 20)	0.077	480.0	0.674	34.783	0.417
20 (from 20)	0.077	320.0	0.400	32.918	0.214	20 (from 20)	0.077	480.0	0.901	35.309	0.415

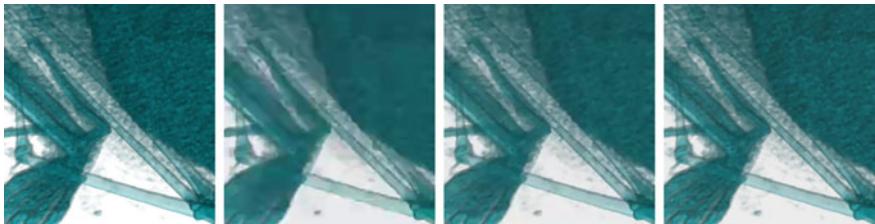


Fig. 9 Comparison between the three different encoding settings. From left to right: Original image, image encoded with the *LOW* setting, image encoded with the *MEDIUM* setting and image encoded with the *HIGH* setting. This image is adapted from our previous work [11]

image for different encoding settings. The resulting matrix is then used to optimize the encoding process.

4.1 Encoder Settings

As shown in Fig. 9, there are three different encoding settings. The *LOW* setting uses the maximum compression and leads to the smallest images sizes but with a severe impact on the resulting quality. Therefore it should not be used for tiles that contain fine structures or details that should be preserved. The *MEDIUM* setting is a trade-off between quality and size. The *HIGH* setting performs a lossless encoding. It should only be used for finer structures since the resulting images are comparably large.

4.2 Prediction of Compressed Tile Size and Quality

CNN-based regression identifies the optimal encoding setting for each tile. The output of the network is a vector, one for each tile, containing the expected quality and size for each of the encoding settings. In order to assess the image quality, the structural similarity index (SSIM) [24] is used. SSIM computes a numerical closeness indication which takes the luminance, contrast and also the structures of the encoded and original images into account.

To reduce the prediction time, the network is relatively simple, and batches of multiple tiles are used as input (Fig. 10). The first convolution layer performs the convolution with a kernel of size 7×7 and 32 kernels in total. The first RELU layer uses the output of the convolution to create a feature map of size $32 \times 240 \times 512$. Then maxpooling, with a kernel of size 4×4 pixels, is performed on the feature map which reduces the size of the map to $32 \times 60 \times 128$. The reduced size still allows the network to detect features in approximate locations, while speeding up the computation. After maxpooling, the second convolution layer is applied, followed

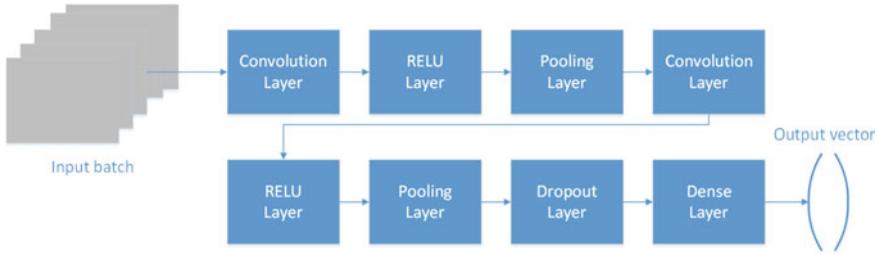


Fig. 10 Overview of all layers of the CNN used to predict the quality and size of the tiles

by the second RELU layer and the second pooling layer. This reduces the size of the feature map to $64 \times 15 \times 32$. The dropout layer addresses overfitting, and the dense layer reduces the map to the desired vector containing the six values.

The dataset in this study consists of renderings of a CT scan of a chameleon from different viewpoints with varying transfer functions. Each image was divided into equal-sized tiles of 240×512 pixels, and each tile was encoded with each of the three encoding settings. Then the SSIM value, i.e. the quality, and the size for each tile was computed. The network was then trained with 22,560 images, with each image divided into tiles and each tile encoded with each of the three settings. For each encoded tile the SSIM value and the size in bytes were stored as well.

4.3 Optimization of Encoder Settings

The goal is to achieve the highest possible encoding setting for each tile while keeping the overall size below a given threshold T . This optimization is equivalent to the multiple-choice knapsack problem and can be optimally solved by minimizing the respective objective function: $\min \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}(1 - \text{SSIM}_{ij})^2$, with N being the number of tiles and M the number of encoding settings. In doing this, two constraints are considered. The first constraint restricts each image tile to exactly one setting: $\forall j \in N : \sum_{i=0}^{M-1} x_{ij} = 1, x \in \{0, 1\}$. The second constraint limits the overall size of all tiles to be less or equal to the given threshold: $\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} \text{SIZE}_{ij} \leq T$.

In the implementation used in the evaluation below, a greedy approach approximates the optimal solution, aiming to yield a fast solution of this problem. This is done by assigning all tiles the *HIGH* setting and sorting them, in descending order, according to their predicted SSIM value of the *MEDIUM* setting. For all tiles with an SSIM value higher than 0.975 the setting is reduced to *MEDIUM* and they are sorted again according to their predicted SSIM value of the next lower setting. This process is repeated until there are no tiles left over where the setting can be reduced without losing too much quality. At this point, the sum of the predicted sizes is computed and, if it is bigger than T , the settings are again reduced where the least quality loss is predicted until the condition is met.

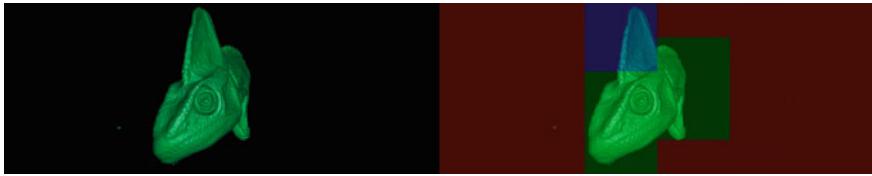


Fig. 11 The image on the right shows the used encoder settings for the image on the left. Tiles that are red used the *LOW* settings, i.e. the maximal compression, tiles that are blue use the *MEDIUM* encoder setting and the green tiles use the *HIGH* setting, i.e. minimal compression

4.4 Results

The prototype implementation achieves a frame rate between 15 and 20 frames per second when encoding images with a resolution of 4096×1200 pixels. The prediction takes most of the time as it takes about 40ms on average to estimate the size and quality for all tiles. One result of the predictions of the CNN is exemplified in Fig. 11. The left half shows the input image and the right half the tiles with the encoding settings used to encode them. It is clearly visible that only the tiles of the image that contain any details are encoded with either the *MEDIUM* (blue) or *HIGH* (green) setting and the background uses the *LOW* (red) setting. The overall quality of the resulting image is on the same level as if the complete image was encoded with the *MEDIUM* setting. The algorithm uses 10 MBit/s on average compared to 13 MBit/s for the *MEDIUM* setting.

High prediction accuracy is required to provide a solid basis for adjustments. To analyze the accuracy of the trained CNN, the mean squared error (MSE), the average absolute error (AAE), the standard deviation (STD) of the AAE and the maximum AAE for both the predicted SSIM values and the tile sizes are assessed. Since the range of the SSIM values is relatively close to the maximum, the MSE decreases from 2.43×10^{-5} , for the *LOW* setting, to 5.38×10^{-6} , for the *HIGH* setting. The same trend can be observed for the AAE, the STD and the maximum AAE as well. For the tile sizes, on the other hand, the MSE increases from 7.95×10^{-5} , for the *LOW* setting, to 1.1×10^{-3} , for the *HIGH* setting. This is due to the fact that the range of tile sizes increases with the compression quality, leading to a bigger pool of potential values which also results in higher error values. The results indicate an approximated error rate of roughly 5% for the prediction model, which is sufficient for utilizing its quality and size estimations in practice.

5 Visualization Load Balancing and Performance Prediction

During the course of a simulation, the computational effort required for rendering with a given configuration of visualization parameters can significantly vary. This can be due to changes in the data itself or an adapted partitioning across nodes. In addition, the performance can also change significantly according to the visualization parameter settings. For instance, visualization usually is rather sensitive to sampling density in object and image space as well as the camera setup, i.e., the perspective and distance to the rendered data set [5].

Estimating performance beforehand, e.g., by means of predictive modeling, has various advantages. For example, general knowledge about the computational cost of visualizing a data set of a particular size with a specific algorithm can be used for infrastructure planning [22]. Furthermore, knowing the cost of an upcoming frame at runtime can be used for parameter tuning (e.g., trading quality for responsiveness) or load balancing [4]. The former application scenario is referred to as offline performance prediction and the latter scenario is referred to as online or real-time prediction.

Due to the conceptual differences in requirements for these two scenarios, different performance models are required. Those differences include which changes need to be predicted, the required input data, and restrictions regarding the prediction run time of the model. An online model typically has a strict budget with respect to execution time, e.g., a few milliseconds if you consider an interactive application with more than 25 frames per second. For offline performance prediction, longer execution times are typically acceptable.

5.1 Real-Time Performance Prediction

Real-time performance prediction can be used as a basis to dynamically steer rendering quality or perform load balancing during runtime of an interactive application [3, 4]. Using volume raycasting as an example, the impact of rendering parameters on execution times of upcoming frames can be predicted on the fly, by using only local information. For this, a prediction model can be used that is based on a kernel recursive least squares filter [6].

This model not only features a minimal computational footprint to make the approach viable for real-time application, but also allows for online learning, basically improving the prediction accuracy with each additional frame. The core of the recursive least squared filter is an optimization aimed at recursively finding coefficients that minimize a weighted linear least squares cost function. In this case the solution is updated every frame. An extension using the kernel method with radial basis functions allows for regression analysis on non-linear data. The minimization regarding a weight w is formulated in Eq. 1:

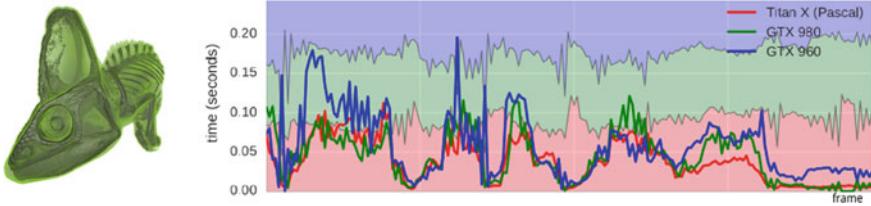


Fig. 12 Execution times per frame (solid curves) and load distribution ratios (stacked shading), on three different GPUs, for volume rendering a CT scan. The different GPUs are encoded by color, one example frame of the rendered chameleon data set is depicted on the left. The data was recorded during an interactive sequence of changes in camera configuration (viewing angle, zoom) and transfer function. This image is adapted from our previous work [4]

$$\min_w \left(\sum_i^n \lambda^{n-i} (y_i - X_i^T \times w)^2 \right) \quad (1)$$

Here, (X_i, y_i) denotes a pair of training points, where X_i is a feature vector and y_i the target scalar value. λ acts as a “forgetting factor” and is used to give exponentially less weight to older samples. The target value to predict is the cost of a single sample along the ray during volume raycasting. The feature vector contains different performance-influencing values available during rendering. This includes viewing angles, the size of a splatted voxel, the step size along the rays, and sampling depth estimation. For the latter, the depth of all rays before they terminate, due to either early ray termination or empty space skipping, has to be estimated. Every feature used in the prediction approach influences caching and execution behavior at the graphics hardware level, and thus the overall execution performance.

The predicted cost per sample combined with depth estimation, the number of rays, and the sampling distance along the rays can directly be used to calculate the execution time of the upcoming frame. The frame time predictions can be used for two purposes. First, to dynamically steer the rendering quality to achieve interactive frame rates. This is implemented by adjusting the sampling rate of the raycasting in image space as well as in object space. Second, the information of the predicted run time is used to perform load balancing across several devices. Here, the rendering task is dynamically distributed evenly across multiple available graphics cards based on a per-device prediction of the rendering time. This can be especially useful not only for dealing with dynamically varying loads but also for systems equipped with heterogeneous rendering hardware.

Figure 12 shows a graph of recorded frame execution times when using the volume renderer and the prediction model to optimize load balancing. Depicted is a sequence of frames of volume rendering of a CT scan of a chameleon and one exemplary frame of the sequence. The frame sequence consists of user interactions, i.e., camera configuration changes such as rotation and zoom, as well as modifications to the transfer function. The volume raycasting is performed on three GPUs with different computational capabilities, encoded by color in the graph. Based on the real-time prediction

of the rendering times per GPU, the load across the graphics cards is dynamically balanced. That means, the frame times per GPU (solid lines) stay comparably equal so that single GPUs neither stall overall rendering times because of slow performance, nor do GPUs idle for a longer period because of particularly fast execution. The work distribution is continually adjusted based on the rendering capabilities of the specific GPU with regard to the assigned chunk of the image.

5.2 Offline Performance Prediction

A second application scenario in this context is the static performance prediction in high performance computing environments to support rendering hardware acquisition [22]. Here, a neural network-based approach is used to predict the performance of a distributed GPU-based volume renderer. Using timing measurements of a single cluster and individual GPUs, the performance gain of upgrading or extending a cluster's graphics hardware can be predicted. Using the model, a performance prediction of upgrading the whole cluster but keeping the network configuration is also possible.

Formally, the main objective of the approach is to predict the total render time of a frame T_{cluster} based on cluster size C , data set D , the node's hardware H , image resolution I , and view parameters V :

$$(C, D, H, I, V) \rightarrow T_{\text{cluster}}. \quad (2)$$

For the neural network to make accurate predictions, training has to be done with a large amount of data to keep the model general and avoid a possible bias. However, different GPU clusters are rare and exchanging all GPUs of a cluster is unfeasible. Therefore, the initial model has to be split into two levels aligned with the two phases of an object-space distributed volume renderer: local rendering (resulting in local render time T_{local}) and compositing. Equation 2 can be divided accordingly:

$$(D, H, I, V) \rightarrow T_{\text{local}}, \quad (3)$$

$$(I, C, T_{\text{local}}) \rightarrow T_{\text{cluster}}. \quad (4)$$

In Eq. 3, data set D , hardware H , image resolution I , and view parameters V define a local render time T_{local} . Equation 4 models the compositing phase, mapping cluster size C , image resolution I , and local render time T_{local} to the final cluster frame time T_{cluster} . The advantage of this reformulation is that Eq. 4 does not contain information about the rendering hardware used. This allows for emulation of different rendering times on single nodes by stalling local execution time T_{local} , effectively generating more measurement data on a single cluster. This data is used to train the neural network that predicts Eq. 4. This model eventually captures performance characteristics of hardware for compositing, network, and topology. This means that using

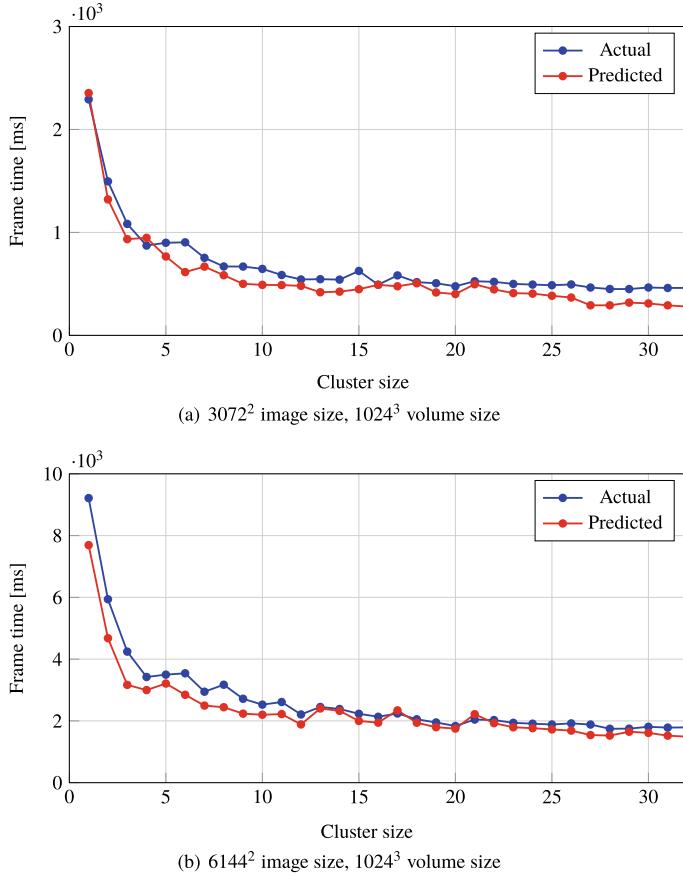


Fig. 13 Evaluation of the performance prediction model in the cluster upgrade scenario: training is done using data in single GPU mode, predicted is performance of a dual GPU cluster. This image is adapted from our previous work [22]

this model enables meaningful predictions for a cluster on the basis of local render time measurements from one node equipped with the target hardware.

The neural network contains an input layer for the input features I , C , T_{local} and T_{cluster} . Further, the experimentally determined inner structure for the network contains two hidden layers, consisting of 16 and eight neurons respectively. As activation function, the rectified linear unit (ReLU) is chosen for faster training.

The model is evaluated using two different scenarios. In the first scenario, the impact on volume rendering performance of a GPU upgrade in a cluster is predicted. For this, a 33 node cluster consisting of nodes equipped with two Intel Xeon E5620 CPUs, 24 GB RAM, two NVIDIA GeForce GTX 480 GPUs and DDR InfiniBand is used. The upgrade is emulated by deactivating one of the GPUs per node and only using data from single GPU mode for training. Testing the renderer in dual GPU

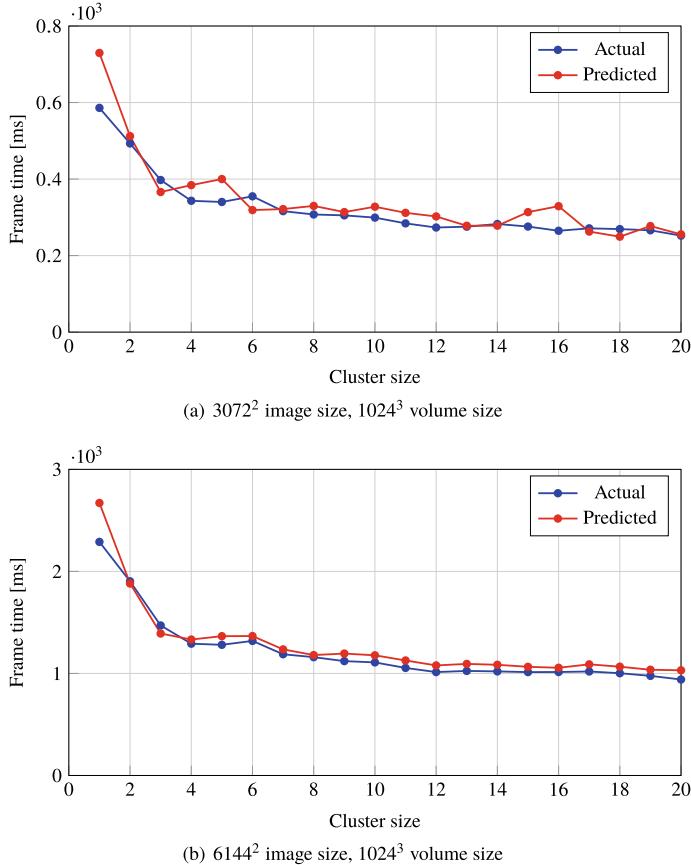


Fig. 14 Evaluation of the performance prediction model to predict the performance of a different cluster. Training is done using data from one cluster, predicted is the performance of a different cluster with a similar network configuration. This image is adapted from our previous work [22]

mode enables a comparison between prediction and actual performance. Figure 13 shows the results for two different configurations. As can be seen, the model is able to accurately predict performance for different cluster sizes (R^2 score of 0.95).

In the second scenario, an investigation on how accurately the model can predict rendering performance across multiple clusters is carried out. Training is performed on the 33 node cluster with both single and dual GPU modes. For testing, a second cluster consisting of 20 nodes, each equipped with two Intel Xeon E5-2640 v3 CPUs, 128 GB RAM, an NVIDIA Quadro M6000 and FDR InfiniBand was used. The results for different configurations are depicted in Fig. 14. The model is able to perform well in this scenario also (R^2 score of 0.93). However, in this case the graphs show a small bias (prediction of slower performance) that can be attributed to the different network interconnects used.

Overall, using the model, accurate predictions can be made for the upgrade of GPUs within a cluster as well as for upgrading to a completely new cluster with a comparable network configuration.

6 Conclusion

The content, quality, and performance characteristics of a visualization depend on a variety of different parameters. Identifying the best set of parameters for a given application can generally be considered an optimization problem. An objective function can formalize and prioritize what is desired, e.g., maximum image quality, fast response times, even distribution of load, etc. This needs to be achieved under user-definable constraints, for instance a given number of compute resources and a time budget.

To make an informed decision when addressing this optimization problem, the interdependency of parameters and the resulting outcome needs to be analyzed and understood. Earlier in this chapter, this has been exemplified with regard to performance characteristics when changing the resource allocation (Sect. 2) and evaluating the impact of VDI parameters on representation quality, size, and generation time (Sect. 3). While insights gained this way can be quite helpful in supporting manual configuration, this can also be tedious for the user and it is particularly challenging in the presence of complex interdependencies. Also, this means that adjustments cannot be flexibly made at a high rate at run-time. To enable automatic adaption, the measured evaluation data and gained insight can be employed to develop a model that allows predictions. For this, substantially different approaches to modeling can be taken, ranging from fully manual design to machine learning schemes yielding black box models. Such models can be considered surrogates of the actual visualization procedure that are much cheaper to evaluate, and with this can serve as a basis to address the aforementioned optimization problem more directly. This has been exemplified earlier in the context of adapting encoder settings based on a CNN model (Sect. 4), and adjusting sampling in volume raycasting to achieve the best quality for a given latency using kernel-based models (Sect. 5).

While striving for efficiency is generally of high relevance in scientific visualization, it is particularly crucial in HPC environments where small improvements, multiplied by the number of allocated compute nodes, have a big impact (especially considering the significant cost of compute time on HPC resources). It is even more important for in situ visualization, as the visualization can potentially have a detrimental impact on simulation performance, either by drawing on resources (cf. discussion in Sect. 2) or stalling computation when shipping data in transit or otherwise copying/serializing it (cf. loose coupling scheme in Sect. 2).

However, assessing characteristics and choosing parameters accordingly is also particularly difficult in the case of in situ visualization for various reasons. Systems are quite complex, with a variety of integrated frameworks and codes; increasingly, there are heterogeneous parallel processing hardware architectures such as CPUs

and GPUs as well as different configurations of memory layers, storage, and inter-connects. In particular, the coupling of data generation and visualization makes the whole system significantly more complex. Also, in the in situ visualization context, computational time is not the only performance metric of interest. Other important factors include, but are not limited to, memory usage, the volume of data transfers, and power consumption.

In conclusion, performance prediction and respective adaption can play an important role in automatically optimizing system performance and avoiding the need for manual adjustments. Such an approach is flexible and can adapt to changing situations, making it particularly promising for the interplay of advanced visualization techniques with long-running simulations of complex processes. At the same time, it is a challenging direction for future work. The involved complexity as outlined above means that, for in situ visualization scenarios, implementing dynamic adaptation schemes is particularly difficult. To address this, modular modeling and adaptation approaches targeting specific parts of the overall system could prove useful for handling the complexity of the full system as part of a divide-and-conquer approach.

Acknowledgements This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy—EXC-2075 (SimTech)—390740016 and as part of Project A02 of SFB/Transregio 161 (project number 251654672). It was also partially funded by the German Bundesministerium für Bildung und Forschung (BMBF) as part of project “TaLPas” (Task-based Load Balancing and Auto-tuning in Particle Simulations). We would like to thank Intel® Corporation for additional support via the Intel® Graphics and Visualization Institutes of XeLLENCE program (CG #35512501). The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this chapter. Additionally, the authors would like to thank the ls1 Mardyn development team for their support and Matthias Heinen for providing the simulation configurations.

References

1. Blom, D.S., Ertl, T., Fernandes, O., Frey, S., Klimach, H., Krupp, V., Mehl, M., Roller, S., Sternel, D.C., Uekermann, B., Winter, T., Van Zuijlen, A.H.: Partitioned fluid-structure-acoustics interaction on distributed data. In: Springer, editor, Software for Exascale Computing—SPPEXA 2013–2015, vol. 113, pp. 267–291 (2016)
2. Bosse, S., Maniry, D., Wiegand, Samek, W.: A deep neural network for image quality assessment. In: IEEE International Conference on Image Processing, pp. 3773–3777 (2016)
3. Bruder, V., Frey, S., Ertl, T.: Real-time performance prediction and tuning for interactive volume raycasting. In: SIGGRAPH ASIA 2016 Symposium on Visualization, New York, NY, USA, pp. 7:1–7:8. ACM (2016)
4. Bruder, V., Frey, S., Ertl, T.: Prediction-based load balancing and resolution tuning for interactive volume raycasting. Vis. Inf. (2017)
5. Bruder, V., Müller, C., Frey, S., Ertl, T.: On evaluating runtime performance of interactive visualizations. IEEE Trans. Vis. Comput. Graph. 1–1 (2019)
6. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least-squares algorithm. IEEE Trans. Signal Process. **52**(8), 2275–2285 (2004)

7. Fernandes, O., Blom, D.S., Frey, S., Van Zuijlen, S.H., Bijl, H., Ertl, T.: On in-situ visualization for strongly coupled partitioned fluid-structure interaction. In: VI International Conference on Computational Methods for Coupled Problems in Science and Engineering (2015)
8. Fernandes, O., Frey, S., Sadlo, F., Ertl, T.: Space-time volumetric depth images for in-situ visualization. In: IEEE Symposium on Large Data Analysis and Visualization, pp. 59–65 (2014)
9. Frey, S., Ertl, T.: Auto-tuning intermediate representations for in situ visualization. In: 2016 New York Scientific Data Summit (NYSDS), pp. 1–10 (2016)
10. Frey, S., Sadlo, F., Ertl, T.: Explorable volumetric depth images from raycasting. In: Conference on Graphics, Patterns and Images, pp. 123–130 (2013)
11. Frieß, F., Landwehr, M., Bruder, V., Frey, S., Ertl, T.: Adaptive encoder settings for interactive remote visualisation on high-resolution displays. In: Symposium on Large Data Analysis and Visualization (LDAV) (2018)
12. Gralka, P., Becher, M., Braun, M., Frieß, F., Müller, C., Rau, T., Schatz, K., Schulz, C., Krone, M., Reina, G., Ertl, T.: MegaMol—A comprehensive prototyping framework for visualizations. *Eur. Phys. J. Spec. Top.* **227**(14), 1817–1829 (2019)
13. Grottel, S., Krone, M., Müller, C., Reina, G., Ertl, T.: Megamol—a prototyping framework for particle-based visualization. *IEEE Trans. Vis. Comput. Graph.* **21**(2), 201–214 (2015)
14. Kang, L., Ye, P., Li, Y., Doermann, D.: Convolutional neural networks for no-reference image quality assessment. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1733–1740 (2014)
15. Larsen, M., Ahrens, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: 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, ISAV’17, pp. 42–46, New York, NY, USA. ACM (2017)
16. Li, C., Bovik, A.C., Wu, X.: Blind image quality assessment using a general regression neural network. *IEEE Trans. Neural Netw.* **22**(5), 793–799 (2011)
17. Moreland, K., Kendall, W., Peterka, T., Huang, J.: An image compositing solution at scale. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, New York, NY, USA. ACM (2011)
18. Niethammer, C., Becker, S., Berreuther, M., Buchholz, M., Eckhardt, W., Heinecke, A., Werth, S., Bungartz, H.-J., Glass, C.W., Hasse, H., Vrabec, J., Horsch, M.: ls1 mardyn: the massively parallel molecular dynamics code for large systems. *J. Chem. Theory Comput.* **10**(10), 4455–4464 (2014). PMID: 26588142
19. O’Leary, P., Ahrens, J., Jourdain, S., Wittenburg, S., Rogers, D.H., Petersen, M.: Cinema image-based in situ analysis and visualization of MPAS-ocean simulations. *Parallel Comput.* **55**, 43–48 (2016)
20. Rau, T., Gralka, P., Fernandes, O., Reina, G., Frey, S., Ertl, T.: The impact of work distribution on in situ visualization: a case study. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV ’19, New York, NY, USA, pp. 17–22. ACM (2019)
21. Rau, T., Krone, M., Reina, G., Ertl, T.: Challenges and opportunities using software-defined visualization in megamol. In: Workshop on Visual Analytics, Information Visualization and Scientific Visualization (WVIS) in the 30th Conference on Graphics, Patterns and Images (SIBGRAPI’17) (2017)
22. Tkachev, G., Frey, S., Müller, C., Bruder, V., Ertl, T.: Prediction of distributed volume visualization performance to support render hardware acquisition. In: Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2017)
23. Wald, I., Johnson, G., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navratil, P.: OSPRay—A CPU ray tracing framework for scientific visualization. *IEEE Trans. Vis. Comput. Graph.* **23**(1), 931–940 (2017)
24. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* **13**(4), 600–612 (2004)

Resource-Aware Optimal Scheduling of In Situ Analysis



Preeti Malakar, Venkatram Vishwanath, Christopher Knight, Todd Munson, and Michael E. Papka

Abstract This chapter describes methodologies to perform in situ computations at desired intervals along with the simulations for different execution modes. This needs to be done in a way such that the simulation throughput is minimally impacted and the analysis output is available immediately within desired intervals. We describe the formulation of optimal resource allocation for simulation and in situ analysis computations as constrained integer linear programs so that the end-to-end simulation-analysis time is minimized. In particular, we describe the scheduling of in situ analyses as a numerical optimization problem to maximize the number of online analyses and minimize overall runtime, subject to resource constraints such as I/O bandwidth, network bandwidth, rate of computation and available memory. We also demonstrate the effectiveness of our approach through real application case studies on supercomputers.

1 Resource Requirements of In Situ Data Analysis

The in situ analyses performed with the simulations are driven by the requirements of the scientists. These analyses may range from collecting simple descriptive statistics to complex principal component analysis. Thus the compute and memory require-

P. Malakar (✉)

Indian Institute of Technology Kanpur, Kanpur, India
e-mail: pmalakar@iitk.ac.in

V. Vishwanath · C. Knight · T. Munson · M. E. Papka
Argonne National Laboratory, Lemont, IL, USA
e-mail: venkat@anl.gov

C. Knight
e-mail: knightc@anl.gov

T. Munson
e-mail: tmunson@anl.gov

M. E. Papka
e-mail: papka@anl.gov

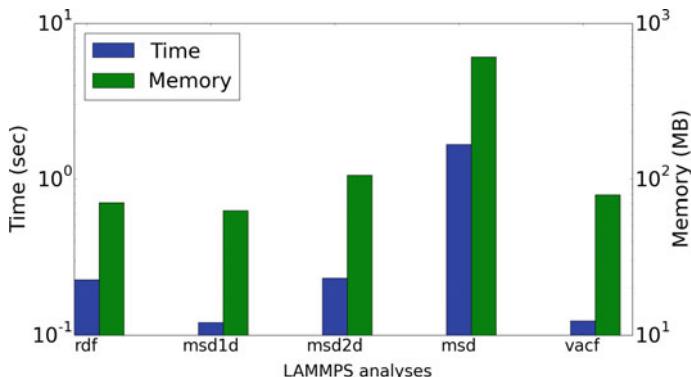


Fig. 1 Compute and memory requirements of different analyses of molecular dynamics simulations. Left y-axis depicts compute requirements, right y-axis depicts memory requirements. This image is reprinted from our previous work [11]

ments of the different desirable analyses may vary. An example of five different analyses of the LAMMPS molecular dynamics simulation code [15] is shown in Fig. 1. Modern simulation codes have diverse resource requirements, such as memory and compute times. Compute-intensive simulations require more processing power and thus may have smaller tolerance for in situ analyses, which may result in lower in situ analyses frequencies. Memory-intensive simulations may have less available free memory for analyses, which may impact the frequency of memory-intensive analyses. Thus, system characteristics play a key role in determining the feasibility of the analyses. Further, slow I/O bandwidth affects the frequency of simulation and analyses output. Therefore, it is essential to carefully schedule the analyses based on the requirements and the feasibility.

The desired frequency of performing an analysis depends on the type of the analysis as well as resource constraints. For certain analyses, it may be desirable to perform the analysis as frequently as possible. However, performing analysis after every simulation time step is nearly infeasible due to resource constraints such as available memory and compute time overhead due to in situ analysis. Certain analyses may also have a desirable minimum interval between two analysis steps. For instance, statistical analyses such as weekly averages in climate simulations have to be performed at a certain fixed frequency. Performing multiple analyses at a high frequency, though desirable, may not be possible due to resource constraints. Thus, the scientists need to determine the feasibility of performing each in situ analysis and its frequency *a priori*. For this, we need to consider both application resource requirements as well as computing resource characteristics. This is important in order to meet the science goals within desired time limits. In this chapter, we describe optimal scheduling of in situ analysis based on the current resource configurations and the application characteristics and requirements.

The mode of simulation and analysis execution also dictates the feasibility. In this chapter, we describe three different modes of in situ analysis. First we describe the

modes, their resource characteristics and the coupling between the simulation and analysis for each mode. Next, we will present the formulation of the optimization problem for scheduling the in situ analyses. We consider the following three modes of in situ analysis. They differ in proximity, access and division of execution.

1. Mode 1 (Same job, tightly-coupled): The simulation and analysis processes execute on the same set of processes alternately. This is the time sharing mode (time division).
2. Mode 2 (Same job, loosely-coupled): The simulation and the analysis processes execute on a different set of processes simultaneously. This is categorized as space sharing (space division).
3. Mode 3 (Different jobs, loosely-coupled): The simulation and analysis runs as two different jobs, either on the same supercomputer or different. They share data via some common storage.

2 Effect of System Parameters on In Situ Analysis

An in situ data analysis consists of computing certain quantities. An analysis step occurs after a certain number of simulation time steps, denoted as analysis frequency. This is often determined by the user empirically. Here, we consider that this frequency is set at the beginning of the job, and does not change during the execution. However, the analysis frequency has a large range (anywhere between 1 and the total number of simulation time steps). Since we intend to perform as many analysis computations in situ to maximize the insight gained from the simulations, the goal is to maximize the number of analysis steps. However, executing in situ analysis may increase the memory and compute requirements of the overall application. For example, the higher analysis execution times, the lower the analysis frequency must be. This may depend on a user-specified threshold for tolerance on compute time overhead due to in situ analysis. In the following subsections, we will describe the methodology to achieve this goal.

In addition to memory and execution time, there are other factors that determine the execution sequence of simulation and analysis depending on the code. Certain implementation designs play a key role. The analysis may output its result instantaneously or utilize an analysis memory buffer for further computation later followed by output. For example, the daily average temperature in a climate simulation may be computed every 24 simulation hours and stored as running average in the local node memory, followed by output every 48 simulation hours. The memory required by analysis may be pre-allocated in some applications, while some may require allocation at every analysis step and hence there may be an additional time overhead. The analysis memory may be freed after computation and output or retained. For example, in the molecular dynamics simulation code LAMMPS [15], certain analyses such as the calculation of mean square displacements of molecules require a large pre-allocated memory for subsequent analysis as well as additional memory during

the analysis. On the other hand, the multiphysics, multiscale astrophysics simulation code FLASH [8] allocates and deallocates memory required for analysis on-the-fly. The coupling is also influenced by the science objectives. For instance, in certain production runs, a scientist may require a lower execution overhead (threshold) for the total in situ analyses' execution time while they may set a higher threshold during exploratory and debug runs. Additionally, the problem being simulated may require a set of analyses to be performed at simulation time, with each analysis having different importance values. Application developers may select any model of analysis coupling from these wide range of options. We carefully consider and model many such possible scenarios for in situ analysis. We present comprehensive models based on linear optimization for the three in situ analysis modes.

3 Optimal Scheduling for Mode 1 (Same Job, Space-Division)

Figure 2 depicts a possible interleaving of simulation and analysis time steps in this case. The figure illustrates the time steps when simulation (S) is executed and analysis (A) is executed. It also shows the time steps when simulation output (O_S) and analysis output (O_A) are written to disk. The simulation output frequency is decided by the user. Analysis may output to disk after some number of analysis steps. This analysis output frequency and analysis frequency impacts the overall end-to-end simulation time. This is because the processors are blocked for analysis and the simulation is stalled. We have developed a model to recommend the optimal analysis frequency and its output frequency. We will next describe a mathematical model of these executions and our assumptions with respect to resource requirements of simulation and analysis.

3.1 Problem Parameters

Let us assume that we are given a set of desired analyses \mathcal{A} to be performed in situ. Each analysis is associated with time and memory requirements. Table 1 describes

S – Simulation, O_S – Simulation Output
 A – Analysis, O_A – Analysis Output
 $SSSSASO_SSSAO_ASSO_SSASSSO_SA_O_ASSS$

Fig. 2 Analysis occurs every 4 simulation time steps. Analysis outputs every 2 analysis time steps. Simulation outputs every 5 simulation time steps. This image is reprinted from our previous work [12]

Table 1 Input parameters for each analysis $i \in \mathcal{A}$ and available resources

Parameter	Parameter description
ft_i	Fixed setup time required per analysis
it_i	Time required per analysis per simulation time step
ct_i	Compute time required per analysis step
ot_i	Output time required per output step
cth	Threshold (time) per simulation step for analyses
fm_i	Fixed memory allocated per analysis
im_i	Input memory allocated per analysis per simulation step
cm_i	Memory allocated per analysis step
om_i	Memory allocated per output step
mth	Maximum memory available for analyses
w_i	Weight (importance) of each analysis
itv_i	Minimum interval between analysis steps
bw	Average I/O bandwidth between simulation site and disk

the input parameters for each analysis $i \in \mathcal{A}$. Let ft and fm refer to the fixed initialization time and memory allocation required by an analysis at the start of the simulation execution, a one-time cost. Let it and im be the execution time and memory required at every simulation time step to facilitate the analysis during the analysis step. A simple example is temporal analysis, where it is the time required to copy simulation data from simulation memory to temporary analysis memory so that data is not overwritten and facilitates temporal analysis. In such cases, this additional overhead is incurred at each simulation step to facilitate the analysis at the analysis step. Let ct be the time required for performing the analysis during the analysis step, and cm be the associated memory required to facilitate this analysis. A simple example of cm is the additional memory needed by the analysis to allocate any intermediate memory needed for the analysis computation. Let om refer to the output memory for the results produced by the analysis after computation and ot be the time required to write this analysis output to the storage.

Let bw be the average write bandwidth to disk from the simulation site. Application scientists may provide an upper threshold on time allowed for in situ analyses. cth denotes the maximum threshold on analysis time per simulation time step. Let $Steps$ denote the total number of simulation time steps. A common usage scenario for in situ analysis is where the application scientist provides an upper bound on the total overhead of performing the in situ analyses. For example, one could require a maximum of 10% overhead on the overall simulation. Let mth refer to maximum memory available for analyses during the entire simulation. Scientists are expected to run multiple analyses in situ wherein the analyses will be of varying importance to the simulation. We model this importance by assigning a weight w to each analysis; higher weight implies more importance. Additionally, a large class of analyses require a minimum interval of itv steps between consecutive analyses steps. Examples of such analyses include daily mean average temperature in climate simulations.

3.2 Problem Formulation

The problem is to schedule these analyses such that maximum insight is gained with minimum overhead. We formulate this problem as a mixed-integer linear program [14] with the objective of maximizing the following:

- the number of times $|\mathcal{C}_i|$ each analysis i is performed.
- the total number of different analyses, $|\mathcal{A}|$.
- the total importance of analyses performed in situ. This is denoted by $w_i * |\mathcal{C}_i|$.

The decision variables are the set of feasible in situ analyses \mathcal{A} , the set of simulation time steps at which analysis is performed \mathcal{C} , and the set of simulation time steps at which the analysis output is written \mathcal{O} . The objective function of the linear program is given as

$$\text{maximize} \left(|\mathcal{A}| + \sum_{i \in \mathcal{A}} w_i * |\mathcal{C}_i| \right) \quad (1)$$

Next, we describe and model the two main constraints – the execution time associated with various components of the in situ analyses and their associated memory costs.

Time constraint: The desirable in situ analyses must complete within a stipulated additional time such that the simulation time is not significantly increased. This additional time for all analyses should be less than the maximum allowable threshold as specified by the scientist. This time includes the time for all additional computations and I/O such as the initialization/setup time per analysis (ft), time required for analysis per simulation time step (it), time required for performing analysis (ct), and time required for writing the analysis output (ot). These times depend on the type of analysis and its implementation, and therefore may not be present in some cases. Time constraints are shown in Eqs. 2–3. $t\text{Analyze}_{i,j}$ is the cumulative time spent on the i th in situ analysis from start of simulation to j th time step. The initialization cost (time) is added only once at step 0 for $t\text{Analyze}$ for each feasible analysis as shown in Eq. 3.

$$\begin{aligned} t\text{Analyze}_{i,j} &= t\text{Analyze}_{i,j-1} + it_i \\ &\quad + ct_i \text{ (if } j \in \mathcal{C} \text{)} + ot_i \text{ (if } j \in \mathcal{O} \text{)} \\ &\quad \forall i \in \mathcal{A}, j \in \{1, \dots, \text{Steps}\} \end{aligned} \quad (2)$$

$$t\text{Analyze}_{i,0} = ft_i \quad \forall i \in \mathcal{A} \quad (3)$$

$$\sum_{i \in \mathcal{A}} t\text{Analyze}_{i,\text{Steps}} \leq cth \cdot \text{Steps} \quad (4)$$

it is the cost (time) paid every simulation step. ct is the time required to perform the analysis computation at every analysis step, and ot is paid whenever analysis output is written to disk. ot can be substituted by $\frac{om_i}{bw}$. Note that it is not required that

the time spent on performing analysis at one analysis step is less than the allowed threshold per simulation time step, cth . However, at the end of the simulation, the sum of the total time spent on all in situ analyses should be less than the threshold $cth \cdot Steps$. This is shown in inequality Eq. 4.

Memory constraint: The analysis computations are feasible only when required memory is available on the compute nodes. An analysis may allocate a fixed amount of memory at the beginning of the simulation or may allocate new memory at every simulation time step. In situ analysis may also allocate memory at every analysis step and deallocate memory at every output step. These choices depend on the analysis type and its implementation. The total memory required by an analysis will be the sum of its input and output memory requirements. This sum should be less than the available memory on the nodes. Here, we consider these different scenarios through the constraints defined in Eqs. 5–8. $mStart$ and $mEnd$ are the amounts of memory used by an analysis at the start and end of each simulation step. Equation 5 shows the equation for $mStart$. At every step, the variable memory im may be allocated, hence it is added to $mStart$ along with the available memory at the end of the previous time step $j - 1$. At every analysis step, cm may be additionally allocated, and hence it is conditionally added to $mStart$. At every output step, analysis may allocate separate buffer om for output. Therefore, om is also conditionally added to $mStart$. $mEnd$ is the available memory after the end of each step, which is equal to $mStart$ at all steps, except for output step. At the end of output step, the additional analysis memory buffers are assumed to be freed, and reset to the initial memory allocation fm as shown in Eq. 6. Memory is constrained by mth , hence the sum of $mStart$ for all analyses in all time steps should be less than mth as shown in Eq. 8.

$$\begin{aligned} mStart_{i,j} = & mEnd_{i,j-1} + im_i \\ & + cm_i \text{ (if } j \in \mathcal{C}) + om_i \text{ (if } j \in \mathcal{O}) \\ \forall i \in \mathcal{A}, j \in \{1, \dots, Steps\} \end{aligned} \quad (5)$$

$$mEnd_{i,j} = \begin{cases} fm_i & \text{(if } j \in \mathcal{O}) \\ mStart_{i,j} & \text{(otherwise)} \end{cases} \quad \begin{aligned} \forall i \in \mathcal{A}, j \in \{1, \dots, Steps\} \end{aligned} \quad (6)$$

$$mEnd_{i,0} = fm_i \quad \forall i \in \mathcal{A} \quad (7)$$

$$\sum_{i \in \mathcal{A}} mStart_{i,j} \leq mth \quad \forall j \in \{1, \dots, Steps\} \quad (8)$$

Interval constraint: The minimum interval between analysis steps itv_i provides an upper bound for the number of analysis steps during $Steps$ simulation times steps. $\frac{Steps}{itv_i}$ denotes the maximum number of analysis steps for i th analysis. We keep a running total of the number of steps in which analysis is not performed and require this running total to exceed itv_i before an analysis can be performed. The running total is reset after the analysis is performed.

$$|\mathcal{C}_i| \leq \frac{\text{Steps}}{itv_i} \quad \forall i \in \mathcal{A} \quad (9)$$

Solution: The solution for \mathcal{C} and \mathcal{O} are obtained from the above constraints by introducing 0–1 variables for the conditional equations. For example, in Eq. 5, the third term of the right hand expression cm_i (if $j \in \mathcal{C}$) is replaced by $om_i \cdot analysis_{i,j}$. Here, the binary variable $analysis_{i,j}$ is 0 when there is no analysis after the j th simulation step and 1 when there is an analysis. Similarly, in Eq. 5, the last term om_i (if $j \in \mathcal{O}$) is replaced by $om_i \cdot output_{i,j}$ where the binary (0 or 1) variable $output$ is 0 when there is no output after an analyses performed in step j and 1 otherwise. Hence, the count of $analysis$ and $output$ for each analysis in \mathcal{A} gives the solution to our problem. The analyses for which the counts are positive form the set \mathcal{A} .

4 Optimal Scheduling for Mode 2 (Same Job, Time-Division)

The simulation and analysis processes are part of the same job in the time sharing case. They execute on different set of nodes. The simulation processes transfer data directly to the analysis processes, alleviating I/O cost for analysis. We assume that each analysis process receives data from one or more simulation processes. Here, we consider blocking sends. The simulation does not stall for the analysis, except for data transfer. We also assume that data for all feasible analysis is transferred at the same time. Here, the total simulation and analysis time depends on the number of processes allocated to each, along with the transfer time, the data size, and the transfer frequency from simulation to analysis processes. Several code changes may be required in the simulation and analysis codes, depending on the application implementation. The simulation and analysis should set up different MPI communicators. Thus, in order to minimize the total execution time of the simulation and analysis, the analysis processes must be ready to receive data when the simulation processes send. The goal is to determine the optimal transfer frequency from the simulation to the analysis processes, without stalling the simulation and performing as many analyses as possible within that time. First we discuss the problem parameters for this case, followed by our mathematical formulation.

4.1 Problem Parameters

Table 2 illustrates the system and simulation parameters for this case. The first column depicts the notation for the various parameters used in the formulation. Let \mathcal{A} be the set of desired analyses. $steps$ denotes the total number of simulation time steps. $sime$ denotes the simulation time per time step, which depends on the number of processes

Table 2 System parameters and application (simulation+analysis) parameters

Parameter	Parameter description
<i>steps</i>	Total number of simulation time steps
<i>stime</i>	Simulation time per time step
<i>atime_i</i>	Analysis time per time step for <i>i</i> th analysis
<i>imp_i</i>	Importance of <i>i</i> th analysis
<i>execute_i</i>	Execution feasibility of <i>i</i> th analysis
<i>n</i>	Analysis interval
<i>xfer_i</i>	Time required to transfer data for <i>i</i> th analysis
<i>amem_i</i>	Memory required for <i>i</i> th analysis
<i>maxmem</i>	Maximum available memory in analysis processes
<i>xfer_UB</i>	Transfer time upper bound
<i>n_LB</i>	Analysis interval lower bound

allocated for the simulation job. m denotes the number of desired analyses for the simulation. $atime_i$ denotes the computation time for the i th analysis, which depends on the number of processes allocated for the analysis and the scalability of the analysis computation. imp_i denotes the importance of the i th analysis. The importance of the analysis can be taken into account when resource constraints do not permit the execution of all the various analysis computations along with the simulation. n is the number of simulation time steps between two consecutive analysis steps. $amem_i$ is the required memory and $xfer_i$ is the data transfer time for the i th analysis. This depends on the data size and the network bandwidth between the simulation and analysis processes. The maximum available memory in the analysis processes is denoted by $maxmem$. Typically analysis runs on fewer cores than simulation. Thus the available memory in the analysis processes is limited. Also, the memory requirement for the data transfer in the analysis cores is significantly high for large-scale simulations. Hence it is essential to consider the memory requirements of every analysis to determine its in situ execution feasibility.

$xfer_UB$ and n_LB (last two rows in the table) are user-specified thresholds. $xfer_UB$ is the upper bound on the time to transfer data from simulation to analysis. This is effectively a bound on the stall time of simulation while the data is transferred. This can be determined by the user based on resource availability. n_LB is the lower bound on analysis interval. This may be determined by the user based on the type of analysis and simulation. The goal is to perform multiple analyses at the maximum possible frequency. However, we are constrained by the total execution time and the available memory. We have formulated this problem as a mixed integer linear program as described next.

4.2 Problem Formulation

The two objectives are (1) to maximize the execution of the most important analyses and (2) to minimize the number of simulation time steps between two analysis steps. These objectives ensure that maximum insight from the simulation is available quickly and we do not miss important events. We apply the weighted sum method to find the Pareto optimal solution [6]. We construct a single-objective optimization problem using linear scalarization. The objective function has two weighted terms as shown in Eq. 10. The decision variables are the feasibility, $execute_i$, for each analysis and analysis interval, n . The i th analysis is executed if $execute_i$ (binary decision variable) is 1. The first term denotes the overall importance of the feasible analyses and the second term denotes the analysis interval n multiplied by a weight w , which can be chosen based on the subjective preference of the user. Such multi-objective optimization problems often require human intervention to determine a useful solution. w can be set according to the user's preference to minimizing the analysis interval over the number of analyses.

$$\text{maximize} \left(\sum_{i \in \mathcal{A}} imp_i \times execute_i \right) - w \cdot n \quad (10)$$

The constraints related to time, memory and bounds are shown in Eqs. 11–14. Equation 11 signifies the constraint on execution time of all feasible analyses. We assume that all the analyses are executed one after another in an analysis time step. Thus, the time to compute all feasible analyses should be less than the time between two analysis steps ($= n \times stime$). This ensures that in situ analysis does not stall simulation, and the analysis processes are done with the analyses and are ready to receive data after n simulation time steps. Equation 12 ensures that the sum of memory required by all feasible analyses is less than the available memory in the analysis processes. Equation 13 specifies that the total transfer time for the feasible analyses for $\frac{steps}{n}$ times is less than the user-specified threshold $xfer_UB$. This bound is useful in case of low network bandwidths and high transfer times for large data. Finally, the recommended analysis interval n should be greater than the minimum analysis interval (Eq. 14).

$$\sum_{i \in \mathcal{A}} atime_i \cdot execute_i \leq n \cdot stime \quad (11)$$

$$\sum_{i \in \mathcal{A}} amem_i \cdot execute_i \leq maxmem \quad (12)$$

$$\sum_{i \in \mathcal{A}} xfer_i \cdot execute_i \cdot steps \leq n \cdot xfer_UB \quad (13)$$

$$n \geq n_LB \quad (14)$$

Table 3 Analysis and system parameters for Mode 3

Parameter	Parameter description
$treq_i$	Number of time steps required for 1 analysis step of i th analysis
$wsize_i$	Output size for 1 analysis step of i th analysis
$wtime_i$	Write time for 1 analysis step of i th analysis
$rtime_i$	Read time for 1 analysis step of i th analysis
na_i	Number of times i th analysis can be performed
\mathcal{D}	Available storage space
WT_UB	Upper bound on write time
AT_UB	Upper bound on analysis time

4.3 Optimal Scheduling for Mode 3 (Different Jobs)

In this mode of execution, we assume that the simulation and analysis jobs are loosely-coupled, executing on a different set of nodes as different jobs. The simulation job writes output to a shared storage at a specified output frequency. This storage may be based on non-volatile memory [13], SSD or spinning disks. The amount of available storage space and its I/O bandwidth depends on the storage medium. We assume that there is a finite storage space to write the output. The analysis job reads the output while the simulation is running and performs various analysis computations. In contrast to traditional post-processing, the analysis job starts processing the output as soon as a new time step is written by the simulation. The simulation updates this information to a configuration file. This approach requires minimal changes to the simulation code. The simulation is typically a more compute-intensive job and requires a large number of processes, while the analysis requires fewer processes. Using this approach, the simulation can continue progressing efficiently without stalling for the analysis phase. The rate of the simulation (throughput) depends on the number of processors used by the simulation and the time required to write the output. The analysis job's throughput depends on the time to read the output and to compute its analysis. Large-scale simulations write outputs typically at an empirically determined output frequency. However, due to storage space limitations, large-scale simulations such as billion-atom molecular dynamics simulations and trillion particle cosmological simulations may quickly overflow their storage limits. Hence, an optimal output frequency needs to be determined. The goal is to perform the most desired analyses at simulation-time as frequently as possible, constrained to minimizing the overall runtime. Next we describe the problem parameters and formulation.

Table 2 shows some of the parameters used in this formulation. Additional parameters are shown in Table 3. The first column shows the notation for the various parameters used in this formulation. We assume that one analysis computation step for an analysis requires simulation output of $treq_i$ time steps. $treq_i$ is 1 for non-temporal analysis. Temporal analyses such as time series analysis may require more than 1

time step to perform the analysis. $wsize_i$ denotes the size of output of 1 analysis step for i th analysis in \mathcal{A} . $wtime_i$ denotes the time to write the simulation output for 1 analysis time step of the i th analysis. $rtime_i$ denotes the read time for 1 analysis step of i th analysis. na_i is the frequency of i th analysis.

Large-scale simulations are capable of producing terabytes of output within a few hours [9]. For example, a 100 million-atom molecular dynamics simulation can generate 3 TB of data in less than 30 minutes when the output frequency is high. Therefore the total available storage space, \mathcal{D} , is an important factor in the formulation. We also consider two user-defined parameters— WT_UB , the upper bound on write time by the simulation, and AT_UB , the upper bound on analysis time by the analysis. These thresholds may be decided by the user based on available compute resources. The objective of our problem is to determine the optimal frequency and feasibility of each analysis, given the various input parameters specified in Tables 2 and 3. We formulate this as a mixed integer linear program (MILP) [4, 14] with the objective function shown in Eq. 15.

$$\text{maximize} \left(\sum_{i \in \mathcal{A}} imp_i \times na_i + \sum_{i \in \mathcal{A}} execute_i \right) \quad (15)$$

$$na_i \leq \left(\mathcal{D} \cdot \frac{imp_i}{total_importance} / wsize_i \right) \cdot execute_i \quad \forall i \in \mathcal{A} \quad (16)$$

$$\sum_{i \in \mathcal{A}} wsize_i \cdot na_i \leq \mathcal{D} \quad (17)$$

$$\sum_{i \in \mathcal{A}} wtime_i \cdot na_i \leq WT_UB \quad (18)$$

$$\sum_{i \in \mathcal{A}} rtime_i \cdot na_i + \sum_{i \in \mathcal{A}} atime_i \cdot na_i \leq AT_UB \quad (19)$$

This ensures that we maximize the frequency and number of the most important analyses. The decision variables are feasibility of an analysis, $execute_i$, and its frequency na_i . Equations 16–19 show the various constraints that we have considered, related to the total execution time and the available storage space. Equation 16 specifies the bound on the number of executions of the i th analysis. $total_importance$ is the sum of importance (imp_i) of all analyses in \mathcal{A} . The multiplier of $execute_i$ denotes the available storage space for i th analysis proportional to its importance imp_i .

The sum of output sizes of all feasible analyses should be less than the total available storage space to avoid storage overflow (see Eq. 17). The bandwidth of NVRAM is higher but the available space on NVRAM may be limited, whereas the hard disk drive may have lower write bandwidth but the available space may be higher. Therefore we consider both the total storage space as well as the time to write to the storage. Equation 18 specifies an upper bound on the write time. Equation 19 specifies the analysis time constraint, which includes the time to read from storage and analysis computation time. The threshold for this, AT_UB , can be decided based on resource availability. We have not considered the memory constraint in

Table 4 Analyses for simulation of water and ions in LAMMPS

Analysis	Description
rdf	Compute hydronium-water, hydronium-hydronium, hydronium-ion (A1) and ion-water, ion-ion (A2) RDFs averaged over all molecules
vacf	Compute velocity auto-correlation function for the water-oxygen, hydronium-oxygen, and ion atoms (A3)
msd	Compute mean squared displacements averaged over all hydronium and ions (A4)
histo	histogram of atom positions and velocities (A5)
fft	1D fast fourier transform of atom positions and velocities (A6)

this mode because the number of analysis cores can be chosen based on the memory requirement of the most memory-intensive analysis. Next, we demonstrate the utility of the formulations through experimental results.

5 Experimental Evaluations

We evaluate using two codes—(1) the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) classical molecular dynamics simulation code [10, 15] and (2) the FLASH multiphysics multiscale simulation code [8]. The LAMMPS problem investigated is a box of water molecules solvating two types of ions. Table 4 lists the analyses investigated for this problem. These are representative analyses for molecular dynamics simulation—radial distribution functions (rdf), mean square displacements (msd) of molecules/ions, velocity auto-correlation functions (vacf), histograms (histo) and fast fourier transforms (fft) which is a temporal analysis. Combined, these physical observables provide key information on understanding the structure and dynamics of liquids and materials [3]. Additionally, their respective algorithms (e.g. accumulating histograms, computing time averages, evaluating correlation functions) are representative of those employed in the calculation of a large class of physical observables. We ran the FLASH simulation for the Sedov problem using three dimensions with 16^3 cells per block. Each block consists of 10 mesh variables and we can vary the problem size by adjusting the global number of blocks to simulate larger domain sizes. We perform three different analyses for FLASH—(1) vorticity, (2) L1 error norm for density and pressure and (3) L2 error norm for x, y, z velocity variables.

The first optimization model is implemented in the GAMS [5] modeling language and solved using the CPLEX 12.6.1 solver. We used the AMPL [7] modeling language for the second and third formulations. This was solved using the MINLP solver which implements branch and bound method. Maximum solve time for all problem configurations was 0.3 s. The recommended parameters (output of the solver) is used to run the experiments. The execution times were required by the solver as input

parameters, and were empirically determined by running a small number of time steps on few process counts followed by interpolation [12]. Memory estimates were obtained through application profiling (e.g. MPI timers and IBM profiler HPCT). We show the results of experimental runs from the IBM Blue Gene/Q system, Mira, at Argonne Leadership Computing Facility, Argonne National Laboratory [1, 2].

5.1 Results for Mode 1 (Same Job, Space-Division)

5.1.1 Efficacy of In Situ Scheduling

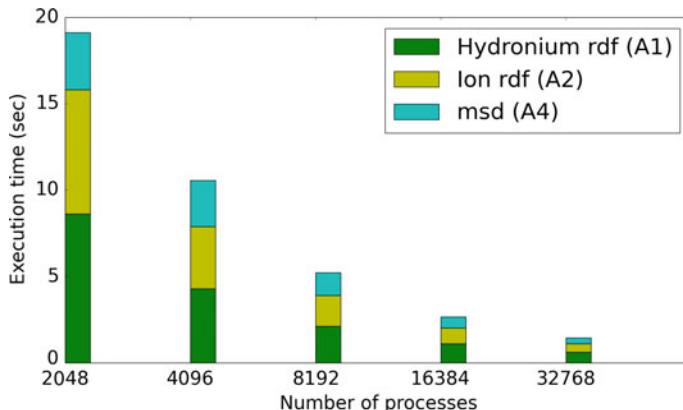
We demonstrate the utility of the optimization approach to allow the user to specify a threshold on the additional time allocated for in situ analyses, which is very important while requesting for resources on a supercomputer. In Table 5, we show results for varying thresholds from 20% to 1% (column 1), specified as a percentage of the simulation time. Columns 2–5 show the feasibilities of the analyses hydro-nium rdf (A1), ion rdf (A2), vacf (A3) and msd (A4) during a simulation of 1000 steps within the specified time thresholds. The solver recommends that A1, A2, A3 may be performed 10 times, i.e. the minimum analysis interval is 100. Note that the optimization model recommends fewer A4 analysis with decreasing threshold. This is because A4 requires higher analysis execution time and output time as well as more memory. Column 6 depicts the total execution time actually taken by the in situ analyses based on this solution. The ratio (%) of the actual execution time for the in situ analyses with respect to the specified threshold limit is shown in the last column. We observe that the total in situ analyses time is always within the specified threshold. Higher threshold (5–20%) allows more time and hence the solver recommends higher number of in situ analyses. The solver does not recommend performing A4 in situ at a threshold of 1% as the execution requirements of A4 exceed the threshold limit. Thus, the optimization model recommends optimal parameters (analyses frequencies) for a given set of application and system parameters. The model accounts for user-specified thresholds and proposes an optimal schedule for the desired in situ analyses. We expect this to play an important role as scientists start adopting in situ analysis.

5.1.2 Efficacy of In Situ Scheduling for Moldable Jobs

Moldable jobs are typically strong scaling jobs accommodated by job schedulers to improve the system utilization. Figure 3 shows strong scaling results on 2048–32768 Mira cores for three analyses—A1, A2 and A4 (see Table 4) in the 100 million atom LAMMPS water+ions simulation. Figure 3 is reprinted from our previous work [12]. The simulation times per time step are 4.16, 2.12, 1.08, 0.61 and 0.4 s on 2048, 4096, 8192, 16384 and 32768 cores of Mira respectively. The stacked bars illustrate the per-analysis execution times (y-axis) in a 1000-step simulation at each core count

Table 5 Threshold (%) and analyses frequencies of 4 analyses for 100 million-atom simulation in LAMMPS on 16384 cores of Mira

Threshold % (time in sec)	A1	A2	A3	A4	Analyses time (s)	% within threshold
20 (129.35)	10	10	10	4	103.47	80
10 (64.69)	10	10	10	2	52.79	81.6
5 (32.34)	10	10	10	1	27.45	84.87
1 (6.46)	10	10	10	0	2.11	32.66

**Fig. 3** Strong scaling (analyses) on 2048–32768 Mira cores in 100 million atom LAMMPS simulation. This image is reprinted from our previous work [12]

on x-axis. These are based on solver recommendations. Since the simulation time decreases with increasing core count, the allowed time for analysis decreases when the threshold is specified as a percentage of simulation time (10% in this case). The solver recommends a frequency of 10 for A1 and A2 in all cases, whereas the recommended frequency of A4 decreases from 10 on 2048 cores to 1 on 32768 cores. This is because msd (A4) does not scale well. However, A1 and A2 can be performed at a higher frequency even at higher core counts due to their scalabilities. Thus, our proposed optimization model effectively determines in situ analyses schedule based on core counts.

5.1.3 Impact of the Importance of Analyses

Table 6 shows the solver recommendations for three in situ analyses frequencies in FLASH simulation—vorticity (F1), L1 error norm (F2) and L2 error norm (F3) for two different sets of importance values. The simulation and analysis alternate on 16384 cores of Mira. The user-specified threshold on the analyses time was 5%

Table 6 Analyses frequencies in FLASH Sedov simulation on 16384 cores of Mira for different importance values of the analyses

	Vorticity (F1)	L1 error norm (F2)	L2 error norm (F3)
Importance (I1)	1	1	1
Frequency	1	10	10
Importance (I2)	2	1	2
Frequency	5	0	10

of the simulation time, which was 0.87 s on an average. This implies that the user allows a maximum execution time of 43.5 s for all the analyses during a 1000-step simulation (870 s). Importance (weight) of each analysis are shown in first and third rows. A higher importance value implies higher priority of the analysis. F1 being more compute intensive than F2 and F3 is recommended only once to maximize the total number of analyses because the importance of all analyses is 1. If the user prefers F1 and F3 over F2 and increases the weights of F1 and F3, the optimization model solution results in a higher frequency of F1 and F3 (fourth row), based on the user's preference. The frequency of F3 is more than F1 because F3 consumes lesser time and memory than F1. Thus our formulation gives immense flexibility to a user to choose the optimal number of analyses based on their importance within resource constraints.

5.2 Results for Mode 2 (Same Job, Time-Division)

In this section, we elaborate solutions from the linear program for second mode of in situ analysis (Sect. 4). Table 7 shows the feasibility results for 6 million atom 1000-step LAMMPS simulation on 3960 Mira nodes with 4 ranks per node (15,840 processes) and 4 OpenMP threads per process. The simulation, being more compute intensive, is run on 15,360 processes and all the analyses are consecutively run on 480 processes. The simulation time per time step was 0.06 s. The analyses have distinct time and memory characteristics (Fig. 1). Their transfer times, analysis times, memory requirements, and importance values are input to the optimization model, which determines the feasibilities of the analyses executions. We set the weight w of the second objective term in Eq. 10 to 0.1 and 0.01 (column 1). w may be selected based on the user's preference between maximizing the number of important analyses and minimizing the interval between two successive analysis steps. The minimum interval, n_LB was 20 (refer Sect. 4). The allowed transfer time overhead, xf_er_UB , was 1% of simulation time. The solution to the linear program are shown in columns 2–6 (Y implies $execute_i = 1$, i.e. the analysis can be feasibly executed) and the last column (the analysis interval, n). The linear program recommends execution of all analyses at an interval of 79 when lower preference ($w = 0.01$) is given to the analysis

Table 7 Analyses feasibilities for 6 million-atom LAMMPS simulation on 15,840 processes on Mira

w	rdf	msd	msd1d	msd2d	vacf	Interval
0.1	Y	N	Y	Y	N	27
0.01	Y	Y	Y	Y	Y	79

interval between two analysis steps. The analysis compute times for rdf, msd, msd1d, msd2d and vacf on 480 processes are 0.49, 2.21, 0.49, 0.61, 0.91 s respectively. When higher preference is given to minimizing analysis interval, i.e. $w = 0.1$, the linear program does not recommend in situ execution of msd and vacf due to their higher compute times. Thus our model is useful in systematically determining the analyses feasibilities based on user's preferences and resource parameters.

5.3 Results for Mode 3 (Different Jobs)

In this section, we present results to evaluate the usefulness of our formulation when simulation and analysis execute as separate jobs on same/different clusters. Table 8 shows the solution from our linear program when the simulation and analysis run as separate jobs, and the data is shared via a common 240 GBps GPFS filesystem with total available storage of 4 TB. The 500 million atom molecular dynamics simulation for 1000 time steps ran on 1024 nodes (with 4 ranks/node) of Mira. The feasible analyses were simultaneously done on 512 nodes (4 ranks/node) of Mira as a separate job. The importance of all analyses was assumed to be 1. The lower bound on the analysis interval was 50 time steps. The simulation time for 1 time step was 0.75 s. The first column of the table shows varying write time threshold WT_UB . The second column shows varying analysis time threshold AT_UB . These thresholds are specified as percentage of the simulation time. Columns 3–8 show the frequency of each analysis as recommended by the solver (see Sect. 4.3). This is also the frequency at which the simulation writes output data for each feasible analyses. The total number of feasible analyses steps increases from 26 to 84 with increase in the allowed time limit for output and analysis. Note that msd analysis requires higher compute times and the temporal analysis, fft, incur higher output times. Thus these are not recommended at lower allowed thresholds (first two rows). Our model helps the user to determine the feasible analysis frequencies based on the thresholds (that can be varied) and resource constraints.

We next show the results for variation in available storage space D and the time thresholds WT_UB and AT_UB . The various analyses feasibilities of a billion atom molecular dynamics simulation of 1000 time steps are shown in Table 9. The simulation ran on 1024 nodes (4 ranks/node) of Mira, where execution time for 1 simulation time step is 1.48 s. We varied WT_UB and AT_UB from 250 to 1000 s. Columns 2–7 show the recommended analysis frequency, with 0 implying that it is infeasible to

Table 8 Number of times each analysis can be performed within constraints. Analysis runs on 2048 processes and simulation runs on 4096 processes on Mira. Total number of feasible analyses shown in the last column

WT_UB (%)	AT_UB (%)	vacf	msd	histo position	histo velocity	fft position	fft velocity	Total #analyses
5	10	7	0	0	19	0	0	26
10	10	20	0	7	19	0	0	46
20	20	20	20	20	20	3	1	84

Table 9 Frequency of each analysis for simulation of 1 billion atoms on 4096 processes of Mira

Available storage (TB)	vacf	msd	histox	histov	fftx	fftv	Total #analyses
<i>WT_UB = 250 s, AT_UB = 250 s</i>							
1	1	1	19	20	0	0	41
10	1	1	19	20	0	0	41
<i>WT_UB = 500 s, AT_UB = 500 s</i>							
1	6	6	20	20	5	1	58
10	1	18	20	20	1	1	60
<i>WT_UB = 1000 s, AT_UB = 1000 s</i>							
1	6	5	20	20	20	11	82
10	20	20	20	20	13	1	94

perform within the given constraints. The histogram computations (histox and histov) can be feasibly executed in all cases. This is because they have lower resource requirements. We observe that the total number of feasible analyses (last column) increases from 41 to 94 with increase in available storage (column 1) and time thresholds. In the first two rows ($WT_UB = 250\text{ s}$, $AT_UB = 250\text{ s}$), the total number of analyses does not increase with increase in storage space. This implies that the main constraints were the time thresholds. When WT_UB and AT_UB are 1000 s , the total number of analyses increase with increase in available storage space from 1 to 10 TB, which implies space is also a constraint. However, note that the number of fft analyses is decreased to allow execution of vacf and msd more frequently to satisfy the maximization criteria of our optimization problem. This is because the importance was set to 1 for all analyses.

The solver recommends that vacf and msd can be feasibly executed once and 18 times respectively (fourth row). This is one of the configurations that can be feasibly computed within the time thresholds. There may be other feasible configurations because there may be multiple optimal solutions as is the case typically in linear programming. The user can attach higher importance value to an analysis if it is more important than the rest, the user may also order the important analyses using the importance values. For example, when the importance of vacf is increased from

1 to 2, keeping the importance of the rest at 1, the model recommends a solution of 20, 2, 20, 20, 0, 0 for the 6 analyses respectively. However, note that the objective function will be optimized within all the specified constraints related to time, memory and storage. Thus, the important analyses will be recommended only if they can be feasibly performed. The importance of an analysis depends on the application, the science requirement and the user's preference. Our model can effectively help the user determine the feasible frequencies for various selections of importance values as per the user's choice. This demonstrates the utility of our formulation to decide the optimal execution frequency of any number of desired analyses within resource constraints and user requirements. This is helpful to plan the large-scale science campaigns which are capable of producing huge volumes of data and hence it is imperative to carefully plan the in situ analysis executions.

The readers are referred to [11, 12] for more detailed experimental evaluation of the methodologies presented in this chapter.

6 Conclusions

In this chapter, we described our formulations for different in situ executions, namely tightly-coupled execution of simulation and analysis as the same job (mode 1), loosely-coupled simulation and analysis on different processes of the same job (mode 2) and loosely-coupled simulation and analysis as different jobs (mode 3). We considered the resource constraints such as available memory or storage, network bandwidth and the application requirements such as minimum analysis interval and importance of analysis. We also considered the execution times and memory to decide the feasibility of execution. Additionally, we accounted for transfer time, data size, output time, additional memory requirements and importance of analysis. There can be multiple (may be more than 10) different analysis needed for a simulation. Recommending the optimal number of analyses and their frequencies is an NP-hard problem. Thus it is inefficient and sub-optimal to manually solve, and a formulation like ours proves very valuable to the scientists. While we have shown the results for analysis computations, the same formulations can be used for in situ visualizations.

This formulation is especially practical and desirable at exascale when the speed of producing data will overwhelm the speed of consuming data. Hence it is extremely crucial to carefully plan the rate at which the simulation produces data, when and where can the analysis be feasibly executed without stalling the simulation, or without taking too long for data transfer. These questions will be more relevant when exascale machines arrive, due to the increasing gap between the compute rates and the network and I/O speeds. The three modes presented in this chapter are subject to the resource availability and constraints, the application code, and the user's preference. Given the comprehensive knowledge of system parameters in these three cases, our models can help determine the feasibilities of various in situ analysis modes a priori to the production runs.

Acknowledgements This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-06CH11357. This research used resources of the NERSC supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-05CH11231. This work was supported in part by the DOE Office of Science, ASCR, under award numbers 57L38, 57L32, 57L11, 57K50, and 5080500.

References

1. Argonne Leadership Computing Facility's Supercomputer Mira. <http://www.alcf.anl.gov/mira>
2. Computing Resources at Argonne Leadership Computing Facility. <https://www.alcf.anl.gov/computing-resources>
3. Allen, M.P., Tildesley, D.J.: Computer Simulation of Liquids. Oxford Science Publications (1989)
4. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
5. Brooke, A., Kendrick, D., Meeraus, A.: GAMS: A User's Guide. The Scientific Press, South San Francisco, California (1988)
6. Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level and Safety with Optimization Algorithms. Multi-objective Optimization, pp. 11–36. Springer London, London (2008)
7. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2 edn. Duxbury Press (2003)
8. Fryxell, B., Olson, K., Ricker, P., Timmes, F.X., Zingale, M., Lamb, D.Q., MacNeice, P., Rosner, R., Truran, J.W., Tufo, H.: FLASH: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophys. J. Suppl.* **131**, 273–334 (2000)
9. Gerber, R., Allcock, W., Beggio, C., Campbell, S., Cherry, A., Cholia, S., Dart, E., England, C., Fahey, T., Foerster, F., Goldstone, R., Hick, J., Karelitz, D., Kelly, K., Monroe, L., Prabhat, Skinner, D., White, J.: DOE High Performance Computing Operational Review (HPCOR): Enabling Data-Driven Scientific Discovery at HPC Facilities. Technical report, Berkeley, CA (US) (2014)
10. LAMMPS Molecular Dynamics Simulator. <http://lammps.sandia.gov>
11. Malakar, P., Vishwanath, V., Knight, C., Munson, T., Papka, M.E.: Optimal execution of co-analysis for large-scale molecular dynamics simulations. In: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 702–715 (2016)
12. Malakar, P., Vishwanath, V., Munson, T., Knight, C., Hereld, M., Leyffer, S., Papka, M.E.: Optimal scheduling of in-situ analysis for large-scale scientific simulations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2015)
13. Mittal, S., Vetter, J.S.: A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Trans. Parallel Distrib Syst (TPDS)* (2015)
14. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley, New York, NY (1988)
15. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **117**(1), 1–19 (1995)

Tools

Leveraging Production Visualization Tools In Situ



Kenneth Moreland, Andrew C. Bauer, Berk Geveci, Patrick O’Leary, and Brad Whitlock

Abstract The visualization community has invested decades of research and development into producing large-scale production visualization tools. Although *in situ* is a paradigm shift for large-scale visualization, much of the same algorithms and operations apply regardless of whether the visualization is run post hoc or *in situ*. Thus, there is a great benefit to taking the large-scale code originally designed for post hoc use and leveraging it for use *in situ*. This chapter describes two *in situ* libraries, Libsim and Catalyst, that are based on mature visualization tools, VisIt and ParaView, respectively. Because they are based on fully-featured visualization packages, they each provide a wealth of features. For each of these systems we outline how the simulation and visualization software are coupled, what the runtime behavior and communication between these components are, and how the underlying implementation works. We also provide use cases demonstrating the systems in action. Both of these *in situ* libraries, as well as the underlying products they are based on, are made freely available as open-source products. The overviews in this chapter provide a toehold to the practical application of *in situ* visualization.

K. Moreland (✉)
Sandia National Laboratories, Albuquerque, NM, USA
e-mail: kmorel@sandia.gov

A. C. Bauer
United States Army Corps of Engineers, Washington, DC, USA
e-mail: andrew.c.bauer9.civ@mail.mil

B. Geveci · P. O’Leary
Kitware, Inc., Clifton Park, NY, USA
e-mail: berk.geveci@kitware.com

P. O’Leary
e-mail: patrick.oleary@kitware.com

B. Whitlock
Intelligent Light, Rutherford, NJ, USA
e-mail: bjw@ilight.com

1 Introduction

Although *in situ* is a paradigm shift for large-scale visualization, much of the same algorithms and operations apply regardless of whether the visualization is run post hoc or *in situ*. Thus, there is a great benefit to taking the large-scale code originally designed for post hoc use and leveraging it for use *in situ*. Two of the most popular post hoc visualization tools are VisIt [9] and ParaView [2]. Contributing to the success of these tools is that they each are feature rich, have proven parallel scalability, have automated scripting capabilities, are free, and have a large development community. To leverage these capabilities for an *in situ* environment, each tool now provides a library that allows data and control to pass from another software tool. VisIt provides a library named Libsim [32], and ParaView provides a library named Catalyst [5]. In this chapter we review these libraries and demonstrate how they are used to implement *in situ* visualization.

The introduction of this book lists many important features of *in situ* visualization that motivate the implementation and use of Libsim and Catalyst. However, the introduction also lists several limitations of *in situ* visualization that do not apply to post hoc visualization. The upshot is that for the foreseeable future both *in situ* and post hoc visualization will be important for discovery at large computing scales, and so providing both types of visualization is important. Because Libsim and Catalyst each derive functionality from their respective classic tools, they immediately make available both *in situ* and post hoc visualization. Furthermore, visualizations made post hoc are easily made *in situ* and vice versa.

We present Libsim and Catalyst together in this chapter because there are many common features the two libraries share. The two systems share the same *in situ* taxonomy described in the introduction.

Integration Type Both tools are general purpose and designed to work well with a variety of simulation codes. However, their primary function is specific to visualization and the simulation must be modified to use the library.

Proximity Libsim and Catalyst assume they are running in close proximity using the same resources as the simulation.

Access Because Libsim and Catalyst are libraries that share the same memory space as the simulation, it is possible for these codes to directly access the simulation's memory. However, they only access memory specifically given to them, and the data must be in a specified format.

Division of Execution Libsim and Catalyst use time division to alternate use of the simulation's resources.

Operation Controls The main mode of operation is to perform visualizations according to a predefined batch script. However, both Libsim and Catalyst are capable of performing human-in-the-loop visualization by attaching a remote GUI to a running simulation.

Output Type Libsim and Catalyst are each capable of producing a wide variety of outputs. Images and image databases [3] are common outputs, but derived geometric structures and statistics are also possible data products.

In addition to having similar properties, Libsim and Catalyst share similar methods to interface with simulations, to specify what visualization operations to perform, and to instantiate the visualization operation. Both Libsim and Catalyst are interfaced to a simulation by writing an “adapter.” The adapter is primarily responsible for converting the data representation used by the simulation to the data representation used by Libsim and Catalyst. Both Libsim and Catalyst use VTK [25] as their underlying implementation, and thus the adapter for either must convert the simulation’s data format to VTK’s data format. VTK can reference data in external arrays, so often the adaption of a simulation’s data structures to VTK’s data structures can be done without copying the data.

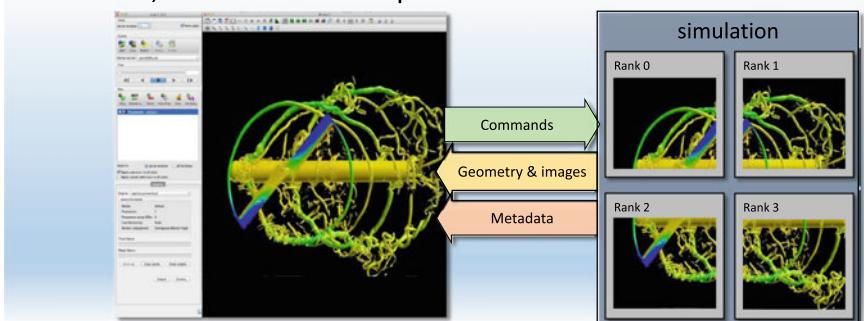
Also similar among the two libraries is their runtime behavior. Each allows the simulation to operate in its own execution loop. At the simulation’s discretion, it periodically invokes Libsim or Catalyst with an updated collection of data. Under typical batch operation, the library processes the data, saves whatever visualization product is generated, and returns control back to the simulation. Both libraries also support a mode in which a live, remote GUI is updated. In this mode control can either be immediately returned to the simulation, or the simulation may be blocked while a remote user interactively explores the data, which is particularly useful for debugging the simulation.

The following two sections provide details for Libsim and Catalyst. Each section describes how the respective library is integrated with a simulation, how the library behaves at runtime, and the underlying implementation of the library. Because of their similarity there is redundancy in these descriptions. For clarity, we have repeated descriptions in each section to provide a thorough explanation of each.

2 Libsim

Libsim [33] is a library that enables in situ visualization using VisIt [9], a massively parallel visualization and data analysis tool built on VTK. VisIt contains a rich set of data readers, operators, and plots. These features read, filter or transform data, and ultimately provide a visual representation of the data to allow for exploration and analysis. Many of these features can be chained together to build pipelines that create sophisticated visualizations. Libsim satisfies multiple use cases, shown in Fig. 1. Libsim was conceived originally as an online visualization mechanism for debugging simulations with the aid of the VisIt GUI. Over time, Libsim evolved to allow both interactive and batch uses cases that allows it to generate a host of data products without a user in the loop. Today, virtually anything that is possible in the VisIt GUI is also possible from Libsim. This flexibility has enabled Libsim to be integrated into diverse simulations related to fields of study such as Computational Fluid Dynamics (CFD) or Cosmology. Libsim is highly scalable and has been run at levels of concurrency surpassing 130K cores.

Interactive, Human in the Loop



Batch, Automated

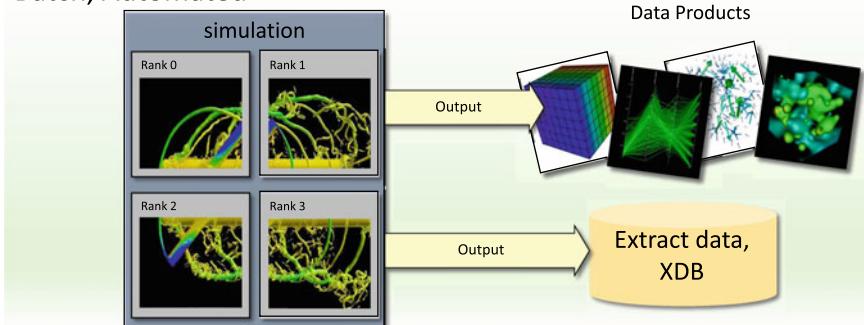


Fig. 1 Libsim supports interactive and batch use cases

2.1 Integration with Simulation

Libsim integrates with applications as a set of library calls that are usually encapsulated into a module called a data adaptor (depicted in Fig. 2). Libsim provides C, FORTRAN, and Python bindings to minimize the amount of cross-language programming that is asked of application scientists. Libsim provides a relatively low-level application programming interface (API) so it can be integrated flexibly into host simulations. Libsim can be used directly, or it can be used within other infrastructures that integrate into the simulation, such SENSEI [4] or Damaris [10]. The typical procedure for instrumenting a simulation with Libsim involves writing a data adaptor and proceeding through four stages: initialization, exposing data, iteration, and adding user interface. During initialization, the simulation sets up the relevant environment and calls functions to either prepare for interactive connections or for batch operations. Writing the data adaptor involves exposing simulation data to Libsim. The next stages are optional. Iteration involves adding code that will produce any plots or data extracts. The final stage adds a user interface and registers simulation functions to respond to user-interaction via the VisIt GUI.

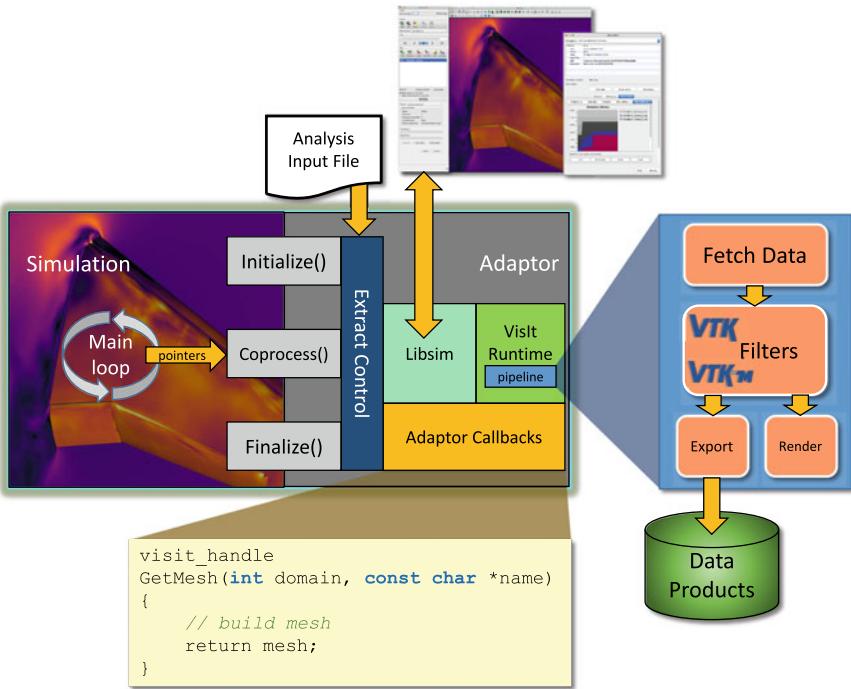


Fig. 2 Simulations instrumented with Libsim link to the Libsim library. The simulation supplies data adaptor functions that expose data to VisIt pipelines. The VisIt pipelines can supply a running VisIt instance with data or produce in situ data products

The Libsim API can be thought of as having two components: a control interface and a data interface. The control interface is responsible for setting up environment, event handling, and registering data callback functions. The data interface is responsible for annotating simulation memory and packaging related data arrays into mesh data structures that can be used as inputs to VisIt. Data arrays are passed by pointer, allowing both zero-copy access to simulation data and transfer of array ownership to VisIt so data can be freed when no longer needed. Arrays can be contiguous in memory as in structure-of-array (SOA) data layouts or they can use combinations of strides and offsets to access simulation data as in array-of-structures (AOS) data structures. Libsim supports commonly used mesh types, including rectilinear, curvilinear, Adaptive Mesh Refinement (AMR), and unstructured grids consisting of finite element cell types. Libsim also can also support computational domains that are not actually meshes such as Constructive Solid Geometry (CSG). However data are represented, Libsim usually relies on the simulation's data decomposition when exposing data to VisIt, and the simulation can expose multiple meshes with their own domain decompositions. Libsim permits simulations to add field data on the mesh centered on the cells or on the points. Field data consists of scalars, vectors, tensors,

labels, and arrays with an arbitrary number of tuples per element. Libsim includes additional data model concepts, allowing simulations to specify domain adjacency, ghost data, mixed material cells, and material species.

During the instrumentation process, a decision must be made whether to support interactive connections or batch operations via Libsim, or both. The paths differ somewhat, though in both cases there are some upfront calls that can be made to set up the environment for Libsim. This consists of VisIt's environment and the parallel environment. When interactive connections are expected, Libsim will write a small `.sim2` file containing network connection information that VisIt can use to initiate a connection to the simulation. This file is not needed for batch-only operation. The following code example includes the Libsim header files, sets up Libsim for parallel operation, discovers environment settings needed to load VisIt runtime libraries, and finally creates the `.sim2` file needed for interactive connections.

```
#include <VisItControlInterface_V2.h>
#include <VisItDataInterface_V2.h>

/* Broadcast callbacks */
static int
bcast_int(int *value, int sender, void *cbdata)
{
    return MPI_Bcast(value, 1, MPI_INT, sender, MPI_COMM_WORLD);
}
static int
bcast_string(char *str, int len, int sender, void *cbdata)
{
    return MPI_Bcast(str, len, MPI_CHAR, sender, MPI_COMM_WORLD);
}
void libsim_initialize(int interactive)
{
    /* Parallel setup */
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    VisItSetBroadcastIntFunction2(bcast_int, NULL);
    VisItSetBroadcastStringFunction2(bcast_string, NULL);
    VisItSetParallel(size > 1);
    VisItSetParallelRank(rank);
    /* Get VisIt environment */
    char *env = NULL;
    if(rank == 0)
        env = VisItGetEnvironment();
    VisItSetupEnvironment2(env);
    if(env != NULL)
        free(env);
    if(rank == 0 && interactive)
    {
        /* Write out .sim2 file that VisIt uses to connect. */
        VisItInitializeSocketAndDumpSimFile(
            "simulation_name"
            "Comment about the simulation",
            "/path/to/where/sim/was/started",
            NULL, NULL, "simulation_name.sim2");
    }
}
```

When integrating Libsim for interactive operation, calls to the control interface to handle events must be inserted into the simulation. Libsim provides the `VisItDetectInput()` function for this purpose. It listens for connections from

a VisIt client. Simulations can build event loops using the `VisItDetectInput` function or call it in a polling manner from their own event loops. When a connection request is detected, the function will return a value indicating that other Libsim functions must be called to complete the connection request and load the runtime library. Once the runtime library is loaded, the developer may register data callback functions that expose simulation data as Libsim objects. Data callback functions are called by VisIt's runtime library to inquire about simulation metadata and when specific meshes and fields are needed in order to create a specific data product. Data callbacks must be installed once the VisIt runtime library has been loaded. In a batch-style integration, this can occur immediately after the call to `VisItInitializeRuntime()`, whereas for interactive use, the data callbacks must be installed after a successful call to `VisItAttemptToCompleteConnection()`, which signifies a successful connection of VisIt's viewer application to the simulation.

```

static void
libsim_bcast_cmd_cb(int *command, void *cbdata)
{
    MPI_Bcast(command, 1, MPI_INT, 0, MPI_COMM_WORLD);
}

static void
libsim_control_cb(
    const char *cmd, const char *args, void *cbdata)
{
    /* Optional: Respond to text commands */
}

/* MetaData and Mesh callbacks shown later...*/

static void libsim_setup_callbacks(void)
{
    void *cbdata = /* Point this at application data */;
    VisitSetCommandCallback(libsim_cmd_cb, cbdata);
    VisitSetSlaveProcessCallback2(libsim_bcast_cmd_cb, cbdata);
    VisitSetGetMetaDataTable(libsim_metadata_cb, cbdata);
    VisitSetGetMesh(libsim_mesh_cb, cbdata);
}

/* Simplified example - invoked by the simulation. */
void libsim_interactive(void)
{
    switch(VisItDetectInput(blocking, -1))
    {
        case 0:
            /* No input from VisIt, return control to sim. */
            break;
        case 1:
            /* VisIt is trying to connect to sim. */
            if(VisItAttemptToCompleteConnection() == VISIT_OKAY)
                libsim_setup_callbacks();
            break;
        case 2:
            /* VisIt wants to tell the engine something. */
            if(!VisitProcessEngineCommand())
                VisitDisconnect();
            break;
    }
}

```

The deferred nature of Libsim data requests ensures that the simulation does not have to waste time computing results that might not be used, as when computing derived fields for visualization. Data requests are assembled inside of the VisIt runtime libraries from its execution contract, which includes a manifest of all of the data needed to create a visualization. Libsim's callback function design enables the VisIt runtime library to request data on demand from the Libsim adaptor in the simulation. Data are requested in stages, first metadata is obtained to inform the VisIt runtime about the meshes and variables provided by the simulation. Simulations can expose as little data or as much data as they like. The callback functions include a user-defined data argument that allows application data to be associated with callbacks when they are registered in order to make it easier to access application data from callbacks when they are invoked by the VisIt runtime library.

```

visit_handle
libsim_metadata_cb(void *cbdata)
{
    visit_handle md = VISIT_INVALID_HANDLE,
                  mmd = VISIT_INVALID_HANDLE;
    /* Create metadata. */
    if(VisIt_SimulationMetaDataTable_alloc(&md) == VISIT_OKAY)
    {
        /* Access application data */
        application_data *app = (application_data *)cbdata;

        /* Set the simulation state. */
        VisIt_SimulationMetaDataTable_setMode(md,
                                              VISIT_SIMMODE_RUNNING);
        VisIt_SimulationMetaDataTable_setCycleTime(md, app->cycle,
                                                    app->time);

        /* Add mesh metadata. */
        if(VisIt_MeshMetaDataTable_alloc(&mmd) == VISIT_OKAY)
        {
            /* Set the mesh's properties.*/
            VisIt_MeshMetaDataTable_setName(mmd, "mesh");
            VisIt_MeshMetaDataTable_setMeshType(
                mmd, VISIT_MESH_TYPE_RECTILINEAR);
            VisIt_MeshMetaDataTable_setTopologicalDimension(mmd, 3);
            VisIt_MeshMetaDataTable_setSpatialDimension(mmd, 3);
            VisIt_MeshMetaDataTable_setNumDomains(
                mmd, app->total_num_domains);
            VisIt_SimulationMetaDataTable_addMesh(md, mmd);
        }

        /* We could expose more meshes, variables, etc. */
    }
    return md;
}

```

Once the data requirements are determined for a visualization, Libsim invokes the registered mesh callback to obtain mesh data on a per-domain basis. Libsim is flexible and can represent several mesh types. Meshes, as with most Libsim data constructs, are constructed from arrays. Libsim provides functions that enable simulation data arrays to be annotated with size, type, offset, and stride information so arrays can be passed back to VisIt to be used zero-copy as much as possible. In addition, simulation callback functions can wrap temporary memory that VisIt is allowed to free in case zero-copy representations are not feasible. Once the mesh callback has

been executed, variables and then other types of data are requested, each from their respective callback function. As a simulation adaptor grows more complete, additional callbacks can be registered to support variables, materials, AMR nesting, mesh decompositions, etc.

```
visit_handle
libsim_mesh_cb(int domain, const char *name, void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;
    if(VisIt_RectilinearMesh_alloc(&h) != VISIT_ERROR)
    {
        visit_handle hx, hy, hz;
        /* Access application data */
        application_data *app = (application_data *)cbdata;
        VisIt_VariableData_alloc(&hx);
        VisIt_VariableData_alloc(&hy);
        VisIt_VariableData_alloc(&hz);
        VisIt_VariableData_setDataD(hx, VISIT_OWNER_SIM, 1,
                                   app->dims[0], app->xc);
        VisIt_VariableData_setDataD(hy, VISIT_OWNER_SIM, 1,
                                   app->dims[1], app->yc);
        VisIt_VariableData_setDataD(hz, VISIT_OWNER_SIM, 1,
                                   app->dims[2], app->zc);
        VisIt_RectilinearMesh_setCoordsXYZ(h, hx, hy, hz);
    }
    return h;
}
```

With simulations able to produce an ever increasing amount of data, Libsim's emphasis gradually shifted from being a tool for debugging simulation codes towards production of data products without massive amounts of I/O. To generate data products, the simulation can call Libsim functions to set up VisIt plots and VisIt operators and to set their attributes before saving images or exporting processed datasets. These operations can also be set up via a VisIt session file rather than relying on fixed sets of plots. This allows the user to connect using VisIt interactively to set up the desired visualization, save the configuration to a session file, and then apply the recipe in batch to produce movies and other data products.

```
/* Save plots designated by a session file. */
VisItRestoreSession("setup.session");
VisItSaveWindow("a0000.png", 1024, 1024, VISIT_IMAGEFORMAT_PNG);

/* Set up some plots directly */
VisItAddPlot("Mesh", "mesh");
VisItAddPlot("Pseudocolor", "pressure");
VisItDrawPlots();
VisItSaveWindow("a0001.png", 1048, 1024, VISIT_IMAGEFORMAT_PNG);
```

Libsim has been used increasingly with CFD codes with common needs for producing lightweight surface-based data extracts that can be explored using desktop visualization tools. Surface data extracts often consist of slices, isosurfaces, or boundary surfaces plus field data. Generating such extracts in situ results in a drastic reduction in saved data and time needed compared to extracting such data from bulk volume data during post-processing. To permit general surface extracts to be specified via an external configuration file and simplify multiple aspects of instrumenting codes using Libsim (particularly for parallel event loops), we have created a companion

library called “extract control”. Extract control enables multiple extract types (e.g., surfaces, images, or Cinema databases [3]) to be requested via a convenient YAML file that the user can change, as opposed to direct Libsim function calls or using VisIt session files. The extract control library also encapsulates some of the usual boilerplate code needed to support interactive event loops as well as batch-style Libsim integrations, resulting in fewer lines of code.

Interactive instrumentation using Libsim allows the VisIt GUI to use the simulation as a normal compute engine, making it possible to do most kinds of analysis or data interrogation with large file-based datasets. Interactive simulations benefit from other features provided by VisIt and Libsim. For instance, Libsim provides functions that let the simulation provide sample data that can be aggregated into strip charts that plot quantities of interest over time. Strip charts can display arbitrary sample data, though time and memory measurements are commonly plotted. The VisIt GUI displays strip charts and other simulation state in the Simulation window. The simulation window also exposes controls published by the simulation. These controls take the form of command buttons in the simplest case that, when pressed, can invoke callback functions in the simulation adaptor. This allows the user to initiate actions in the simulation based on button clicks in the VisIt GUI. The VisIt GUI also allows for simulation-specific custom user interfaces. Custom user interfaces are designed using Qt Designer and the VisIt GUI can replicate such user interfaces as extensions within the Simulation window. Custom user interfaces enable the VisIt GUI to alter simulation parameters to affect more complicated steering actions. This feature was successfully used by Sanderson et al. [24] to create a customized simulation dashboard for the Uintah software suite.

2.2 Runtime Behavior

Libsim accepts control from the simulation and then enters an event loop or other batch-oriented code in the simulation adaptor to generate data extracts and immediately return. When Libsim’s operations complete, control is returned to the simulation. Libsim’s runtime behavior depends on how it was used to instrument the simulation, and its behavior varies between human-in-the-loop-blocking to nobody in the loop, non-blocking. The behavior for interactive use cases is determined by how the `VisItDetectInput()` function was called when instrumenting the simulation. The function can be used to implement blocking event loops or polling event loops that are invoked periodically from the simulation. Blocking calls return when commands have been received by the VisIt GUI and may result in additional calls that request user input. Blocking calls may also include a timeout that enables the function to return after a specified period of inactivity to return control to the simulation. Libsim includes other functions that can be called in conjunction with the event loop to notify VisIt’s runtime library of new simulation data so it can be used to push data to the VisIt GUI. This feature allows the VisIt GUI to connect to the running simulation and recompute its plots in response to updates from the simulation so the

user can watch the simulation evolve. Connecting to the running simulation, making plots, watching for a while, and then disconnecting is supported in simulations that use Libsim and this cycle can be repeated over the life of the simulation.

2.3 *Underlying Implementation*

VisIt functionality is divided into different processes, according to function. VisIt provides client programs such as the GUI so users can analyze data interactively. VisIt’s viewer acts as a central hub, which manages state, communication with other programs, and rendering data. VisIt’s compute “engine” reads data and executes any plot and operator pipelines to generate geometry or image data for the viewer. The compute engine can run locally or on other HPC systems via client/server mode. Libsim enables a simulation to act as a proxy for the VisIt compute engine. Libsim is actually separated into a front-end library and a runtime library. The front-end library is minimal and is linked to the simulation. The front-end library provides all of the user-facing functions such as event handling while providing an interface to runtime library functions that are loaded later once Libsim is actually told to do work. This separation allows simulations to link with Libsim once but dynamically change the version of VisIt used at runtime. An important function of the front end library is to write a `sim2` file, which is a file containing networking information that the VisIt GUI can use to initiate a socket connection to the simulation. Since VisIt relies on the `sim2` file for connecting to simulations, from a user’s point of view, accessing a running simulation is essentially the same as accessing any file-based dataset. Upon opening the `sim2` file, VisIt initiates a socket connection to the simulation and once a successful connection is made via the `VisItDetectInput()` function, the simulation dynamically loads the Libsim runtime library and calls additional data adaptor code to register data-producing callback functions with Libsim.

The Libsim runtime library incorporates parts of the VisIt engine and viewer so it can manage plots and execute visualization pipelines to deliver VisIt functionality. When plots are created, VisIt instantiates a visualization pipeline consisting of various data analysis filters. The pipeline uses a contract mechanism to build a list of data that is needed to produce the desired visualization. The list of data in the contract is used to request simulation data via a specialized database plug-in. Libsim’s database plugin invokes callback functions from the simulation data adaptor to obtain data rather than reading from files. A callback function is responsible for returning different types of data such as a mesh domain or a specific variable on that mesh domain. Data are returned by making Libsim function calls that package simulation data into Libsim objects. The Libsim objects ferry data from the simulation into the Libsim database reader where they are unpacked and used to assemble VTK datasets that VisIt can use internally in its visualization pipelines.

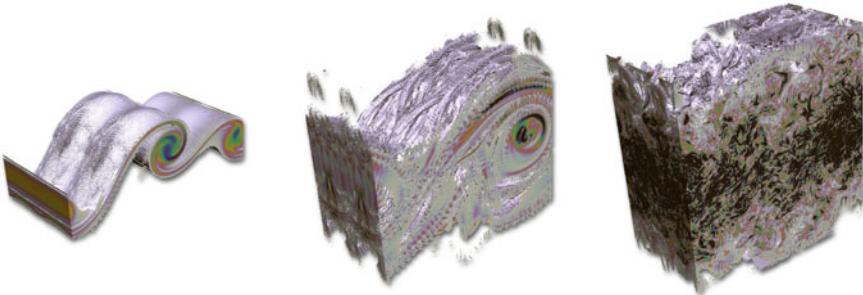


Fig. 3 The Evolution of Temporal Mixing Layer from Initial to Vortex Breakdown

2.4 Use Case

Libsim has been used in a variety of domains and applications. Libsim automatically produces batch data extracts (such as geometry, rendered images, or Cinema databases) at scale and lets VisIt connect to running simulations for interactive exploration, monitoring, and steering. In the CFD domain, Libsim has been used to generate in situ data extracts consisting of reduced sets of geometry suitable for post hoc data analysis of engineering datasets [11, 14]. Forsythe et al. [13] successfully used this approach in CREATE-AV Kestrel to generate geometric extracts to accelerate analysis of fully coupled high-fidelity simulations of a rotorcraft in a ship's airwake. Helicopters landing on sea-based landing platforms experience turbulent airflows resulting from their surroundings such as the ship airwake produced from air moving around the structures on naval ships. Airwakes produce nonlinear aerodynamic effects that must be taken into account in order to accurately simulate landing in such a setting, as in this flight simulator coupling. Libsim was integrated into Kestrel to produce geometric data extracts in FieldView XDB format representing 45 s in 2700 time steps, taken every 5 simulation time steps. The resulting extract-based simulation data are drastically smaller than a corresponding volume dataset, and the system “*rendered the animations in hours rather than the days it would have otherwise taken [23]*.”

In a larger computation, Libsim was used as an in situ analysis infrastructure coupled to the AVF-LESLIE [28, 29] combustion code by SENSEI. AVF-LESLIE was configured to simulate unsteady dynamics of a temporally evolving planar mixing layer at the interface between two fluids. This interface results in a type of fundamental flow that mimics the dynamics encountered when two fluid layers slide past one another and is found in atmospheric and ocean fluid dynamics as well as combustion and chemical processing. Visualizations of the flowfield in Fig. 3 show isosurfaces of the vorticity field, at 10,000, 100,000, and 200,000 time steps where the flow evolves from the initial flow field, vortex braids begin to form, wrap and then the flow breaks down leading to homogeneous turbulence, respectively.

AVF-LESLIE was statically linked to Libsim and VisIt and run on Titan at Oak Ridge Leadership Class Computing Facility on 131,072 cores. Static linking was selected because of an observation that Libsim's usual deferred loading of the VisIt runtime library incurred significant overhead when running at large scale on Titan. A SENSEI data adaptor was created for AVF-LESLIE. It passed structured mesh and field data from the main FORTRAN-based simulation through a C-language compatibility layer where data pointers were used to create VTK datasets that were passed into SENSEI. VTK datasets were exposed to Libsim inside SENSEI via an additional data adaptor that ultimately passed data to the VisIt runtime library to create data products. Two types of in situ computations were performed: a rendering workflow, and an extract-based workflow. The rendering workflow generated 1600×1600 pixel images of a vorticity isosurface and composited partial images into a single PNG image using tree-based compositing within VisIt's runtime library. The vorticity quantity was computed on demand in the SENSEI adaptor for AVF-LESLIE. The extract workflow saved the same isosurface to FieldView XDB files, aggregating geometry to smaller subgroups of 96 ranks to reduce file system contention. A typical extract from this dataset was approximately 200 times smaller than the full volume data, and this enabled the project to save data 20 times more frequently while remaining still 10 times smaller than when using volume data.

Libsim continues to provide in situ capabilities for frameworks and codes that need to run at large scale and want to leverage the capabilities in a fully-featured visualization tool such as VisIt.

3 Catalyst

The ParaView Catalyst library is a system that addresses challenges of in situ visualization and is designed to be easily integrated directly into large-scale simulation codes. Built on and designed to interoperate with the standard visualization toolkit VTK and scalable ParaView application, it enables simulations to perform analysis intelligently, generate relevant output data, and visualize results concurrent with a running simulation. The ability to concurrently visualize and analyze data from simulations is synonymous with in situ processing, co-processing, co-analysis, concurrent visualization, and co-visualization. Thus ParaView Catalyst, or Catalyst, is often referred to as a co-processing, or in situ, library for high-performance computing (HPC).

Figure 4 demonstrates a typical workflow using Catalyst for in situ processing. In this figure, we assume a simulation code is integrated with Catalyst. The end-user initiates the workflow by creating a Python script using the ParaView application graphical user interface (GUI), which specifies the desired output from the simulation. Next, when the simulation starts, it loads the Python script; then, during execution, Catalyst generates synchronously (i.e. while the simulation is running) any analysis and visualization output. Catalyst can produce images (i.e. screenshots)

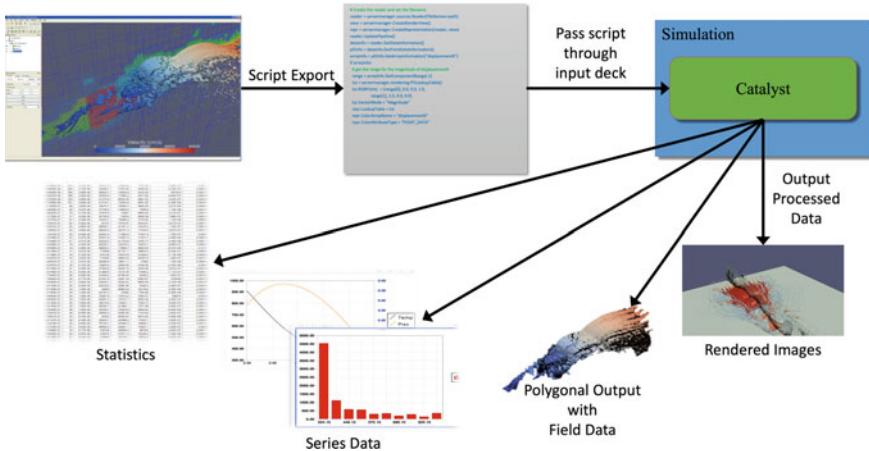


Fig. 4 In situ workflow with a variety of Catalyst outputs

and image databases [3], compute statistical quantities, generate plots, and extract derived information such as polygonal data, such as iso-surfaces, to visualize.

A variety of simulation codes have used Catalyst. A subset list of these codes instrumented to use Catalyst include PHASTA from the University of Colorado, Boulder [22]; MPAS-Atmosphere and MPAS-Ocean from the climate modeling group at Los Alamos National Laboratory (LANL) and the National Center for Atmospheric Research (NCAR) [31]; XRAGE, NPIC, and VPIC from LANL [20]; HPCMP CREATE-AV™ Helios from the U.S. Army's CCDC AvMC Technology Development Directorate; CTH, Albany and the Sierra simulation framework from Sandia National Laboratories [18]; H3D from the University of California, San Diego (UCSD), and Code Saturne from Électricité de France (EDF) [16].

The most significant scale run to date used over 1 million MPI processes on Argonne National Laboratory's BlueGene/Q Mira machine [4]. The scaling studies utilized PHASTA, a highly scalable CFD code, developed by Kenneth Jansen at the University of Colorado, Boulder, for simulating active flow control on complex wing design.

3.1 Integration with Simulation

In this section, we describe how developers can interface a simulation code with the ParaView Catalyst libraries. The interface to the simulation code is called an *adaptor*. Its primary function is to adapt the information in internal data structures of the simulation code and transform these data structures into forms that Catalyst can process. We depict this process in Fig. 5.



Fig. 5 ParaView Catalyst interface architecture

A developer creating an adaptor needs to know the simulation code data structures, the VTK data model, and the Catalyst application programming interface (API).

3.1.1 Simulation Codebase Footprint

Although interfacing Catalyst with a simulation code may require significant effort, the impact on the codebase is minimal. In most situations, the simulation code only calls three functions.

First, we must *initialize* Catalyst in order to place the environment in the proper state. For codes that depend on the message-passing interface (MPI), we place this method after the `MPI_Init()` call.

```

MPI_Init(argc, argv);
#ifndef CATALYST
CatalystInitialize(argc, argv);
#endif
  
```

Next, we call the *coprocess* method to check on any computations that Catalyst may need to perform. This call needs to provide the simulation mesh and field data structures to the adaptor as well as time and time step information. It may also provide additional control information, but that is not required. Typically, we call the *coprocess* method at the end of every time step in the simulation code after updating the fields and possibly the mesh.

```

for (int timeStep=0; timeStep < numberOfTimeSteps; timeStep++) {
    // < simulation does its thing >
    // < update fields and possibly mesh after timeStep >
#ifndef CATALYST
    CatalystCoProcess(timeStep, time, <grid info>, <field info>);
#endif
}
  
```

Finally, we must *finalize* Catalyst state and adequately clean up. For codes that depend on MPI, we place this method before the `MPI_Finalize()` call.

```

#ifndef CATALYST
CatalystFinalize();
#endif
MPI_Finalize();
  
```

In general, we colocate the *initialize* and the *finalize* methods with the *coprocess* method in the adaptor and the developer implements the adaptor code in a separate source file, which simplifies the simulation code build process.

3.1.2 Instrumentation Details

As shown in Fig. 5, the adaptor code is responsible for the interface between the simulation code and Catalyst.

Core to the adaptor is the `vtkCPPProcessor` class, which manages the in situ analysis and visualization pipelines, which in turn automate the flow of data through a series of tasks. Given

```
vtkCPPProcessor* Processor = NULL; // static data
```

we can define an example *initialize* method, `CatalystInitialize`, as

```
void CatalystInitialize(int numScripts, char* scripts[]) {
    if (Processor == NULL) {
        Processor = vtkCPPProcessor::New();
        Processor->Initialize();
    }
    // scripts are passed in as command line arguments
    for (int i=0; i<numScripts; i++) {
        vtkCPPythonScriptPipeline* pipeline =
            vtkCPPythonScriptPipeline::New();
        pipeline->Initialize(scripts[i]);
        Processor->AddPipeline(pipeline);
        pipeline->Delete();
    }
}
```

In this way, we provide pipeline scripts passed in as command-line arguments to an instantiation of `vtkCPPProcessor` to manage. For our example, the `finalize` method, `CatalystFinalize`, simply deletes the storage for any defined pipelines.

```
void CatalystFinalize() {
    if (Processor) {
        Processor->Delete();
        Processor = NULL;
    }
}
```

Besides being responsible for initializing and finalizing Catalyst, the other responsibilities of the adaptor are:

- Determining whether or not to perform co-processing.
- Mapping the simulation fields and mesh to VTK data objects for co-processing.

For this example, we specify the mesh as a uniform, rectilinear grid defined by the number of points in each direction and the uniform spacing between points. There is only one field associated with this mesh, which is called temperature and defined over the points (vertices or nodes) of the mesh. Thus, the `coprocess` method, `CatalystCoProcess`, performs the following commonly required tasks:

```
void CatalystCoProcess(
    int timeStep, double time, unsigned int numPoints[3],
    unsigned int numGlobalPoints[3], double spacing[3],
    double* field) {
    vtkCPDataDescription* dataDescription =

```

```

vtkCPDataDescription::New();

dataDescription->AddInput("input");
// 1. Specify the current time and time step for Catalyst.
dataDescription->SetTimeData(time, timeStep);
// 2. Check whether Catalyst has anything to do at this time.
if (Processor->RequestDataDescription(dataDescription) != 0) {
    // 3. Create the mapped VTK mesh.
    vtkImageData* grid = vtkImageData::New();
    grid->SetExtents(
        0, numPoints[0]-1, 0, numPoints[1]-1, 0, numPoints[2]-1);
    // 4. Identify the VTK mesh for Catalyst to use.
    dataDescription->GetInputDescriptionByName("input")->
        SetGrid(grid);
    dataDescription->GetInputDescriptionByName("input")->
        SetWholeExtent(0, numGlobalPoints[0]-1,
                      0, numGlobalPoints[1]-1,
                      0, numGlobalPoints[2]-1);

    grid->Delete();
    // 5. Associate mapped VTK fields with the mapped VTK mesh.
    vtkDoubleArray* array = vtkDoubleArray::New();
    array->SetName("temperature");
    array->SetArray(field, grid->GetNumberOfPoints(), 1);
    grid->GetPointData()->AddArray(array);
    array->Delete();
    // 6. Call CoProcess to execute pipelines.
    Processor->CoProcess(dataDescription);
}
dataDescription->Delete();
}

```

In Sect. 3.3 we'll discuss the details of the API to help solidify the understanding of the flow of information.

3.2 Runtime Behavior

The analysis and visualization methods can be implemented in C++ or Python and can run in situ, in transit, or a hybrid of the two methods. Python scripts can be crafted from scratch or using the ParaView GUI to set up prototypes and export as Catalyst scripts interactively.

We designed Catalyst to run synchronously (tightly coupled) with the simulation supporting in situ workflows, where we execute analysis methods and visualization pipelines alongside the simulation, leveraging the same address space.

Catalyst can support in transit workflows using two sub-groups of a global MPI communicator: one for simulation processes and one for analysis and visualization processes. However, the data movement from the simulation processes is not automatic and requires the writing of an additional communication routine during instrumentation.

Much more commonly, Catalyst enables hybrid workflows using either VTK's I/O capabilities or by leveraging additional middleware such as ADIOS [4]. For example, analysis methods and visualization pipelines could send intermediate results to burst buffers, and ParaView or another application would pull data from the burst buffers for interaction and further analysis.

Also, Catalyst can connect to a separately running ParaView Live session for exploring results on the fly. The Live method can facilitate a Monitoring/Steering workflow. This capability, in turn, enables subtly unique steering workflows, where the analysis methods and visualization pipelines are modified interactively through user feedback.

Finally, we aligned synchronous and asynchronous communication patterns with specific Catalyst workflows. Live supports both, and communications can be changed, as described above with hybrid workflows, utilizing third-party software.

3.3 Underlying Implementation

The core of our implementation is how the adaptor passes information back and forth between the simulation code and Catalyst. We need to exchange three types of information: VTK data objects, pipelines, and control information. The VTK data objects are the information containing the input to the pipelines. The pipelines specify what operations to perform on the data and how to output the results. The control information specifies when each pipeline should execute, and the required information from the VTK data objects needed to execute the pipelines properly.

Before providing the details of the API, we want to describe the flow of information and its purpose. This information affords a higher level of understanding of how the pieces work together.

First, we initialize Catalyst, which sets up the environment and establishes the pipelines to execute. Next, we execute the pipelines as required. Finally, we finalize Catalyst.

The initialize and finalize steps are reasonably straightforward, but the intermediate step has a lot happening in the underlying implementation. Principally, the intermediate step queries the pipelines to see if any of the pipelines require processing. If not, then control returns immediately to the simulation code. This query is nearly instantaneous, where the expectation of many calls wastes negligible compute cycles. On the other hand, if one or more pipelines demand re-execution, then the adaptor needs to update the VTK data objects representing the mesh and fields from the simulation, and then execute the desired pipelines with Catalyst. The execution time can vary widely depending on the quantity and type of tasks. Once the re-executing pipelines finish, then control returns to the simulation code.

The main classes of interest for the Catalyst API are `vtkCPProcessor`, `vtkCPPipeline`, `vtkCPDataDescription`, `vtkCPIInputDataDescription`, and the derived classes that are specialized for Python. When Catalyst is built with Python support, all of these classes are Python wrapped as well.

`vtkCPProcessor` is responsible for managing the pipelines. This management includes storing them, querying them, and executing them. Note that the `AddPipeline` method fundamentally adds a pipeline (`vtkCPPipeline` or `vtkCPPythonScriptPipeline`) for execution at requested times. This class mimics the structure of the simulation instrumentation.

First, the `Initialize` method initializes the object and sets up Catalyst. The initialization method uses either `MPI_COMM_WORLD` or an API supplied MPI communicator. Note that the `Initialize` method can depend on `vtkMPICommunicatorOpaqueComm`, defined in `vtkMPI.h`, and is used to avoid directly having to include the `mpi.h` header file. Next, the `CoProcess` method executes the proper pipelines based on information in the required argument description. When applying this method, we update and add the description to the `vtkDataObject` representing the mesh and fields. We use the helper method, `RequestDataDescription`, to determine, for a given description, if we desire execution of any pipelines. For this method, the description argument should have the current time and time step set and the identifier for available inputs. Finally, the `Finalize` method releases all resources used by Catalyst. If a Catalyst Python script includes a `Finalize` method, we execute this method at this point.

The `vtkCPDataDescription` class stores information passed between the adaptor and the pipelines. The provided information comes from either the adaptor for the pipeline or the pipeline for the adaptor. The adaptor needs to provide the pipelines with the current time, the current time step, and the names for input meshes produced by the simulation. For most use cases, the adaptor will provide a single input mesh to Catalyst called `input`. Naming the inputs is needed for situations where the adaptor provides multiple input meshes with each mesh treated independently.

The `vtkCPIInputDataDescription` class is similar to `vtkCPDataDescription` in that it passes information between the adaptor and the pipelines. The difference is that `vtkCPIInputDataDescription` passes information about the meshes and fields.

Finally, there are a variety of other methods to increase the efficiency of the adaptor. For example, to streamline data preparation for coprocessing, other methods may inform the adaptor of the requested fields for the pipelines.

3.4 HPCMP CREATE-AV™ Helios Use Case

The U.S. Department of Defense’s High-Performance Computing Modernization Program’s (HPCMP) Computational Research and Engineering Acquisition Tools and Environments for Air Vehicles (CREATE-AV) project has overseen the development of a rotorcraft simulation tool called Helios [26, 27, 34], a high-fidelity, multi-disciplinary computational analysis platform for rotorcraft aeromechanics simulations. Used by academia, government, and industry, Helios handles the aerodynamics solutions using a dual-mesh paradigm: body-fitting meshes in the near-body region and adaptive mesh refinement (AMR) meshes in the off-body region.

The Helios package contains tools for a near-complete workflow, except geometry creation and post-processing tools. These tools include specifying rotor blade geometry and movement, mesh assembly and partitioning, a graphical user interface (GUI) for defining simulation input parameters, the parallel simulation environment, and management of simulation results. The default computational fluid dynamics

(CFD) libraries include SAMCart for CFD solution in the off-body region and a choice of using kCFD and mStrand for the near-body region. Additionally, FUN3D and OVERFLOW can be used as plugins CFD solvers for the near-body region.

3.4.1 Specialized Workflow for Rotorcraft Analysis

Helios handles specialized complex high-fidelity simulations of coupled fluid-structure interaction for a variety of flight conditions, including rotorcraft flying in their turbulent wake. Likewise, the Helios development team tailored the in situ processing.

The first Helios release that used ParaView Catalyst for in situ capabilities was version 3, which only included streamlines, slices, and contours and required the end-user to hand-edit Python input files. December 2019 marked the latest Helios release, version 10. Since version 3, the developers enhanced the in situ operations with particle paths, Cartesian extracts, taps, and derived variable calculations, defined through Helios's pre-processing GUI, shown in Fig. 6. Helios can utilize custom Catalyst scripts created with the ParaView GUI, but end-users seem to prefer using the pre-processing GUI due to the specialized nature of rotorcraft analysis.

The widely varying post-processing experience with specific tools by Helios end-users dictated the use of data extracts for the in situ outputs. Thus, enabling the end-user to manage their regular post-processing workflow with familiar tools.

3.4.2 Specialized Catalyst Edition

ParaView is a large software project with a variety of functionalities not required for the batch in situ processing done with Helios. Since Helios does not generate in situ images, all rendering components can be excluded from the Helios specific Catalyst in situ library. In addition, we can remove most data readers and writers except the readers for in situ restart and the writers for extracts (no requirement for input/output libraries like HDF5 or NetCDF). Customizing ParaView Catalyst specific to Helios provides the following benefits:

- Reduction in the number of source code files and associated compile time.
- Decrease in the number of third-party library dependencies and simplifying the build and install process.
- Reduction in the Catalyst library size and faster shared library load times due to smaller library size and a smaller number of linked libraries [7].

Version 10 of Helios uses a specialized Catalyst edition based off of the ParaView 5.6 release that includes Python wrapping.¹ Table 1 shows the memory load for three different Helios shared library build configurations: without Catalyst, with Catalyst

¹ This ParaView edition is available at <https://github.com/acbauer/ParaViewParticleTrackingCatalystEdition>.

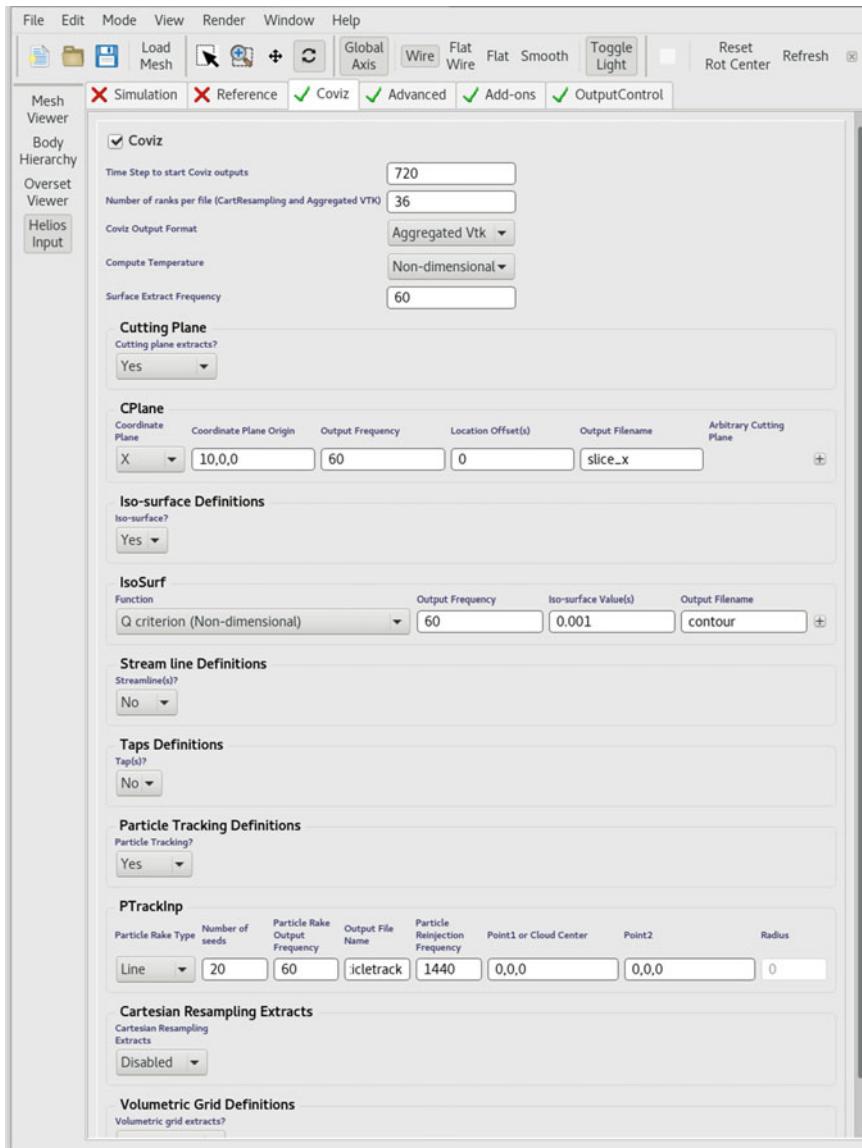
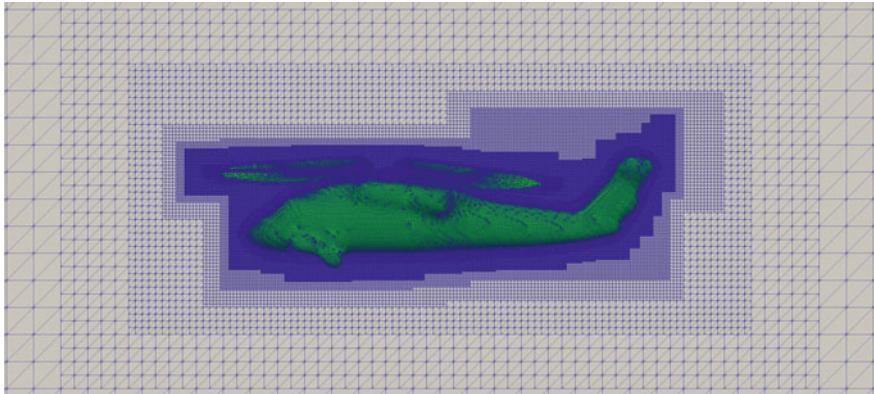


Fig. 6 Example set up of the in situ panel of Helios's pre-processing GUI

Table 1 Helios memory reporting on node 1 on a Cray XC40/50

In Situ build type	Memory usage (MB)
None	24,450
Standard ParaView Catalyst	29,798
Helios Catalyst Edition	26,958

**Fig. 7** In situ slice and surface extract showing the Helios near-body grids (green) and off-body grids (grey)

5.6, and the specialized Catalyst edition. The in situ extracts for these comparison runs were the internal surface and slice. The simulation runs used the Department of Defense’s Onyx HPC machine at the Engineer Research and Development Center, which is a Cray XC40/50 with the memory reported from the first node by Helios’s memory reporting routines.

3.4.3 Combination of Bespoke and VTK Functionality

Helios uses a dual-grid paradigm where an AMR grid is used sufficiently far from the rotorcraft body and a curve-fitting grid around the rotorcraft body, as shown in Fig. 7. Depending on the setup, the end-user may select different solvers on the near-body grid and over the computational domain.

For example, kCFD could be used to compute the CFD solution on the grid for the rotorcraft fuselage, OVERFLOW could be used to compute the CFD solution for the rotor blades, and SAMCart could be used to compute the CFD solution over the off-body AMR grid. The rotor blade grids rotate, and their intersection with the other grids will change as the simulation proceeds. Thus, the grid overlap needs to be computed dynamically along with the blanking on the overlapping portion of the grids.

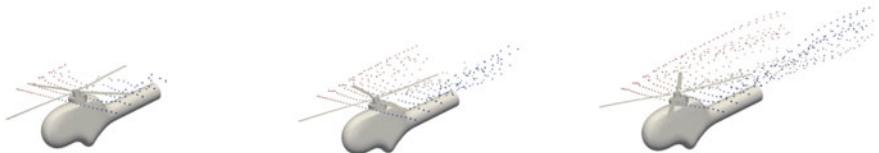


Fig. 8 In situ particle path and surface extract outputs for Higher-harmonic control Aeroacoustics Rotor Test II case. Images used with permission from Andrew Bauer courtesy of Kitware Source Quarterly Magazine

PUNDIT is a library that computes the overlap and blanking while transferring fields between grids. One of the Helios in situ outputs is an interpolated result onto a Cartesian grid where PUNDIT performs the interpolation computation. The Python-wrapped PUNDIT seamlessly operates within VTK's Python wrapping to interface with both Numpy and the VTK writers to output the desired information for a variety of post-processing tools. This combination of bespoke Helios and VTK functionality provides a convenient way to implement essential in situ functionality.

3.4.4 Temporal Analysis

Traditionally, modifying an in situ temporal analysis with an associated post-processing tool to work in situ has been difficult principally due to the pipeline architecture employed by these visualization tools. ParaView has a separate in situ particle path filter that works around this limitation. This filter is responsible for caching the dataset from previous time steps to relieve the visualization pipeline of this obligation. Additionally, it supports a simulation restarting the in situ particle path computation by reading specified particle locations from a file. The in situ particle path computation must behave the same regardless of whether the simulation was restarted or continuously computed from the initial conditions. Figure 8 demonstrates this functionality for the popular Higher-harmonic control Aeroacoustics Rotor Test II (HART-II) test case that maintains a good validation database.

Besides in situ particle paths, Helios supports temporal averaging of the interpolated output onto the Cartesian grid. The reason for implementing this functionality in a bespoke manner was the simplicity in computing the temporal average natively within Helios. As with the previous bespoke solution, it supports simulation restart using VTK writers and readers to dump out and read back in, respectively, restart information.

3.4.5 Zero-Copy Issues

For the in situ particle path filter to update the particle path location at a time step, it requires the full solution at both the current and previous time steps. This requirement

prevents the adaptor from using a zero-copy of the simulation data arrays on the full dataset since the CFD solvers are not caching their meshes or fields for previous time steps. Also, because Helios uses multiple CFD solvers in a single CFD simulation, each of these CFD solvers will have a different non-dimensionalization scheme and store fields in the solver specific non-dimensionalized form.

Also, the Helios in situ output is always in the SAMCart, or off-body CFD solver, non-dimensionalized form. Thus, all near-body CFD fields require conversion to this non-dimensionalization form. This conversion prevents using zero-copying of the near-body fields, even without requesting in situ particle path output. In the future, after ParaView 5.6, the `vtkScaledSOADataArrayTemplate` class² will be used to alleviate this limitation.

3.4.6 Common Output Benefit

There are multiple reasons that the Helios tools and workflow support multiple CFD solvers for the near-body grids. A primary reason is the validation by comparing the results of different CFD solvers for the same simulation case. Comparing results through full data dumps in each CFD solver’s native format is a complex and burdensome task. With in situ data extracts, both the near-body and off-body grids use a common data format, and all of the fields are in a consistent non-dimensionalization scheme, regardless of which near-body CFD solver used, enabling easy comparisons.

4 Conclusion

Libsim and Catalyst provide extensive tools for performing in situ visualization. They are likely the most feature-rich in situ libraries available to date.

That said, other similar in situ libraries exist. Lighter weight scripting libraries like Mayavi [8, 21] and yt [30] have been leveraged to perform in situ visualization. Other libraries like Ascent [15] are being designed from the ground up with in situ in mind. In contrast, some simulation frameworks, such as SCIRun [19], incorporate their own visualization functionality that can be used in situ.

Directly using the Libsim or Catalyst library requires what is often referred to as a “closely coupled” or “on-node proximity” in which the library is linked with the data producing program. However, they can be used with a general interface layer such as SENSEI [4] or Damaris/Viz [10] to decouple the in situ library from the data production. I/O libraries such as ADIOS [1, 17] can similarly be used for decoupling.

See Bauer, et al. [6] for a broader literature review of current in situ tools.

We have seen in this chapter that Libsim and Catalyst share many features and design decisions. When they initially started, each had their own focus. Libsim got its early start as an interactive simulation debugging tool, but as file I/O became a major

² <https://vtk.org/doc/nightly/html/classvtkScaledSOADataArrayTemplate.html>.

bottleneck on HPC, Libsim's main mode shifted to batch processing. Conversely, Catalyst got its start as a batch coprocessing library [12], but as use grew, interactive capabilities were added. Today, the functionality of the two tools overlap. The major difference is in the post-processing tool that each best interfaces with (VisIt versus ParaView), and simulation teams would do well to integrate the *in situ* library that works best with the other visualization tools used by the team.

Acknowledgements This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This chapter describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the chapter do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND 2020-8456 B.

Material presented in this chapter is a product of the CREATE (Computational Research and Engineering for Acquisition Tools and Environments) element of the U.S. Department of Defense HPC Modernization Program Office (HPCMO). Detailed input from the CREATE-AVTM Helios development team was provided in order to properly customize the *in situ* workflow for rotorcraft analysis. Mark Potsdam of the U.S. Army's CCDC AvMC Technology Development Directorate was the main technical point of contact for Army SBIRs and has contributed significantly to the vision of Catalyst.

References

1. Abbasi, H., Lofstead, J., Zheng, F., Schwan, K., Wolf, M., Klasky, S.: Extending I/O through high performance data services. In: IEEE International Conference on Cluster Computing and Workshops (2009). <https://doi.org/10.1109/CLUSTR.2009.5289167>
2. Ahrens, J., Geveci, B., Law, C.: ParaView: An end-user tool for large data visualization. In: Visualization Handbook. Elsevier (2005). ISBN 978-0123875822
3. Ahrens, J., Jourdain, S., O'Leary, P., Patchett, J., Rogers, D.H., Petersen, M.: An image-based approach to extreme scale *in situ* visualization and analysis. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 424–434 (2014). <https://doi.org/10.1109/SC.2014.40>
4. Ayachit, U., Bauer, A., Duque, E.P.N., Eisenhauer, G., Ferrier, N., Gu, J., Jansen, K.E., Loring, B., Lukic, Z., Menon, S., Morozov, D., O'Leary, P., Ranjan, R., Rasquin, M., Stone, C.P., Vishwanath, V., Weber, G.H., Whitlock, B., Wolf, M., Wu, K.J., Bethel, E.W.: Performance analysis, design considerations, and applications of extreme-scale *in situ* infrastructures. In: SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2016). <https://doi.org/10.1109/SC.2016.78>
5. Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., Mauldin, J.: Paraview catalyst: Enabling *in situ* data analysis and visualization. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV 2015), pp. 25–29 (2015). <https://doi.org/10.1145/2828612.2828624>
6. Bauer, A.C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., Moreland, K., O'Leary, P., Vishwanath, V., Whitlock, B., Bethel, E.W.: In situ methods, infrastructures, and applications on high performance computing platforms. Comput. Gr. Forum **35**(3), 577–597 (2016). <https://doi.org/10.1111/cgf.12930>

7. Boeckel, B., Ayachit, U.: Why is paraview using all that memory? (2014) <https://blog.kitware.com/why-is-paraview-using-all-that-memory/>
8. Buffat, M., Cadiou, A., Penven, L.L., Pera, C.: In situ analysis and visualization of massively parallel computations. *Int. J. High Perform. Comput. Appl.* **31**(1), 83–90 (2017). <https://doi.org/10.1177/1094342015597081>
9. Childs, H.R., Brugger, E., Whitlock, B.J., Meredith, J.S., Ahern, S., Biagas, K., Miller, M.C., Weber, G.H., Harrison, C., Pugmire, D., Fogal, T., Garth, C., Sanderson, A., Bethel, E.W., Durant, M., Camp, D., Favre, J.M., Rubel, O., Navratil, P.: VisIt: An end-user tool for visualizing and analyzing very large data. In: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, pp. 357–368. Chapman and Hall (2012)
10. Dorier, M., Sisneros, R., Peterka, T., Antoniu, G., Semeraro, D.: Damaris/Viz: a nonintrusive, adaptable and user-friendly in situ visualization framework. In: *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)* (2013). <https://doi.org/10.1109/LDAV.2013.6675160>
11. Duque, E.P., Whitlock, B.J., Stone, C.P.: The impact of in situ data processing and analytics upon weak scaling of CFD solvers and workflows. In: *ParCFD* (2015)
12. Fabian, N., Moreland, K., Thompson, D., Bauer, A.C., Marion, P., Geveci, B., Rasquin, M., Jansen, K.E.: The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In: *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization*, pp. 89–96 (2011). <https://doi.org/10.1109/LDAV.2011.6092322>
13. Forsythe, J.R., Lynch, E., Polksy, S., Spalart, P.: Coupled flight simulator and cfd calculations of ship airwake using kestrel. In: *53rd AIAA Aerospace Sciences Meeting* (2015). <https://doi.org/10.2514/6.2015-0556>
14. Kirby, A., Yang, Z., Mavriplis, D., Duque, E., Whitlock, B.: Visualization and data analytics challenges of large-scale high-fidelity numerical simulations of wind energy applications. In: *2018 AIAA Aerospace Sciences Meeting* (2018). <https://doi.org/10.2514/6.2018-1171>
15. Larsen, M., Ahrens, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: 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 (ISAV '17)*, pp. 42–46 (2017). <https://doi.org/10.1145/3144769.3144778>
16. Lorendeau, B., Fournier, Y., Ribes, A.: In-situ visualization in fluid mechanics using catalyst: A case study for Code Saturne. In: *IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)* (2013). <https://doi.org/10.1109/LDAV.2013.6675158>
17. Moreland, K., Oldfield, R., Marion, P., Jourdain, S., Podhorszki, N., Vishwanath, V., Fabian, N., Docan, C., Parashar, M., Hereld, M., Papka, M.E., Klasky, S.: Examples of *in transit* visualization. In: *Petascale Data Analytics: Challenges and Opportunities (PDAC-11)* (2011)
18. Oldfield, R.A., Moreland, K., Fabian, N., Rogers, D.: Evaluation of methods to integrate analysis into a large-scale shock physics code. In: *Proceedings of the 28th ACM international Conference on Supercomputing (ICS '14)*, pp. 83–92 (2014). <https://doi.org/10.1145/2597652.2597668>
19. Parker, S.G., Johnson, C.R.: SCIRun: A scientific programming environment for computational steering. In: *Proceedings ACM/IEEE Conference on Supercomputing* (1995)
20. Patchett, J., Ahrens, J., Nouanesengsy, B., Fasel, P., Oleary, P., Sewell, C., Woodring, J., Mitchell, C., Lo, L.T., Myers, K., Wendelberger, J., Canada, C., Daniels, M., Abhold, H., Rockefeller, G.: LANL CSSE L2: Case study of in situ data analysis in asc integrated codes. Technical Report LA-UR-13-26599, Los Alamos National Laboratory (2013)
21. Ramachandran, P., Varoquaux, G.: Mayavi: 3D visualization of scientific data. *Comput. Sci. Eng.* **13**(2), 40–51 (2011). <https://doi.org/10.1109/MCSE.2011.35>
22. Rasquin, M., Smith, C., Chitale, K., Seol, E.S., Matthews, B.A., Martin, J.L., Sahni, O., Loy, R.M., Shephard, M.S., Jansen, K.E.: Scalable implicit flow solver for realistic wing simulations with flow control. *Comput. Sci. Eng.* **16**, 13–21 (2014). <https://doi.org/10.1109/MCSE.2014.75>
23. Rintala, R.: In situ XDB Workflow Allows Coupling of CFD to Flight Simulator for Ship Airwake/Helicopter Interaction (2015). <http://www.ilight.com/en/news/in-situ>

- xdb-workflow-allows-coupling-of-cfd-to-flight-simulator-for-ship-airwake-helicopter-interaction (Accessed January 15, 2020)
- 24. Sanderson, A., Humphrey, A., Schmidt, J., Sisneros, R.: Coupling the uintah framework and the visit toolkit for parallel in situ data analysis and visualization and computational steering. In: Weiland, M., Alam, S., Shalf, J., Yokota, R. (eds.) High Performance Computing - ISC High Performance 2018 International Workshops, Revised Selected Papers, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 201–214. Springer (2018). https://doi.org/10.1007/978-3-030-02465-9_14
 - 25. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit: An Object Oriented Approach to 3D Graphics, fourth edn. Kitware Inc. (2004). ISBN 1-930934-19-X
 - 26. Sitaraman, J., Potsdam, M., Wissink, A., Jayaraman, B., Datta, A., Mavriplis, D., Saberi, H.: Rotor loads prediction using helios: a multisolver framework for rotorcraft aeromechanics analysis. *J. Aircr.* **50**(2), 478–492 (2013). <https://doi.org/10.2514/1.C031897>
 - 27. Sitaraman, J., Wissink, A., Sankaran, V., Jayaraman, B., Datta, A., Yang, Z., Mavriplis, D., Saberi, H., Potsdam, M., O'Brien, D., Cheng, R., Hariharan, N., Strawn, R.: Application of the helios computational platform to rotorcraft flowfields. In: 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition (2010). <https://doi.org/10.2514/6.2010-1230>
 - 28. Smith, T.M., Menon, S.: The structure of premixed flame in a spatially evolving turbulent flow. *Combust. Sci. Technol.* **119** (1996)
 - 29. Stone, C.P., Menon, S.: Open loop control of combustion instabilities in a model gas turbine combustor. *J. Turbul.* **4** (2003)
 - 30. Turk, M.J., Smith, B.D., Oishi, J.S., Skory, S., Skillman, S.W., Abel, T., Norman, M.L.: yt: a multi-code analysis toolkit for astrophysical simulation data. *Astrophys. J. Suppl. Ser.* **192**(9) (2011). <https://doi.org/10.1088/0067-0049/192/1/9>
 - 31. Turuncoglu, U.U.: Towards in-situ visualization integrated earth system models: RegESM 1.1 regional modelling system. *Geoscientific Model Development Discussions* (2018). <https://doi.org/10.5194/gmd-2018-179>
 - 32. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Eurographics Symposium on Parallel Graphics and Visualization (2011). <https://doi.org/10.2312/EGPGV/EGPGV11/101-109>
 - 33. Whitlock, B.J., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization, pp. 101–109. Eurographics Association (2011). <https://doi.org/10.2312/EGPGV/EGPGV11/101-109>
 - 34. Wissink, A.M., Potsdam, M., Sankaran, V., Sitaraman, J., Mavriplis, D.: A dual-mesh unstructured adaptive cartesian computational fluid dynamics approach for hover prediction. *J. Am. Helicopter Soc.* **61**(1), 1–19 (2016). <https://doi.org/10.4050/JAHS.61.012004>

The Adaptable IO System (ADIOS)



**David Pugmire, Norbert Podhorszki, Scott Klasky, Matthew Wolf,
James Kress, Mark Kim, Nicholas Thompson, Jeremy Logan, Ruonan Wang,
Kshitij Mehta, Eric Suchyta, William Godoy, Jong Choi, George Ostrouchov,
Lipeng Wan, Jieyang Chen, Berk Geveci Chuck Atkins, Caitlin Ross,
Greg Eisenhauer, Junmin Gu, John Wu, Axel Huebl, and Seiji Tsutsumi**

Abstract The Adaptable I/O System (ADIOS) provides a publish/subscribe abstraction for data access and storage. The framework provides various engines for producing and consuming data through different mediums (storage, memory, network) for various application scenarios. ADIOS engines exist to write/read files on a storage system, to couple independent simulations together or to stream data from a simulation to analysis and visualization tools via the computer's network infrastructure, and to stream experimental/observational data from the producer to data processors via the wide-area-network. Both lossy and lossless compression are supported by ADIOS to provide for seamless exchange of data between producer and consumer. In this work we provide a description for the ADIOS framework and the abstractions provided. We demonstrate the capabilities of the ADIOS framework using a number of examples, including strong coupling of simulation codes, *in situ* visualization running on a separate computing cluster, and streaming of experimental data between Asia and the United States.

D. Pugmire (✉) · N. Podhorszki · S. Klasky · M. Wolf · J. Kress · M. Kim · N. Thompson ·
J. Logan · R. Wang · K. Mehta · E. Suchyta · W. Godoy · J. Choi · G. Ostrouchov · L. Wan ·

J. Chen

Oak Ridge National Laboratory, Oak Ridge, TN, USA

e-mail: pugmire@ornl.gov

B. G. C. Atkins · C. Ross

Kitware, Inc., Clifton Park, NY, USA

G. Eisenhauer

Georgia Institute of Technology, Atlanta, GA, USA

J. Gu · J. Wu · A. Huebl

Lawrence Berkeley National Laboratory, Berkeley, CA, USA

S. Tsutsumi

Japan Aerospace Exploration Agency, Tokyo, Japan

1 Introduction

The Adaptable I/O System (ADIOS) was designed with the observation that applications almost universally read and write files from storage, and that this can be used as an abstraction for access to data [13]. ADIOS is a middleware layer that sits between the application and the computing system to manage the movement of data. This middleware layer makes it possible for an application to write data to a target that is determined at runtime. One possible target is traditional file storage. Other targets are able to support in situ processing methods, and include a memory buffer on the nodes where the application is running, or over the network to a memory buffer on another set of resources. Applications (e.g., visualization and analysis codes) can read data from any of the file or in situ targets. A schematic showing examples of these use cases in given in Fig. 1. The advantage of this design is that the sharing and movement of data is decoupled from the producer and the consumer, and can be modified at runtime as needed. This advantage addresses several of the challenges described in the Background chapter. Workflow execution is made easier when data producers and consumers can be connected together without modifying the source codes. Software complexity is reduced because the issues related to data access across evolving systems are provided by the middleware layer. Better resilience is possible because the producer and consumer need not run together in the same memory space.

In terms of the taxonomy described in the Background chapter, ADIOS can be classified in the following ways. *Integration type*: Apart from the usage of the ADIOS API for I/O, no additional instrumentation is needed by the application. *Proximity*: The visualization can run on the same or different computing resources. *Access*: The visualization can directly access memory in the simulation, or it can be copied to a memory buffer on the same or on different computing resources. *Division of Execution*: Synchronous or asynchronous are both possible, providing support for both time and space division. *Operation Controls*: Human in the loop is possible using both synchronous (blocking) or asynchronous (non-blocking) modes.

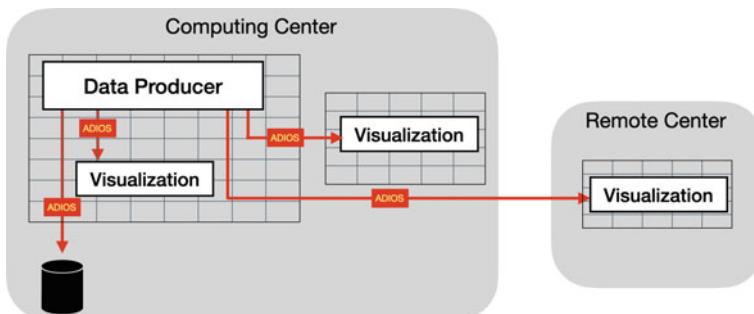


Fig. 1 Examples of ADIOS usage. In this example, the producer can write data to disk, to a visualization process sharing the same resource as the simulation, to a separate set of nodes, or across the network to a remote computing center

The ADIOS framework design was based on the goal to provide an I/O abstraction for parallel and distributed applications that expresses *what* data is produced for output and *when* that data is ready for output, or what data an application wants to read and when [16, 22, 23]. This is achieved in ADIOS through the use of different types of I/O *engines*. When an application writes a set of data, it uses the appropriate engine (e.g., File engine, In situ engine, etc.) to do the actual data movement. Similarly for a reader, it uses the appropriate Engine to request the data that it needs. For flexibility, the types of engine used by producer and consumer can be specified in a configuration file for runtime selection. In this way, applications that either read or write data need only select the appropriate output engine, and do not need to be concerned with the implementation details needed to achieve scalable performance.

The separation of concerns, namely that an application only need to be concerned about the data production and consumption but not how the data should be delivered, allows for creating optimized ADIOS engines that all can work with the same application code. Using the ADIOS interface makes application I/O *scalable*, a primary goal of the ADIOS framework, which is designed to work well on the largest supercomputers. ADIOS regularly runs on the largest supercomputers in the world for applications that consume and produce multiple petabytes of data during the course of a simulation run.

In this chapter we describe details of the ADIOS framework and how it can be used for in situ visualization. In Sect. 2, we describe the I/O abstractions used by ADIOS and how applications and visualization codes can use them. Section 2.1 provides a description of the engines provided by ADIOS and how they handle the movement of data. Advanced features in ADIOS are described in Sect. 2.2, followed by a discussion on the relative strengths of each engine and coding examples in Sects. 2.3 and 2.4. In Sect. 3 we describe the use of ADIOS for in situ visualization with application partners, followed by some concluding remarks in Sect. 4.

2 ADIOS I/O Abstraction

A parallel application that produces data uses ADIOS to define *variables* (n -dimensional distributed arrays of a particular type) and *attributes* (labels associated with individual variables or the entire output data set). It also specifies when the data is available for output. The output is organized around output *Steps*, instead of individual variables. A Step includes all the variables and attributes that are to be sent to the target at once. There is nothing in the ADIOS interface that prescribes how to handle the data (e.g. data aggregation among the processes, targeting a single file or one file per process, handling multiple readers, and handling the disappearance of a potential reader). These belong to the *IO strategy* and are implemented in various ways by different Engines. The user can control the behavior of the application by choosing a specific engine, and parameterizing it with available options.

Similarly, a reading application only declares what data it wants to retrieve from a source, each process of the parallel application declaring what it needs, and when

it expects the data to be in its memory. The input is also organized around Steps, not individual variables. The semantics of the ADIOS API ensure that a reader never gets into an inconsistent state where portion of the data of some variables belong to a certain step, and other portion to another step.

Analysis and visualization are typically data-hungry operations [7]. This makes scalable access to data key. In an in situ environment, the ability to maintain clear boundaries between simulation and analysis tasks can promote fault tolerance, interoperability, programmability and maintainability.

The flexibility of the data movement abstractions provided by ADIOS makes it easy to integrate with analysis and visualization applications. The “file-like” API provided by ADIOS allows seamless reads from disk, from memory, or streaming over the network. Likewise, on the write side, outputs produced by analysis and visualization applications can be written to disk, or shared with other applications through memory or streamed over the network. The abstraction used by ADIOS makes it easy to move data as it does something that the application is already doing, namely, reading and writing from files.

The abstraction provides the same access to data regardless of where the data are located, be it disk, memory or streaming. As examples, the VisIt [6] and ParaView [1] visualization tools have support for reading data from ADIOS in this manner. Both tools provide access to ADIOS data using a data reader plugin. The plugin reads the ADIOS data and creates mesh-based data that can be visualized by the VisIt and ParaView tools. An example of VisIt visualizing streaming data is described in Sect. 3.3.

When visualization is used in an in situ environment, the ability to have clear boundaries between the simulation and analysis and visualization, and rely on a middleware layer for the sharing and exchange of data is valuable in a number of ways. (1) It makes it much easier to reconfigure components in a workflow based on the needs of the scientific campaign. (2) Fault tolerance is increased because the simulation can be separated from the visualization. (3) The mechanism for sharing data between producer and consumer can more easily be modified, changed or replaced.

In the remainder of this section we describe ADIOS in more detail. In Sect. 2.1 and 2.2 we describe the ADIOS engines used to move data and some advanced topics on reduction and data interpretation. In Sect. 2.3 we discuss the characteristics of these engines and how they relate to visualization and analysis needs and costs. Finally, in Sect. 2.4 we show some code examples of how ADIOS is used for both data producers and data consumers.

2.1 ADIOS Engines

The mechanism in ADIOS for moving data is called an *engine*. An engine is tasked with executing the I/O heavy operations associated with the movement of data. Each engine supports a unified interface that allows data producers to *put* data, and data consumers to *get* data. The details of moving the data between source

and destination are left to particular implementation details of each engine. ADIOS provides a number of engines, which are described below.

2.1.1 File-Based Engines

ADIOS provides two types of engines for performing parallel IO of data to disk storage:

BPFile Engine. The BPFile engine is the default engine for storage. The output file target is a directory, which contains both metadata and data files. The number of files is tailored to the capability of the file system, not to the number of writers or readers, which ensures scalable I/O performance. The steps stored in a single file target, can be read by other applications step-by-step simultaneously. Therefore, this engine can be used for in situ processing through the file system.

HDF5 Engine. This engine can be used to write and read HDF5 formatted files. It uses the Parallel HDF5 library, so it provides only a compatibility layer to process HDF5 files in an ADIOS application workflow and is only as scalable as the HDF5 library itself. Streaming access to data is not currently available, but will be available once it is supported by HDF5.

2.1.2 Data Staging Engines

Data staging is a generic concept in ADIOS for providing concurrent access to data to one or more consumers through memory or streamed over the network. It can map onto both time and space division of the taxonomy from the Background chapter. Data staging engines are typically used for doing in situ analysis and visualization and code coupling. With staging engines, the data producer will write the data using the ADIOS API. The data are then available to be read by the consumers using the ADIOS API. Each staging engine has the ability to move the data in different ways, which are described below.

Scalable Staging Transport (SST). The most versatile and flexible staging engine uses either RDMA, TCP, UDP, or shared memory to move data from a producer (parallel application) to other consumers (multiple independent parallel applications). Consumers can come and go dynamically without affecting the producer. The output step is buffered in the producer's memory, and readers pull out portions of the buffered data with RDMA operations or communicate with a thread in the producer to receive it via TCP. The requirement of all engines to always provide a consistent view of a step to a reader may result in blocking the producer from progressing if the consumer is slower than the producer. The SST writer engine aggregates metadata for every step, and shares it with all readers. Readers then issue remote reading operations based on the I/O pattern in metadata. This allows the I/O pattern to vary over time. SST also allows readers to disconnect and reconnect while writers keep

writing. To address different application requirements, the SST buffering policy can be configured at run-time. This includes keeping only the most recent step, buffering a fixed window of consecutive steps, or blocking until the step is consumed. In cases where strong coupling is required between applications, the buffer limit can be set to 1, which ensures that every step is consumed by the reader before the producer moves to the next data step. For use cases like interactive visualization, buffering only the latest step is useful since the user typically does not want to block the simulation while a particular time step is explored. While SST aims to provide the flexibility for addressing various application requirements, the fact that it manages metadata for every single step may be overkill for uses cases where the metadata does not change frequently, or at all.

Insitu-MPI. This engine focuses on the speed of data movement for use cases where the metadata is constant across the workflow and a single metadata aggregation at the first data step will suffice. After this first step, each writer and reader knows the exact I/O pattern and direct communication is performed using asynchronous send and receive operations using an MPI communicator. Since it uses MPI, the producer and consumer applications must be launched within a single `mpiexec` command using the Multiple Program Multiple Data (MPMD) mode. The engine directly sends the application data to the consumer, hence, the producer is synchronized to the consumer at every step to avoid modifying the data before it is received. For very large applications with constant I/O patterns, the Insitu-MPI engine can provide CPU savings for metadata management. However, since it must be launched in MPMD mode under MPI, the flexibility of readers dynamically join or leave is not supported at run-time.

Staging for Strong Coupling (SSC). The SSC engine is also designed for applications that have constant metadata over time. Similar to the Insitu-MPI engine, the SSC engine aggregates metadata once on the first time step. The main differences between SSC and Insitu-MPI are that SSC uses one-sided MPI communication and that the producer output is buffered. The one sided MPI paradigm does not require the send and receive calls to be paired. Instead, it allows direct access to remote memory of another process. The buffering of application data, on the other hand, enables the producer to continue with the computation while the data is transferred to the consumer. In very large scale coupling use cases this approach saves the overhead of one side waiting for the other side to complete the send and receive pairs, and makes it possible for applications to very quickly, and frequently exchange data.

DataMan. This engine focuses on providing good bandwidth over wide-area-networks (WAN). It uses the publish and subscribe communication mechanism of the ZeroMQ library and has been optimized specifically for long-distance low-latency data movement. Unlike other staging engines, such as SSC described above, Data-Man does not guarantee that every data step is transferred. Instead, the subscriber is designed to read only the latest data steps, while ignoring the previous steps. This saves the two-way communications for checking step completion, which usually means several hundred milliseconds in inter-continental data transfers. Because

of this, the data transfer latency is greatly reduced and can support near-real-time analysis better than other engines over the WAN.

2.2 *Advanced Data Management Services*

ADIOS has a number of internal and external supports for advanced management of data. These include data compression, and schemas for providing additional information about ADIOS data to help downstream processing applications, such as analysis and visualization to properly interpret the raw data.

2.2.1 **Data Compression**

ADIOS supports operators as a mechanism for performing calculations on the data before it is written by an engine. A general purpose operator, called a Callback provides the user with the ability to perform arbitrary calculations and manipulations to the data inside the engine. Data compression is provided in ADIOS using this mechanism. It provides support for a number of different lossless and lossy compression methods, which are described below.

In the classical workflow for high-performance scientific simulations, the entire data set is written to storage after generation. This will no longer be viable at the exascale, simply because the amount of data will swamp the filesystem. To accelerate scientific discovery, we must prioritize information over data. It will be vital to take advantage of a priori user information to prioritize the most useful data so that I/O can be completed under standard HPC time constraints. (For example, on Summit [27], jobs are limited to 24 h.) One solution is data compression. ADIOS supports storing or transporting data in compressed form to reduce the I/O cost while preserving key information, which in turns speed up simulations or in situ data analysis and visualization [5]. Enabling compression requires minimal development effort from users. Simply specifying an operator for each variable enables ADIOS to automatically compress or decompress at the point of data publication or subscription. Lossless compressors such as BZip2 [25] preserve every bit of the data, but compression ratios observed in practice are minimal. Lossy compressors such as MGARD [2–4], SZ [10, 20, 26], and ZFP [21] provide much higher compression ratios (usually more than an order of magnitude than lossless), but information is lost. However, most lossy compressors allow control of the loss through parameters, which can be easily set in ADIOS. Also, as derived quantities in data are particular important for scientific discovery, one of the compressors supported by ADIOS, MGARD, can consider one or more relevant quantities of interest and reduce the data so as to preserve these quantities. Furthermore, ADIOS supports the meta-compressor Blosc which provides further lossless compressors (Zstd, Snappy, BloscLZ, LZ4HC) as well as shuffle pre-conditioners. In GPU-centric applications, using ADIOS with Blosc's threaded-chunked compressor variants regularly trades unutilized CPU-cycles for I/O speedup [15].

2.2.2 Schemas

Schemas provide the ability to annotate the semantics of the array-based layout of data in ADIOS. These provide the meaning of each data array, and the relationship between groups of arrays in an ADIOS file or stream. This capability makes it easier for tools using ADIOS to be used together in, for example, a complex scientific workflow. Two examples of such schemas are described below.

ADIS Visualization Schema. The Adaptable Data Interface for Services (ADIS) is a schema for describing mesh-based data that are used by visualisation tools. ADIS uses a JavaScript Object Notation (JSON) formatted strings to describe the content of ADIOS data. For example, for ADIOS data arrays representing field data on a uniform grid, the ADIS schema will specify that there is a uniform mesh of a given size, and the names of the arrays in the ADIOS stream for each field and the association on the mesh (e.g., zone centered, point centered, etc.). For more complex mesh types, like unstructured grids, ADIS specifies the names of the arrays for specifying the relevant mesh structures (e.g., point coordinate values, cell information, etc.).

ADIS also supports the creation of data sets from ADIOS in the VTK-m [24] format. Given a schema, and an ADIOS file/stream, ADIS will read data from ADIOS and construct the appropriate VTK-m data object.

openPMD Schema. The Open Standard for Particle-Mesh Data Files (openPMD) is a schema for describing mesh- and particle-based data. Its primary focus is the exchange, archival and replicability of scientific data via a minimal set of conventions and meta information. The schema is defined in the so-called “base standard” and “extensions”. The former is agnostic of the data’s scientific domain and can be automatically verified/parsed, visualized, scaled and dimensionally analyzed (describing units and quantities). The base standard also provides means to document authorship, hardware and software environments towards reproducible research. Based on this, standardized meta-information in openPMD schema “extensions” add further meaning for domain-scientists, e.g. by documenting algorithms and methods that generated a data set.

Contrary to visualization-focused and domain-specific schemas, openPMD is a notion for *scientifically self-describing* data in general, providing a unified description for data in scientific workflows from source, over processing, analysis and visualization to archival in (open) data repositories. openPMD is widely adopted in plasma-physics, particle-accelerator physics, photon-science, among others.¹

The schema can be added to data described via hierarchical, self-describing (portable) data formats. Open implementations are available in C++, Python and Fortran and currently range from MPI-parallel ADIOS1, ADIOS2 and HDF5 library backends to serial JSON files. The openPMD schema is versioned, citable and developed on GitHub. Its release is version 1.1.0 and data files using the schema are forward-updatable via lightweight meta-data transformations [14].

¹ Curated list available at <https://github.com/openPMD/openPMD-projects>.

2.3 Discussion

The number of engines available in ADIOS provides a large amount of flexibility when selecting a configuration. Further, multiple executables can be connected using the read/write API of an ADIOS engine to support a range of different types of workflows. The example workflow in Fig. 2 shows ADIOS (indicated with red arrows) being used as the data movement mechanism for a number of tasks. It is used to couple two simulations, in situ visualization on the HPC resource, in transit visualization on a cluster in the HPC center, and transfer data over the WAN to remote site for analysis. Additionally, data from a sensor/experiment is streamed over the WAN for analysis that uses simulation results.

When designing a visualization workflow, the choice of engine for each component is dependent on a number of factors. Broadly speaking these classes of visualization are post hoc, in situ (time division), and in transit (space division). Post hoc visualization is the traditional mode of visualization where the data are read from disk. As discussed in the Background chapter, in situ visualization, while a broadly defined term, is for simplicity, the case where the visualization and simulation use the same set of resources. In transit visualization uses two distinct sets of resources, one dedicated to the simulation and the other dedicated to the visualization. The network is used to transfer data between the two sets of resources. Below, we discuss

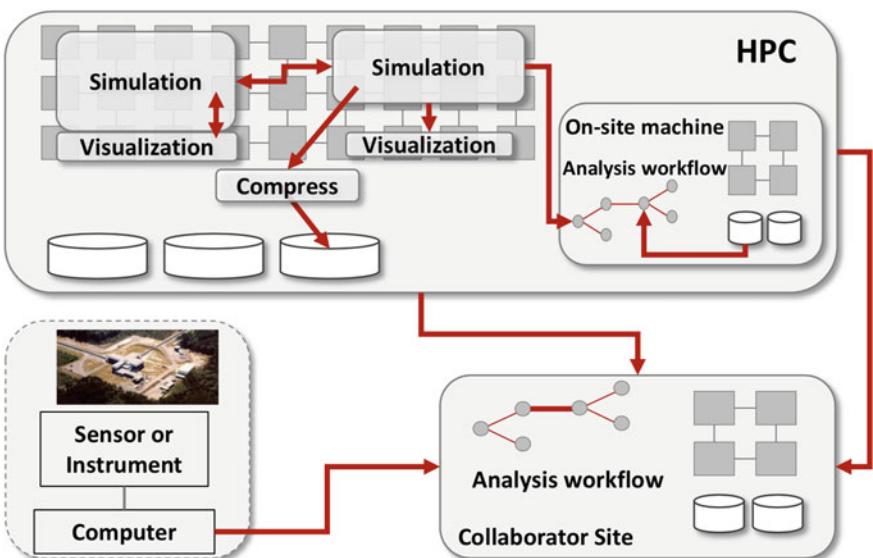


Fig. 2 Example workflow using ADIOS for simulation coupling, in situ visualization, in transit visualization, and streaming of both simulation and experimental data over the WAN to a remote site for analysis

these three modes from the ADIOS and visualization perspectives and the impact of choices made have on visualization functionality and performance.

2.3.1 ADIOS Perspective

From an ADIOS perspective, the following three characteristics are important: (1) data access and movement, (2) fault tolerance, and (3) programmability.

Data Access and Movement. Data access is defined as how much of the total spatio-temporal data are available, as the temporal range of data that are available. Data movement is the amount of data that must be moved from producer to consumer.

- **Post hoc:** Has access to all the spatio-temporal data that have been saved. However, the data movement cost is highest, and may restrict the amount of data available.
- **In situ/time division:** Has the highest access and lowest movement costs for spatio-temporal data as resources are shared with the producer. Access to multiple temporal steps requires additional on-node resources.
- **In transit/space division:** Data access is configurable based on needs. The dedicated resource can be sized to control the amount of spatio-temporal access, as well as temporal range. Since data movement occurs over the internal network, it is much faster than I/O.

Fault Tolerance. Fault tolerance describes the relative robustness of the system with respect to faults occurring in either the producer or the consumer.

- **Post hoc:** The consumer is independent from the data producer, so fault tolerance is very high.
- **In situ/time division:** Because resources are shared, the producer and consumer can impact each other. This includes faults, memory corruption, memory usage, etc.
- **In transit/space division:** Like post hoc, the consumer is independent from the data producer. Faults occurring on the dedicated nodes will not impact the producer.

Programmability Programmability describes the relative ease and flexibility of connecting a simulation with visualization. This includes composing a workflow, connecting components in a workflow, and modifying the underlying data movement mechanism. Since all three classes of visualization use the same abstraction, the programmability is improved by simply changing the engine used.

Table 1 provides a visual representation of the relative strengths of each ADIOS engine with respect the characteristics described above. A score for each engine is assigned based on how well the engine performs with respect to each characteristic described above. A “+” signifies a favorable evaluation, “–” a less favorable evaluation, and “0” for in between.

Table 1 Characterization of each ADIOS engine

		Engine type					
	ADIOS characteristics	BPFile	HDF5	SST	In situ-MPI	SSC	DataMan
Data access and data movement	<i>Spatial Access</i>	+	+	0	0	0	0
	<i>Temporal Fidelity</i>	-	-	+	+	+	0
	<i>Temporal Range</i>	+	+	0	0	0	0
	<i>Movement</i>	-	-	0	0	+	0
	Fault Tolerance	+	+	+	+	0	+
	Programability	+	0	+	+	+	+

2.3.2 Visualization Perspective

From a visualization perspective, a different set of characteristics are important (see for example, [19]). We discuss the following characteristics below: (1) scalability and resource requirements, (2) interactivity, (3) fault tolerance, and (4) programmability.

Scalability and Resource Requirements. Scalability is defined as how efficiently the visualization task can use the allocated resources. Resource requirements is defined as the need for additional resources beyond that of the simulation.

- **Post hoc:** Has the flexibility to allocate resources suitable for the required tasks, however I/O can be slow for large data.
- **In situ/time division:** Since the visualization must run at the scale of the visualization, the performance will depend on the operation. Communication heavy algorithms could suffer poor performance at larger scales.
- **In transit/space division:** Has the flexibility to allocate resources suitable for the required tasks. Since I/O is avoided, access to data can be much faster.

Interactivity. Interactivity is defined as the ability for a user to interact with the data, select regions of interest, and plot the data to extract understanding.

- **Post hoc:** Because visualization is independent from the simulation, full interactivity is possible with all data available.
- **In situ/time division:** Visualization has full access to all of the data that are available on the simulation resources. Due to limited available resources, the temporal range of data could be limited. If the data are shared, the simulation could be blocked while visualization occurs.
- **In transit/space division:** Visualization has full access to all of the spatio-temporal data that are moved to the dedicated resources. Because the data are not shared with the simulation, blocking can be avoided.

Fault Tolerance. As above, fault tolerance refers to the robustness of the visualization to avoid impacting the simulation.

- **Post hoc:** Visualization is independent from the simulation, so fault tolerance is very high.
- **In situ/time division:** Because resources are shared, it is possible for the visualization task to negatively impact the simulation.
- **In transit/space division:** Like post hoc, the visualization is independent from the simulation. Errors occurring on the dedicated nodes will not impact the simulation.

Programmability

As above, programmability describes the ease of using visualization tools with simulation data in a variety of configurations. This includes performing visualization tasks within a workflow, connecting analysis and visualization tasks together, and the ability to access data from different sources. Since all three classes of visualization use the same abstraction, the programmability is improved by simply changing the engine used.

Table 2 provides a visual representation of the relative strengths of each ADIOS engine with respect the important visualization characteristics described above. As above, a “+” signifies a favorable evaluation, “–” a less favorable evaluation, and “0” for in between.

Table 2 Characterization of each ADIOS engine for visualization

		Engine type					
	Visualization characteristics	BPFfile	HDF5	SST	Insitu-MPI	SSC	DataMan
Scalability	<i>Data</i>	–	–	+	+	+	–
	<i>Communication</i>	+	+	+	–	–	+
	<i>Resource</i>	+	+	–	+	+	–
Interactivity	<i>Spatial</i>	–	–	0	+	+	0
	<i>Temporal</i>	–	–	0	+	+	0
	<i>Temporal Range</i>	+	+	0	–	–	0
	<i>Block Simulation</i>	+	+	+	–	–	+
Fault Tolerance		+	+	+	+	0	+
Programmability		+	0	+	+	+	+

2.3.3 In Situ Data Placement and the Associated Performance Implications

Placement (in-line, in-transit, hybrid methods) is an important aspect to consider when planning for the use of in situ techniques. Performance can vary drastically depending on what analysis operations are used, how often they are performed, and at what scale they are performed. This performance difference is due primarily to the scaling characteristics of the analysis algorithms in relation to that of the underlying simulation, and can have a large effect on the overall cost of a simulation plus its visualization and analysis components.

In a work by Kress et al. [17] they look specifically at the cost of performing isocontours and ray tracing with parallel compositing both in-line and in-transit, and observe large cost variations based on placement as the simulation was scaled. Their work found that as the simulation was scaled to 16K cores, that visualization algorithms suffered large slowdowns in-line. However, if the data was transferred from the simulation over the network to a set of dedicated visualization nodes that the visualization routines completed much faster. They bring up a couple of general guidelines in that work: (1) if fastest time to solution is your goal at scale, moving the data and performing the visualization in-transit is the best solution; (2) if the lowest total combined cost of the simulation and visualization routines are the overall goal, the solution becomes more complicated. In general though, as the simulation scales if the visualization routines do not scale as well, moving the visualization routine to a smaller in-transit allocation is the best choice. However, careful consideration has to be given to how large of an in-transit allocation to reserve, and how often the visualization should be performed. A follow on work [18] develops a cost model to evaluate the use of in situ methods at scale.

2.4 *Code Examples*

This section contains examples using ADIOS to read and write data. The first example, shown in Listing 1, illustrates how a simulation code would write output data to disk using the BPFile engine. The engine type is specified in line 11.

Listing 1 Example of simulation writing outputs to a file.

```

1 adios2::ADIOS adios(MPI_COMM_WORLD);
2 // Declare named IO process
3 adios2::IO io = adios.DeclareIO("output");
4
5 // Declare output type and size.
6 adios2::Variable<double> var =
7     io.DefineVariable<double>("var", globalDims, offset, localDims);
8
9
10 // Set engine output to BPFile
11 io.SetEngine("BPFile");
12 adios2::Engine engine = io.Open("output.bp", adios2::Mode::Write);
13
14 // Run Simulation
15 for(...)
16 {
17     double *data = Simulation();
18
19     engine.BeginStep();
20     engine.Put(var, data);
21     engine.EndStep();
22 }
23 engine.Close();

```

Listing 2 shows a program that reads data from the output file and performs visualization on the data.

Listing 2 Example of a visualization program reading data from a file

```

1 adios2::ADIOS adios(MPI_COMM_WORLD);
2 adios2::IO io = adios.DeclareIO("input"); //Declare named IO process
3 io.SetEngine("BPFile");
4 adios2::Engine reader = io.Open("output.bp", adios2::Mode::Read);
5
6 std::vector<double> data;
7
8 while (reader.BeginStep(adios2::StepMode::Read) == adios2::StepStatus::OK)
9 {
10     adios2::Variable<double> var = reader.InquireVariable<double>("var");
11     if(var)
12         reader.Get<double>(var, data);
13     reader.EndStep();
14
15     Visualize(data);
16 }
17 engine.Close();

```

To change the simulation output mode from file based to the SST in situ mode, the only change required in Listings 1 and 2, is to change lines 11 and 3, respectively, from

`io.SetEngine("BPFile");`

to

`io.SetEngine("SST");`

The visualization program will now read the outputs produced by the simulation from the ADIOS stream named “output.bp”, which in this case, will be coming from

the SST engine in the simulation writer process. All of the engine types support by ADIOS can be changed in this way.

An alternative to specifying engine type in the source code is to use a configuration file, which is parsed at runtime and specifies the engines type and IO processes to be used. Both XML and YAML are supported as configuration file formats. the only change required in Listings 1 and 2, is to change line 1 from

```
adios2::ADIOS adios(MPI_COMM_WORLD);
```

to

```
adios2::ADIOS adios('config.xml', MPI_COMM_WORLD);
```

This allows the underlying data movement mechanism to be changed without re-compiling anything.

Listing 3 Configuration file, "config.xml" for examples shown in Listing 1 and 2

```
1 <!-- adios2 config file in XML format -->
2 <?xml version="1.0"?>
3 <adios-config>
4   <io name="output">
5     <!-- engine type can be set at runtime: BPFile, SST, etc. -->
6     <engine type="BPFile">
7     </engine>
8   </io>
9   <io name="input">
10    <engine type="BPFile" />
11  </io>
12 </adios-config>
```

Listing 4 Alternative configuration file, "config.yaml" for examples shown in Listing 1 and 2

```
1 ---
2 # adios2 config file in YAML format
3
4 - IO: "output"
5   Engine:
6     # engine type can be set at runtime: BPFile, SST, etc.
7     Type: "BPFile"
8
9 - IO: "input"
10  Engine:
11    Type: "BPFile"
```

Basic XML and YAML configuration files for ADIOS are shown in Listings 3 and 4 respectively. Changing the “type” field on line 7 from “BPFile” to “SST” will configure ADIOS to use the SST engine when the executables are run.

Listing 5 illustrates how to read ADIOS data using the Python high-level API. ADIOS provides a “pythonic” interface of an iterable container of steps using a generic “read” function that always return a numpy array for easy integration with the Python data analysis ecosystem. Similarly, switching between Engines is done through a parameter in the open function or using a config file as described above.

Listing 5 Python High-Level API Read Example

```

import adios2

with adios2.open("euler.bp", "r", engine_type="BPFile") as fh:

    for fstep in fh:

        # retrieve current step
        step = fstep.current_step()

        # inspect variables dictionary in current step
        step_vars = fstep.available_variables()
        for name, info in step_vars.items():
            print("variable_name: " + name)
            for key, value in info.items():
                print("\t" + key + ": " + value)
            print("\n")

        if( step == 0 ):
            size_in = fh_step.read("size")

        # read variables in current step
        # returning a numpy array for easy integration
        # with data science frameworks (e.g. pandas, scipy)
        T = fstep.read("T")

```

3 Example Use Cases

In this section we demonstrate how the I/O abstraction and engines described above in Sect. 2 can be used with different applications. These examples show how ADIOS engines can be used in different ways to accomplish the in situ processing required by a scientific campaign. The examples show how in situ can be used in both shared and separate resource configurations, and also include an example where data was streamed across the wide area network (WAN). In each example, we motivate the purpose of each scientific example and how ADIOS was used to provide the solution.

3.1 Strong Coupling in a Fusion Simulation

High-Fidelity Whole Device Modeling (WDM) of Magnetically Confined Fusion Plasmas is among the most computationally demanding and scientifically challenging simulation projects. The ten-year goal is to have a complete and comprehensive application that will include all the important physics components required to simulate a full toroidal discharge in a tokamak fusion reactor. The main driver is based on the strong coupling of two advanced and highly scalable gyrokinetic codes, XGC and GENE, where the former is a particle-in-cell code optimized for the treating the edge plasma while the other is a continuum code optimized for the core. Applications

for additional physics are intended to be coupled in the future, e.g. ones for material wall interactions or for high energy particles.

In the WDM workflow, the ADIOS BPFile engine is used to save checkpoint/restart files, offloads variables for in situ analysis and visualization [8]. For in-memory data exchange, the SST and Insitu-MPI engines are used for coupling of the core and edge simulations [11]. To date, three-dimensional field information has been shared between XGC and GENE, but a five-dimensional distribution function-based coupling is now under development. Published results [8, 11] have all relied on synchronous exchange, but asynchronicity will need to be explored in order to mitigate blocking while data are not available. The ADIOS SSC engine is designed to support the asynchronous WDM coupling workflow. ADIOS affords both performant scalability as data sizes grow with increased dimensionality, as well as APIs that support asynchronous operation.

3.2 *Streaming Experimental Data*

Fusion experiments provide critical information to validate and refine simulations that model complex physical processes in the fusion reactor as well as to test and validate hypotheses. Recent advances in sensors and imaging systems, such as sub-microsecond data acquisition capabilities and extremely fast 2D/3D imaging, allow researchers to capture very large volumes of data at high rates for monitoring and diagnostic purposes as well as post-experiment analyses. For example, JET, the world's largest magnetic confinement plasma physics experiment in the UK, is producing about 60 GB of diagnostic data per pulse [12]. A 2-D spatial imaging system, called Electron Cyclotron Emission Imaging (ECEI), in KSTAR, Korea, alone can capture 10 GB of image data per 10 s shot [29].

A system using ADIOS has been developed for KSTAR to support various data challenges by executing remote experimental data processing workflows in fusion science. It is one of the drivers for the development of the DataMan engine to support science workflows execution over the wide-area network (WAN) for near-real-time (NRT) streaming of experiment data to and from an experiment site and remote computing resource facilities.

An example of KSTAR workflow is shown in Fig. 3. This workflow is a multi-level workflow in that each box consists of one or more sub-workflows, each of which can be composed with ADIOS engines. One of the main goals is to stream online fusion experiment data from KSTAR in Korea to a computing facility in USA in order to perform various computational intensive analyses, such as instability prediction and disruption simulation. While our previous effort [9] focused on building remote workflows with data indexing, we are currently working on composing the KSTAR workflow with DataMan. In this workflow, we use ADIOS' DataMan engine to move raw observational data as streams from Korea to the USA. Once data streams arrived in a USA computing facility, we launch a set of analysis and visualization workflows to perform denoising, segmentation, feature detection, and selection for detecting any

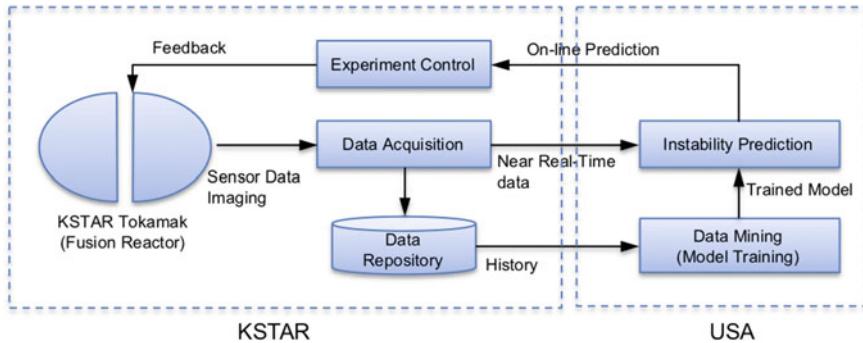


Fig. 3 Fusion instability monitoring and mitigation workflow

instabilities. Visualization results can be delivered back to Korea for designing the next upcoming shots. In short, ADIOS engines enable researchers to compose and execute workflows spanning local resources and remote large-scale high performance computing facilities for NRT analysis and decision-making.

3.3 Interactive in Transit Visualization

The Japan Aerospace Exploration Agency (JAXA) has implemented various ways for visualizing one of their CFD simulations, upacs-mc-LES. The visualization of CFD data consists of both batch and interactive visualization. Batch visualization is performed to create preset view images of the flowfield. Interactive visualization is conducted by interactively using Visit to understand the physics of the flowfield. While interactive visualization is not performed all the time during simulation, it is essential to have the capability to launch and attach the visualization process to the simulation when necessary, then to seamlessly detach when finished.

The agency has a heterogeneous HPC system, the Supercomputer System Generation 2 (JSS2). The main computer is a Fujitsu supercomputer with FX100 CPUs specialized for vector computations. Another cluster with x86 processors and GPUs is available for visualization and GPU-based analysis. There is a shared Lustre file system, which can be used for post-processing. An Ethernet and InfiniBand network connects the two machines, but only a portion of the nodes can communicate between the two machines. Most of the nodes can only communicate with other nodes on the internal network.

Batch visualization in post-processing is an easy way to produce movies of preset 3D visualizations on the GPU cluster, but it is stressing the file system and cannot support the largest simulations due to the I/O overhead. In situ visualization based on LibSim [28] is another approach, where the main computer is used to produce the images within the simulation code. In situ not only allows for producing a movie

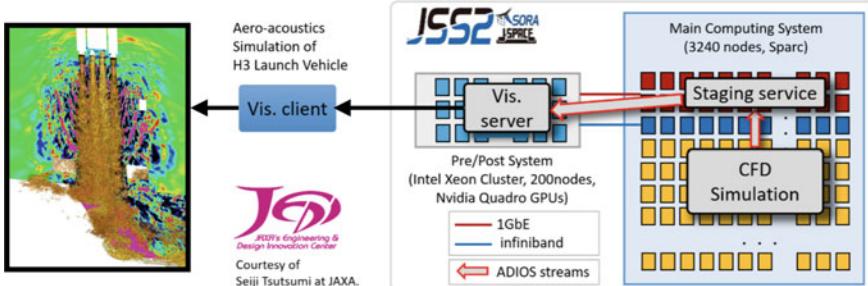


Fig. 4 Two steps of staging of data necessary on the JAXA heterogeneous system for interactive visualization. Simulation data is staged to a concurrent staging service on nodes that have network connections to the GPU cluster. The data is further staged to the visualization server running on the GPU cluster. The visualization client then visualizes the data. The visualization on the left shows acoustic waves on the cross section and exhaust jet are visualized by normalized pressure fluctuation and iso-surface of temperature, respectively

without dumping all data to disk but it also allows for interactive data exploration. ADIOS makes another approach feasible: in transit visualization where the simulation data is streamed from the main computer to another application, which in turn uses LibSim to create the visualizations. The visualization can be performed either on the main computer or on the GPU cluster (see Fig. 4). In all cases, Visit is used as the GUI for attaching to the visualization in case the user wants to interactively explore the data set.

The main drawback of in situ visualization with LibSim, for interactive exploration, is that the simulation process stops during interactive visualization. JAXA users want the simulation to progress with the computation while they are looking at a snapshot in time. In transit visualization using the ADIOS SST engine solves that problem and is as easy to use as in situ visualization when launching them as two separate applications together on the main computer in a single job.

Another advantage of in transit visualization (both for batch and interactive visualization) is that the simulation is not affected by the visualization process in terms of computing performance, nor by abnormal termination of the visualization process. The simulation progresses independently from the visualization and therefore the cost of visualization is amortized. On the other hand, data movement also has a cost and this offsets some of the advantages. As discussed in Sect. 2.3.3, there are trade-offs between the in situ and in transit approaches, and it depends on the simulation size, data size and visualization cost in order to determine which approach works best. Therefore, JAXA wants to maintain and provide all approaches to visualization for its users.

In transit visualization also provides the capability to use the GPU cluster for the visualization. The main difficulty with using a separate machine is that two jobs need to be submitted to two different machines and run at the same time. Current job scheduling policies only support batch processing. Therefore, the only way to do interactive visualization on the GPU cluster is to submit an interactive job once

the simulation is running. This is fine for interactive visualization where the user is present. Although ADIOS makes it possible to run the visualization application immediately and let it wait for the connection to the simulation indefinitely, for a batch visualization of an overnight job, this is still a waste of resources (on the GPU cluster).

Lastly, note that using ADIOS in the simulation to output the data, the target for the data can be a concurrent application for batch visualization on the main computer, or an application on the GPU cluster for interactive/batch visualization, or it can be the Lustre file system for storing data for post-processing. The visualization application is also the same for all the three cases. It is only a matter of the runtime setup and the choice of the ADIOS Engine to run any of these cases.

4 Conclusion

ADIOS was designed from the observation that the API describing traditional I/O to the file system could be abstracted to describe more complicated data movement. Since applications almost always read and/or write data to storage it becomes straightforward to replace the traditional I/O mechanism with an abstraction layer that supports much more complex movement of data with minimal changes to the flow of execution.

In this chapter we have described the high level design of the ADIOS library as well as a description of the currently available engines. We also provided a comparative discussion on each engine and discussed their strengths, weaknesses, and where each is most suitable. To provide some concrete examples of how ADIOS has been used in practice, we described a number of experimental and simulation examples that use ADIOS in their workflow for in situ processing and visualization.

Acknowledgements This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

1. Ahrens, J., Geveci, B., Law, C.: Visualization in the paraview framework. In: Hansen, C., Johnson, C. (eds.) *The Visualization Handbook*, pp. 162–170 (2005)
2. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: Multilevel techniques for compression and reduction of scientific data—the univariate case. *Comput. Vis. Sci.* **19**(5–6), 65–76 (2018)
3. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM J. Sci. Comput.* **41**(2), A1278–A1303 (2019)
4. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: Multilevel techniques for compression and reduction of scientific data—quantitative control of accuracy in derived quantities. *SIAM J. Sci. Comput.* **41**(4), A2146–A2171 (2019)

5. Chen, J., Pugmire, D., Wolf, M., Thompson, N., Logan, J., Mehta, K., Wan, L., Choi, J.Y., Whitney, B., Klasky, S.: Understanding performance-quality trade-offs in scientific visualization workflows with lossy compression. In: 2019 IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5). IEEE (2019). <https://doi.org/10.1109/drbsd549595.2019.00006>
6. Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In: High Performance Visualization—Enabling Extreme-Scale Scientific Insight, Chapman and Hall/CRC Computational Science, pp. 357–372. Chapman and Hall/CRC (2012)
7. Childs, H., Pugmire, D., Ahern, S., Whitlock, B., Howison, M., Prabhat, Weber, G.H., Bethel, E.W.: Extreme scaling of production visualization software on diverse architectures. *IEEE Comput. Gr. Appl.* **30**(3), 22–31 (2010). <https://doi.org/10.1109/MCG.2010.51>
8. Choi, J.Y., Chang, C., Dominski, J., Klasky, S., Merlo, G., Suchyta, E., Ainsworth, M., Allen, B., Cappello, F., Churchill, M., Davis, P., Di, S., Eisenhauer, G., Ethier, S., Foster, I., Geveci, B., Guo, H., Huck, K., Jenko, F., Kim, M., Kress, J., Ku, S., Liu, Q., Logan, J., Malony, A., Mehta, K., Moreland, K., Munson, T., Parashar, M., Peterka, T., Podhorszki, N., Pugmire, D., Tugluk, O., Wang, R., Whitney, B., Wolf, M., Wood, C.: Coupling exascale multiphysics applications: Methods and lessons learned. In: 2018 IEEE 14th International Conference on e-Science (e-Science), pp. 442–452 (2018). <https://doi.org/10.1109/eScience.2018.00133>
9. Choi, J.Y., Wu, K., Wu, J.C., Sim, A., Liu, Q.G., Wolf, M., Chang, C., Klasky, S.: Icee: Wide-area in transit data processing framework for near real-time scientific applications. In: 4th SC Workshop on Petascale (Big) Data Analytics: Challenges and Opportunities in conjunction with SC13, vol. 11 (2013)
10. Di, S., Cappello, F.: Fast error-bounded lossy hpc data compression with sz. In: 2016 IEEE International Parallel and Distributed Processing Symposium (ipdps), pp. 730–739. IEEE (2016)
11. Dominski, J., Ku, S., Chang, C.S., Choi, J., Suchyta, E., Parker, S., Klasky, S., Bhattacharjee, A.: A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes. *Phys. Plasmas* **25**(7), 072308 (2018). <https://doi.org/10.1063/1.5044707>
12. Farthing, J., Budd, T., Capel, A., Cook, N., Edwards, A., Felton, R., Grifh, F., Jones, E.: Data management at jet with a look forward to iter. In: International Conference on Accelerator and Large Experimental Physics Control Systems (2006)
13. Godoy, W.F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J., Germaschewski, K., Huck, K., Huebl, A., Kim, M., Kress, J., Kurc, T., Liu, Q., Logan, J., Mehta, K., Ostrochov, G., Parashar, M., Poeschel, F., Pugmire, D., Suchyta, E., Takahashi, K., Thompson, N., Tsutsumi, S., Wan, L., Wolf, M., Wu, K., Klasky, S.: ADIOS 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX* **12**, 100561 (2020). <https://doi.org/10.1016/j.softx.2020.100561>
14. Huebl, A., Lehe, R., Vay, J.L., Grote, D.P., Sbalzarini, I.F., Kuschel, S., Sagan, D., Pérez, F., Koller, F., Bussmann, M.: Openpmd 1.0.0: A meta data standard for particle and mesh based data. (2015). <https://doi.org/10.5281/ZENODO.591699>, <https://zenodo.org/record/591699>
15. Huebl, A., Widera, R., Schmitt, F., Matthes, A., Podhorszki, N., Choi, J.Y., Klasky, S., Bussmann, M.: On the scalability of data reduction techniques in current and upcoming HPC systems from an application perspective. *Lect. Notes Comput. Sci.* **10524**(4), 15–29 (2017). https://doi.org/10.1007/978-3-319-67630-2_2
16. Klasky, S., et al.: A view from ORNL: Scientific data research opportunities in the big data age. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1357–1368. IEEE (2018)
17. Kress, J., Larsen, M., Choi, J., Kim, M., Wolf, M., Podhorszki, N., Klasky, S., Childs, H., Pugmire, D.: Comparing the efficiency of in situ visualization paradigms at scale. In: International Conference on High Performance Computing, pp. 99–117. Springer (2019)

18. Kress, J., Larsen, M., Choi, J., Kim, M., Wolf, M., Podhorszki, N., Klasky, S., Childs, H., Pugmire, D.: Opportunities for cost savings with in-transit visualization. In: ISC High Performance 2020. ISC (2020)
19. Kress, J., et al.: Loosely coupled in situ visualization: A perspective on why it's here to stay. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV2015, pp. 1–6. ACM, New York, NY, USA (2015). <https://doi.org/10.1145/2828612.2828623>
20. Liang, X., Di, S., Tao, D., Chen, Z., Cappello, F.: An efficient transformation scheme for lossy data compression with point-wise relative error bound. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 179–189. IEEE (2018)
21. Lindstrom, P.: Fixed-rate compressed floating-point arrays. *IEEE Trans. Visual Comput. Gr.* **20**(12), 2674–2683 (2014)
22. Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J.Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., et al.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concur. Comput.: Pract. Exp.* **26**(7), 1453–1473 (2014)
23. Logan, J., Ainsworth, M., Atkins, C., Chen, J., Choi, J.Y., Gu, J., Kress, J.M., Eisenhauer, G., Geveci, B., Godoy, W., Kim, M.B., Kurc, T., Liu, Q., Mehta, K.V., Ostroumov, G., Podhorszki, N., Pugmire, D., Suchtya, E.D., Thompson, N., Tugluk, O., Wan, L., Wang, R., Whitney, B., Wolf, M.D., Wu, K., Klasky, S.A.: Extending the publish/subscribe abstraction for high-performance i/o and data management at extreme scale. *Bull. IEEE Tech. Commit. Data Eng.* **43**(1) (2020)
24. Moreland, K., Sewell, C., Usher, W., Lo, L.T., Meredith, J., Pugmire, D., Kress, J., Schroots, H., Ma, K.L., Childs, H., Larsen, M., Chen, C.M., Maynard, R., Maynard, B.: Vtk-m: accelerating the visualization toolkit for massively threaded architectures. *IEEE Comput. G. Appl.* **36**, 48–58 (2016). <https://doi.org/10.1109/MCG.2016.48>
25. Seward, J.: bzip2 and libbzip2. available at <http://www.bzip.org> (1996)
26. Tao, D., Di, S., Chen, Z., Cappello, F.: Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1129–1139. IEEE (2017)
27. Vazhkudai, S.S., de Supinski, B.R., Bland, A.S., Geist, A., Sexton, J., Kahle, J., Zimmer, C.J., Atchley, S., Oral, S., Maxwell, D.E., Larrea, V.G.V., Bertsch, A., Goldstone, R., Joubert, W., Chambreau, C., Appelhans, D., Blackmore, R., Casses, B., Chochia, G., Davison, G., Ezell, M.A., Gooding, T., Gonsiorowski, E., Grinberg, L., Hanson, B., Hartner, B., Karlin, I., Leininger, M.L., Leverman, D., Marroquin, C., Moody, A., Ohmacht, M., Pankajakshan, R., Pizzano, F., Rogers, J.H., Rosenberg, B., Schmidt, D., Shankar, M., Wang, F., Watson, P., Walkup, B., Weems, L.D., Yin, J.: The design, deployment, and evaluation of the coral pre-exascale systems. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 661–672 (2018)
28. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In: Kuhlen, T., Pajarola, R., Zhou, K. (eds.) Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2011). <https://doi.org/10.2312/EGPGV/EGPGV11/101-109>
29. Yun, G., Lee, W., Choi, M., Kim, J., Park, H., Domier, C., Tobias, B., Liang, T., Kong, X., Luhmann, N., Jr., et al.: Development of kstar ece imaging system for measurement of temperature fluctuations and edge density fluctuations. *Rev. Sci. Instrum.* **81**(10), 10D930 (2010)

Ascent: A Flyweight In Situ Library for Exascale Simulations



Matthew Larsen, Eric Brugger, Hank Childs, and Cyrus Harrison

Abstract This chapter describes Ascent, a production library for in situ visualization and analysis on exascale architectures. It begins by describing the library’s focal points: minimizing encumbrance on simulation codes and enabling diverse and powerful capabilities. The chapter then describes Ascent’s abstractions, interface, and design. It concludes with success stories that highlight its capabilities: in situ visualization of a 97.8 billion element inertial confinement fusion simulation using 16,384 GPUs, delivering radiography capabilities for a Kelvin-Helmholtz simulation, and native rendering of higher-order elements.

1 Introduction

This chapter describes the Ascent library for in situ visualization and analysis. Ascent was “born in situ,” meaning that it was created for the express purpose of performing in situ processing, and all implementation decisions follow this mandate. It was designed to deliver on two goals which are often in tension: (1) minimizing encumbrance on simulation codes that incorporate Ascent, i.e., a “flyweight” design (discussed in Sect. 1.1), and (2) providing diverse and powerful capabilities, especially for modern supercomputers (discussed in Sect. 1.2). In terms of the taxonomy for in situ systems presented in the Introduction chapter:

- Ascent has an API for visualization and analysis that is directly used by a computational simulation. It is primarily controlled by specifying the desired visualization and analyses *a priori*, but it also has an option to perform human-in-the-loop via Jupyter notebooks [13].
- Ascent can be run in multiple ways, but the primary usage is for Ascent to share the same resources as the simulation code (space division), with direct access

M. Larsen (✉) · E. Brugger · C. Harrison
Lawrence Livermore National Laboratory, Livermore, CA, USA
e-mail: larsen30@llnl.gov

H. Childs
University of Oregon, Eugene, OR, USA

to the simulation code’s memory. In particular, Ascent typically does not make a copy of simulation data, but rather uses the simulation’s in-memory representation directly.

Finally, while not a focus of this chapter, Ascent is built for production use by computational simulation codes—it has documentation, examples, engages in modern software engineering practices, etc.

1.1 Flyweight Design

Since Ascent was designed to run on the same resources as a simulation code, it is critical that it minimizes its usage of those resources. In particular, Ascent aims to minimize execution time, memory usage, binary size, and integration effort.

The first two aims, minimizing execution time and memory usage, benefit from directly incorporating VTK-m [21] into Ascent. For minimizing memory, VTK-m’s data model supports many array layouts (e.g., row major versus column major, array of structures versus structures of arrays) in a zero-copy manner. In most cases, the majority of the memory needed by Ascent is only for storing intermediate results, such as the triangles produced by an isosurface algorithm. To minimize execution time, VTK-m provides native support for many-core architectures, including NVIDIA GPUs, various Intel architectures, and planned support for AMD GPUs. Further, Ascent complements VTK-m’s shared-memory parallelism with its own layer for distributed-memory parallelism based on the Message Passing Interface (MPI) library. In all, Ascent is able to perform its algorithms very quickly, since it can make full use of the underlying hardware architecture.

In terms of binary size, Ascent has far fewer dependencies than more mature community visualization tools. Not only does this reduce build complexity, but it results in smaller binary sizes, lessening the linking burden on the simulation. The only required libraries for Ascent are VTK-m and Conduit (described in Sect. 3.1). Ascent also adds its own libraries, specifically VTK-h (an MPI wrapping around VTK-m) and Flow (described in Sect. 4.1). All other libraries are optional.

Finally, Ascent’s API was designed to be simple to learn, easy to integrate for novice users, and small in overall code size. Integrations can be as short as ten lines of code. The API uses Conduit [15] to simplify describing hierarchical data, in situ actions, and to support zero-copy of simulation data. This topic is explored more in Sect. 3.

1.2 Ascent Capabilities

Ascent takes multiple approaches to deliver diverse and powerful capabilities: integrating with many technologies, supporting both visualization and analysis routines,

developing algorithms specifically designed for in situ processing, and providing support for modern supercomputers.

Ascent’s strongest component for delivering capabilities is likely through integrating many additional technologies. In terms of integrations with other visualization and I/O products, Ascent is able to produce Cinema and HDF5 files, interact with ADIOS and Catalyst, and, as previously mentioned, utilizes VTK-m directly for much of its data model and visualization algorithms. It also provides many options for integration, including bindings for controlling Ascent with C, C++, Python, and Fortan. Further, it provides options for controlling Ascent via its code API, YAML files (which can be changed dynamically at runtime), or a Jupyter notebook integration [13]. Finally, Ascent is able to integrate arbitrary code, including Python scripts and C/C++ code. This pathway also enables access to machine learning infrastructures, such as TensorFlow and PyTorch. To demonstrate this pathway concretely, Ascent was recently used in situ to train and leverage a distributed-memory Random Forest on simulation data. A successful integration of Ascent with a simulation code is described in Sect. 5.3. Specifically, this integration incorporated the Devil Ray code for ray-tracing higher order elements.

Ascent has embraced analysis alongside visualization. Noteworthy analysis examples include simulated radiographs and its query infrastructure to get quantitative results such as volumes, surfaces areas, and integrated quantities (e.g., integrate the density field to get the mass). Of course, Ascent supports typical visualization algorithms, such as slicing, isosurfacing, volume rendering, and multiple plot types.

With respect to algorithms specifically for in situ processing, Ascent has a significant investment in its “trigger” system [17], which adapts when visualization and analysis routines are executed, based on the conditions in the system. Ascent also has algorithms for extracting reduced forms of the data, such as a module for extracting Lagrangian flow [5, 23] and the aforementioned Cinema output.

Finally, Ascent has been designed from the beginning with modern supercomputers in mind. Especially because of the integration of VTK-m, Ascent is able to run efficiently on these architectures. In particular, it is able to support GPUs (including multiple GPUs per node) and multi-core CPUs.

1.3 *Organization of This Chapter*

The remainder of this chapter focuses on key elements of Ascent:

- Section 2 discusses the five abstractions for using Ascent.
- Section 3 discusses Ascent’s APIs for inputting data and for controlling execution.
- Section 4 discusses Ascent’s design.
- Section 5 discusses success stories using Ascent.
- Section 6 discusses additional resources about Ascent and related software.

2 Key Abstractions for Ascent

This section describes Ascent's primary abstractions and surveys its general capabilities. Ascent organizes its capabilities into five types of in situ tasks, which it refers to as **actions**. Its five actions are:

- **Pipelines** transform data from one form to another.
- **Scenes** render images.
- **Extracts** are used to move data out of Ascent, i.e., to file I/O or to another framework.
- **Queries** provide quantitative summarizations of the data.
- **Triggers** adapt when other actions are executed, based on the conditions of the simulation.

These actions are highly interoperable, and Ascent can employ multiple actions of the same type at once. Consider the following example of Ascent doing in situ analysis on an input data set, D , for a given cycle of a simulation. Ascent begins by applying an isosurfacing operation to D to make a new data set D' (pipeline). It then applies a trigger to D' —it takes the surface area of the isosurface (query) and compares the total area with the area from the previous time it was executed. If the total area has changed by more than 5%, then Ascent would render an image of D' (scene) and also save the original data set D to disk (extract). Going beyond this example, it is possible to have multiple triggers, arbitrary conditions for the triggers (queries or otherwise), as well as many pipelines, scenes, and extracts. Figure 1 shows an example of this, specifically two pipelines and two extracts.

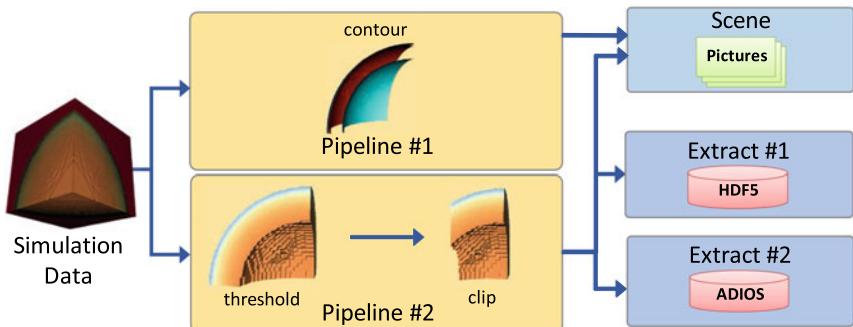


Fig. 1 An example of how Ascent actions can be combined. In this example, simulation data is published to Ascent and that data is relayed into two pipelines (labeled #1 and #2). The first pipeline applies a contour operation, while the second applies threshold and clip operations. The results of these pipelines are used in multiple ways. A scene uses both pipelines as input, while two extracts use only the second pipeline as input, outputting using two different I/O libraries

2.1 Pipelines

Pipelines allows users to describe a series of data transformations, also known as filters, to execute on simulation data. Figure 1 shows typical filters for visualization: contour, threshold, and clip. Ascent allows users to define an arbitrary number of pipelines. In terms of inputs and outputs, the input to a pipeline is either the simulation data or another pipeline, and the outputs of a pipeline can go to scenes, extracts, and queries. Further, triggers can make use of pipelines, and their relationship is discussed further in Sect. 2.5.

Notable filters currently supported by Ascent include:

- Clipping
- Contour
- Histogram
- Isovolumetric
- Particle Advection
- Statistics
- Slice
- Threshold

There are also filters that create new fields: Divergence, Gradient, Logarithm, Q-Criterion, Vector Magnitude, and Vorticity. Finally, Ascent contains two filters targeted specifically for in situ—Lagrangian Flow and Sampling—that reduce data size significantly enough that it can be saved to disk and explored post hoc.

2.2 Scenes

Scenes create images. They typically operate on the output of pipelines, but they also can work directly on the original simulation data. Scenes do not return anything to Ascent that can interact with other actions, but they do save the images they produce to disk for later inspection.

A scene is made up of “renders” and “plots”. Renders can be thought of as sub-scenes, i.e., each scene is made up of many sub-scenes (renders), which contain camera specifications, image dimensions, background and foreground colors, and annotation controls. There can be an arbitrary number of renders for a scene. Plots are the things to render. A plot consists of two things: what data to render and how to render it. The data comes from the input (usually a pipeline), and the method for rendering the data can vary. Currently, Ascent supports four plot types: pseudocolor, mesh, volume rendering, and radiograph plots (see Fig. 2). Each of these plot types contain parameters that control the input data (name of the pipeline to consume, which scalar field to operate on, etc.) and how to carry out the rendering (color tables, scalar ranges, etc.).

In terms of camera placement, Ascent uses the data set’s bounding box to create a default camera that faces the data set. Renders inherit the default camera parameters, simplifying creation of useful cameras positions. Ascent also provides simple controls to rotate the camera on the sphere circumscribing the data set. Of course, the user is free to set all camera parameters explicitly as well.

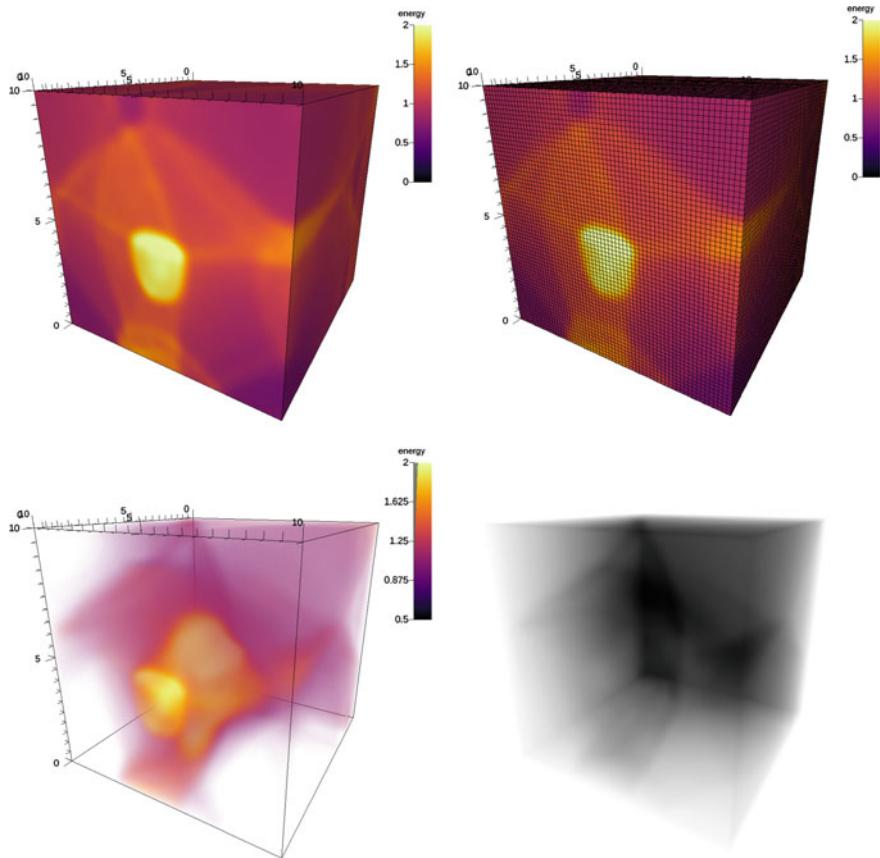


Fig. 2 Example images of Ascent’s four plot types: pseudocolor (top left), mesh (top right, which also includes a pseudocolor plot), volume rendering (bottom left), and radiograph (bottom right). The data for these images comes from CloverLeaf3D, a hydrodynamics proxy-application included with Ascent

Finally, scenes can create Cinema [6] databases, i.e., a large set of images that can be explored after the simulation.

2.3 Extracts

The extracts action covers a broad range of activities. The common theme among these activities is moving data outside Ascent. The word extract is meant to convey “extracting” data from Ascent to somewhere else. Extracts can be as simple as saving data to HDF5 files. That said, extracts are also the mechanism to integrate other tools with Ascent—a gateway to a larger workflow. This is important because Ascent

does not have the long tail of functionality that tools like ParaView or VisIt provide, which were built over decades of development. Through extracts, Ascent can pass data directly to ParaView Catalyst [8]. Another example of using extracts to provide additional functionality is the ADIOS [19] extract, which allows Ascent to link to in transit workflows.

Ascent supports connections to the Python ecosystem through its Python and Jupyter extracts [13, 14]. Python extracts execute custom analysis code provided by the user, and the Jupyter extracts allow for incoming Jupyter notebooks connections from a web browser.

We feel that the Jupyter notebook interface is an important future direction for Ascent, and for in situ as a whole. One of in situ’s greatest weaknesses is the reliance on a priori knowledge, and one strategy to mitigate this weakness is incorporating a human-in-the-loop. Through the Jupyter notebook interface, users can pause a running simulation and interact with the data. Additionally, Jupyter widgets enable fast prototyping of domain specific GUIs.

In all, currently supported extracts include:

- ADIOS
- Babel Flow [22]
- Jupyter Notebooks
- ParaView Catalyst
- Python

2.4 *Queries*

Queries enable users to ask quantitative questions. The inputs to a query can take a varied form: the simulation data directly, the data produced by a pipeline, or even a combination of multiple pipelines and simulation data. That said, typical queries are used to access the current state of the simulation and to summarize data. The results of queries are saved in Ascent’s state. These results can then be used to interact with other actions—a query’s return value can be turned into the parameter for another action (for example setting the camera position in a scene based on the result of a bounding box query) or it can be used to affect triggers.

The mechanism behind queries enables powerful operations. Queries are formed via a Python-like language that enables expressions for math operations, calling functions, and evaluating conditionals. Further, since the results of queries are stored into named identifiers, subsequent queries can build on the results of other queries, allowing for complex combinations. This also enables creating a “time history,” since Ascent can call the same query every time step and accumulate the result.

Examples of queries include:

- *cycle()*: calling a function that returns the current simulation cycle and storing it into a variable.
- *max(field('pressure'))*: the maximum value of the pressure field.

- $\text{entropy}(\text{histogram}(\text{field('gyre')}, \text{num_bins} = 128))$: calculating a histogram of the gyre field with 128 bins, and then calculating the entropy of that histogram.

2.5 Triggers

Triggers are designed to balance the tensions between cost and capturing important phenomena. In a typical scenario, in situ visualization is performed at regular intervals, e.g., every X simulation cycles or every M minutes of wall clock time. This approach creates a tension between two important goals: (1) achieving the visualizations and analyses at the right times within the simulation and (2) minimizing costs. On the one hand, performing visualization frequently maximizes the chances of having visualizations at the right times (and likely also some uninteresting times), but is costly, e.g., adding substantial overhead on top of the simulation. On the other hand, performing visualization infrequently minimizes costs, but also makes it less likely that the visualizations will occur during important phenomena.

Triggers operate in two phases: inspection and action/inaction. The purpose of the inspection phase is to determine if the action should be taken. The trigger should “fire” if the action should be taken. Further, if the inspection routine is cheap (i.e., executes quickly) and accurate (i.e., fires at the right times), then triggers can be an effective strategy. In this case, triggers can be called frequently (possibly every cycle) and still minimize cost (since the visualization in the action/inaction phase is called only when necessary) and get the right information (since the trigger is accurate). Of course, triggers make the most sense when the desired visualization is quite expensive to calculate; if the desired visualization could be calculated as quickly as the inspection, then there is no cost benefit.

A trigger is made up of two parts; for clarity, we term these two parts as trigger-condition and trigger-action. (A trigger-action is only executed if the trigger-condition is true.) In Ascent, any conditional expression can be a trigger-condition, although most trigger-conditions incorporate the queries and expressions infrastructure from Sect. 2.4. Trigger-actions utilize Ascent’s other actions: pipelines, extracts, scenes, and queries. For example, a trigger-action may be to calculate an isosurface (pipeline) and save an image (scene). In practice, triggers are often used for debugging (e.g., saving data to a file when invalid values are found) or for capturing information when some phenomenon occurs (e.g., saving an image when the maximum value of a field exceeds a threshold from the previous value).

Examples of trigger-conditions in Ascent include:

- $\text{cycle()} > 100 \text{ and } \text{cycle()} < 200$: the simulation cycle is between 100 and 200.
- $\text{max}(\text{field('pressure')}) > 100.0$: the max value of the pressure field exceeds 100.
- $\text{magnitude}(\text{max}(\text{field('pressure')}).\text{position} - \text{vector}(0, 0, 0)) > 20.0$: the distance of the max value of the pressure field from the origin exceeds 20.

2.6 Interactions Between Actions

Although some of the interactions between Ascent actions have been mentioned previously, this subsection formalizes and summarizes these interactions. The key takeaway is that connections between actions in Ascent is important and well-supported. This is not always the case; in some projects, interoperability between different types of actions can be difficult or impossible.

Figure 3 shows interactions between Ascent’s actions. It shows three types of interactions via arrow glyphs:

- Takes as input: the action at the tail of the arrow may serve as input for the action at the head of the arrow.
- Used to set parameter for action: the action at the tail of the arrow may set parameter values for the action at the head of the arrow.
- Generates this action: the action at the tail of the arrow may generate actions of the type at the head of the arrow.

Two of these interaction types are simply described. First, queries are the only action that can set parameters for other actions, and it does so for pipelines, scenes, and extracts, as well as for other queries. Second, trigger-actions are the only action that can spawn new actions, and the types it spawns are pipelines, scenes, extracts, and queries.

The last interaction type is on inputs. Simulation data is input to pipelines, scenes, extracts, and queries. Pipelines are input to the same types of actions, meaning that pipelines can serve as input to other pipelines. Further, in these cases, there can be multiple inputs, e.g., one query can have multiple pipelines as input. The last input

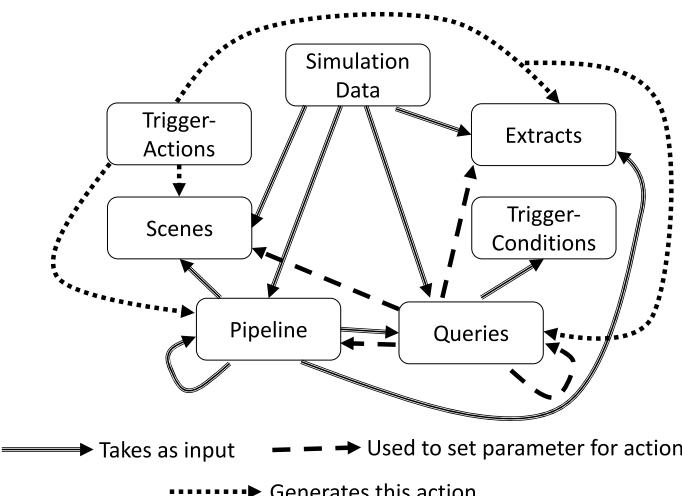


Fig. 3 Interactions between Ascent actions

involves queries and trigger-conditions. In this case, the queries would have their own inputs (i.e., simulation data or pipelines), but the trigger-condition would be firing or not based on the result of the query—the simulation data or pipeline is one step removed.

Finally, Fig. 3 shows simulation data as only being at the tails of arrows, and never the heads, i.e., nothing is ever sent to the simulation. In reality, Ascent does have a mechanism to send data back to the simulation. This is most useful for queries, but also applies to the other actions. For example, it is possible to have a scene return an image (instead of writing it to disk) and then pass that image back to the simulation (as bytes).

3 Ascent APIs

This section describes the API for Ascent. It begins, in Sect. 3.1, by describing Conduit, which is the underlying interface for passing information to Ascent. Ascent then has two main APIs, both of which are built on top of Conduit. The first API, described in Sect. 3.2, is the “data interface,” i.e., how to pass simulation data to Ascent. This API exists outside Ascent, and has its own product name—Blueprint. The second API, described in Sect. 3.3, is the “control interface,” i.e., how to instruct Ascent to carry out the actions described in Sect. 2. Finally, Sect. 3.4 discusses typical experiences when integrating with Ascent.

3.1 *Conduit: A Foundation for In-Memory Data Exchange*

This subsection describes the basics of Conduit [15], a library for describing hierarchical scientific data in C++, C, Fortran, and Python. The API was designed to be intuitive, and is inspired by JSON. While primarily used for data coupling between packages in-core, Conduit also provides easy access to serialization and I/O functions.

The primary data structure in Conduit is a Node. Nodes stores data through a key-value interface. Figure 4 shows how to store the string “value” with the key “K” into a Node.

<pre>Node n; n["K"] = "value"; n.print();</pre>	<pre>K: "value"</pre>
-------------------------------------------------	-----------------------

Fig. 4 On the left, an example of storing a string inside a Conduit Node. On the right, the YAML equivalent

```

Node n;                                dir1:
n[ "dir1 / dir2 / val1" ] = 100.5;      dir2:
n.print();                                val1: 100.5

```

Fig. 5 On the left, an example of using a hierarchical key to store a number. On the right, the YAML equivalent. The YAML has increasing indentation to indicate increasing depth in a hierarchy

```

const int size = 4;
int A[ size ] = {0, 1, 2, 3};
Node n;                                my_array: [0, 1, 2, 3]
n[ "my_array" ].set(A, size);
n.print();

```

Fig. 6 On the left, an example of setting the value of a Node to an Array. On the right, the YAML equivalent

Data in Nodes can be created and accessed through hierarchical keys, and the key string looks much like a UNIX directory structure. Key paths are completely up to the user. Figure 5 illustrates creating a Node hierarchy and storing a floating-point value in a leaf Node. In this example, several Nodes are actually created with parent child relationships defined by the key. Node *n* has a child Node *dir1*, which in turn has a child *dir2*, and the tree ends at the leaf Node *val1*, which stores the data. Nodes can contain many basic data types such as strings, floating-point values, and integers.

Nodes can also contain arrays, and the *set* method copies the values from an array into a Node. Figure 6 shows how to set a Node to the contents of an array. Alternatively, the *set_external* method copies only the pointer (i.e., a shallow copy), and any change to the underlying array would be reflected in both the original array and the data contained inside the Node. Using *set_external* is desirable for large data, such as simulation mesh data, and in environments where memory is constrained.

As already mentioned, Conduit lays the foundation for Ascent’s API. That said, Conduit enjoys widespread use outside Ascent. It is being used for simulation code coupling, effectively as a data store, among other use cases.

3.2 Mesh Blueprint: An In-Memory Mesh Description Interface

In-memory sharing of scientific data is an important use case, certainly for in situ visualization, but also for computational science in general. One barrier to in-memory sharing is how scientific data is represented. If a simulation code hands an array to an in situ visualization routine, then the routine needs to know if this array is a scalar field, coordinate information, connectivity information for an unstructured mesh, etc. Conduit alone is not helpful in addressing this barrier, as it provides a foundation for sharing data, but it provides no guidance on how to arrange data into a data model. This is where Blueprint [16] comes in.

The goal of Blueprint is to facilitate a set of shared higher-level conventions for using Conduit Nodes to hold common simulation data structures. Blueprint is able to describe a wide range of scientific data. Its design emerged through two efforts. First, by surveying existing projects that provide scientific mesh-related APIs including: ADIOS [19], Damaris [10], EAVL [20], MFEM [4], Silo, VTK [24], VTK-m [21], and Xdmf [2]. In this way, lessons learned from previous taxonomies and concrete scientific data models have been incorporated in Blueprint. Second, through discussion with simulation application teams. These discussions were about general in-memory sharing (i.e., including code-to-code sharing), but have proven to be useful for in situ visualization use cases as well. Finally, Blueprint’s mesh conventions are not a replacement for existing mesh data models or APIs. Instead, Blueprint is trying to provide a comprehensive-but-small set of options for describing meshes in-core that simplifies the process of adapting data to several existing mesh-aware APIs.

Blueprint uses four concepts to describe meshes:

- **Coordinate Sets** describe coordinate systems in 1D, 2D, and 3D, using Cartesian, spherical, or cylindrical frames of reference. Blueprint supports not only explicit coordinate sets, but also implicit representations, such as uniform and rectilinear.
- **Topologies** describe the topological structure of the mesh elements. As with coordinate sets, both implicit (e.g., uniform) and explicit (e.g., completely unstructured) topologies are supported.
- **Fields** describe the data associated with the mesh. Fields can be associated with vertices or with elements, and can be scalars or have multiple components. Additionally, Blueprint supports the description of high-order topologies and fields, which are becoming increasingly common.
- **Domain Decomposition Information** is important for distributed-memory applications. In these cases, the mesh is typically distributed among MPI tasks, into “domains,” and many algorithms need information at the boundaries of the domains to proceed. There are many ways to describe how domains abut, and Blueprint’s domain decomposition information allows for capturing the specifics.

A Blueprint data set minimally needs a topology and a coordinate system, but Blueprint supports having any number of topologies and coordinate systems, as well as any number of fields.

Listing 1 shows a Blueprint description of a uniform mesh. Blueprint’s form follows Conduit’s key-value approach. That said, the challenge in learning Blueprint is in learning the reserved words associated with concepts. In the figure’s description of the coordinate system, there are many reserved words: coordsets, type, dims, i, j, k, and uniform. Each of these have the expected meaning. The figure also creates a variable, my_coords. Blueprint does provide help in setting a data set through its protocols—methods to verify if a Conduit Node instance conforms to known conventions.

Listing 1: An example showing how to specify a 10^3 uniform grid in Blueprint.

```

Node mesh;
// 10x10x10 uniform coordinate system
mesh["coordsets/my_coords/type"]="uniform";
mesh["coordsets/my_coords/dims/i"] = 10;
mesh["coordsets/my_coords/dims/j"] = 10;
mesh["coordsets/my_coords/dims/k"] = 10;

// optional origin
mesh["coordsets/my_coords/origin/x"]=-10;
mesh["coordsets/my_coords/origin/y"]=-10;
mesh["coordsets/my_coords/origin/z"]=-10;

mesh["coordsets/my_coords/spacing/dx"]= 1.0;
mesh["coordsets/my_coords/spacing/dy"]= 1.0;
mesh["coordsets/my_coords/spacing/dz"]= 1.0;

mesh["topologies/my_topo/type"] = "uniform";
mesh["topologies/my_topo/coordset"] = "my_coords";

```

3.3 Control Interface

Ascent’s API consists of five calls:

- **open** initializes Ascent. It can optionally take arguments, for passing along information such as the MPI communicator.
- **publish** is the method that enables a simulation code to pass (“publish”) its data to Ascent.
- **execute** specifies which Ascent actions (see Sect. 2) to perform.
- **info** is the mechanism for getting data out from Ascent into the simulation.
- **close** directs Ascent to finalize execution.

Ascent usage typically consists of only four calls: open, publish, execute, and close. These calls are available with multiple language bindings: C, C++, Fortran, and Python.

Listing 2: Typical Ascent usage in C++. The code for “fill data_set” can be found in Listing 1, while the code for “fill actions” can be found in Listing 3.

```

conduit::Node data_set, actions;
// fill data_set
// fill actions
ascent::Ascent ascent;
ascent.open();
ascent.publish(data_set);

```

```
ascent.execute(actions);
ascent.close();
```

The major work in an Ascent integration is setting up the data passed to “publish” and “execute.” The data passed to “publish” (“data_set” in the example) is in the Conduit-Blueprint form, which was discussed in Sect. 3.2. To support distributed-memory processing using MPI, each MPI task describes local mesh data (one or more domains) and passes them to Ascent using a single “publish” call. Ascent operates on all domains published across all MPI Tasks as a larger coherent distributed-memory mesh. The format for the data passed to “execute” (“actions” in the example) is the subject of the remainder of this section.

There are two forms for specifying actions to Ascent. One form is to write code that specifies actions. This code would set up Conduit Nodes, and be passed in as the “actions” in the code listing above. The code would be part of the simulation code, and compiled into the simulation. The other form is to encode the information in a YAML file. In this case, the code for the simulation code would simply direct Ascent to read the YAML file and get the directions for the actions to take from that file. The advantage of this latter form is that it decouples the visualization operations from the simulation code. Specifically, the simulation can be compiled into binary form, and yet the visualization operations to be performed can still be changed. It also allows one person to do code integration (setting up publishing of data) and other people to control the visualization without having to understand the code integration.

Despite the advantages of the YAML approach, we demonstrate setting up actions with the code interface where the visualization operations are compiled into the code. The result of this code listing is to create the “actions” from Listing 2.

Listing 3: Setting up Ascent actions using the C++ bindings. This listing is broken into three parts. The first part makes a pipeline named “pl1” with one filter named “f1.” The second part makes a scene named “s1” which is connected to pipeline “pl1.” The third part tells Ascent to add the pipeline, add the scene, and then execute both.

```
// Part 1: make a pipeline
Node pipelines;
pipelines["pl1/f1/type"] = "contour";
Node contour_params;
contour_params["field"] = "pressure";
double iso_vals[2] = {0.2, 0.4};
contour_params["iso_values"].set_external(iso_vals,2);
pipelines["pl1/f1/params"] = contour_params;

// Part 2: make a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/pipeline"] = "pl1";
```

```
scenes["s1/plots/p1/field"] = "pressure";
scenes["s1/image_prefix"] = "pressure_iso";

// Part 3: prepare Conduit node for Ascent's execute() method
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_pipelines";
add_act["pipelines"] = pipelines;
Node &add_act2 = actions.append();
add_act2["action"] = "add_scenes";
add_act2["scenes"] = scenes;
actions.append()["action"] = "execute";
```

Please see Sect. 6 for links to Ascent’s documentation and more resources that provide details about getting started with Ascent and learning its APIs.

3.4 Typical Experiences Integrating Ascent

Consistent with its flyweight approach, the typical experience integrating Ascent involves a relatively short amount of time and small amount of code. While specifics vary based on the complexity of the simulation code and the amount of things to integrate, a typical integration would involve 50 to 100 lines of code. Build issues tend to be manageable, since the amount of API exposed to a simulation code is small, and since there is strong CMake support. (Native Makefile options are also available.) If a single individual had expertise in both Ascent and a simulation code, then an integration can be done in a few hours or less. Of course, the more typical situation is that an individual has expertise in either Ascent or the simulation code. In this case, additional time is needed to learn the other technology, increasing overall integration time. Finally, reiterating discussion from Sect. 3.3, most integrations are smaller since they focus on only passing data, and leave the work for specifying Ascent controls to YAML files that can be specified at run-time.

4 System Architecture

In the previous section, we described Ascent’s high-level abstractions and capabilities. This section discusses Ascent’s system architecture starting at its inner layer, Flow, and moving outward.

4.1 Flow: A Data-Type Agnostic Data-Flow Based Architecture

Ascent uses a graph-based data-flow architecture. The data-flow architecture allows Ascent to track, re-use, and cleanup intermediate results efficiently while implementing complex visualization pipelines. The design also abstracts how we register and execute operations, which simplifies adding new features.

At Ascent’s core is a simple data-flow library known as Flow, that composes and executes filters, which are the basic unit of execution in Ascent. Flow is an evolution of a Python data-flow network [11], but unlike its ancestor, Flow is a C++ library. Flow supports composing and executing directed acyclic graphs (DAGs) composed of filters [18].

There are four components to Flow:

- **Filter**: basic unit of execution.
- **Graph**: contains the filter graph and manages the adding of filters.
- **Registry**: manages the lifetime of filter results.
- **Workspace**: contains both the registry and filter graph, coordinates graph execution.

4.1.1 Flow Filters

Flow filters are the basic unit of execution inside of Ascent, and almost all functionality inside of Ascent is implemented as a Flow filter. Adding new capabilities to Ascent means wrapping that functionality inside a flow filter. Filters declare an interface, i.e., how many inputs a filter has and if there is an output, and filters are passed a set of parameters inside of a Conduit node (see Sect. 3.1). Filter inputs are tracked as arbitrary pointers and runtime features allow filters to identify and obtain concrete types for processing. In terms of Ascent actions, pipeline filters become part of a chain of data transformations and would minimally have an input data set and an output data set, while scenes and extracts become sinks that have an input data set and no output.

4.1.2 Graph

The graph is a series of Flow filters connected together into a DAG. The graph is responsible for storing filters and connections between filters. The primary graph interface supports adding filters and connecting filter ports (e.g., inputs and outputs) together.

4.1.3 Registry

The registry is a key-value data store used to manage the intermediate results of filters inside the data-flow network. Keys within the registry are reference counted, and data contained inside the registry is deleted when its reference count reaches zero. While the data associated with a key can be a pointer to any type, the majority of the data stored in the registry are Conduit Nodes or VTK-m data sets.

4.1.4 Workspace

The workspace is a container for both the graph and the registry, and the workspace is responsible for executing the DAG. Additionally, the workspace manages the lists of known filters. A filter must be registered with the workspace in order to be added to the graph, and once registered, a filter can be added to the graph by name. Flow uses a topological sort to ensure proper filter execution order, to track all intermediate results, and to provide basic memory management capabilities. During execution, the workspace provides the registry with reference counts that reflect graph connections, allowing intermediate results to be efficiently managed.

Multiple workspaces can co-exist, and in fact, Ascent uses a separate Flow workspace to evaluate expressions within the Ascent runtime.

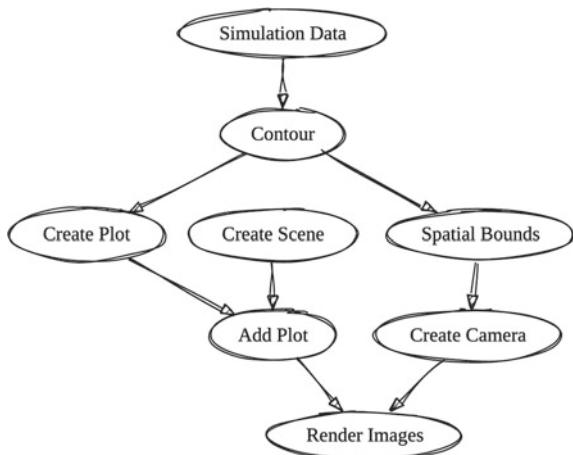
4.2 Runtime

The Ascent runtime builds on top of Flow, and the main responsibility of the runtime is to translate user actions into data-flow networks. For example, when a user describes a series of data transformations inside of a pipeline, the runtime adds the corresponding filter for each transformation to the internal Flow graph. Some actions, like the contour filter, have exactly one filter that the runtime adds when translating actions, while other actions add more than one filter to the graph. Figure 7 shows a simplified view of a data-flow network constructed by the runtime to create contours of a simulation field and render an image of the result.

Internally, the runtime maintains a list of registered filters to map user-facing API names to the corresponding Flow filters which are hidden from the user. In addition to mapping API calls to Flow filters, the filter map also specifies in what actions a filter can be used. Runtime Filters can be registered as either a “transform” or an “extract.” Transforms are only callable inside of pipelines and extracts can only be called at the end of a pipeline. Built-in functionality is registered internally with the Ascent runtime when Ascent is initialized.

The runtime also exposes the filter registration of the underlying data flow network which allows the runtime to inherit Flow’s flexibility. Simulations, or other analysis libraries, are free to register custom capabilities at runtime, and this allows outside functionality to build off the capability provided by Ascent. Just like internal filters,

Fig. 7 An example data flow network using Flow assembled by the Ascent runtime



custom filters can either be registered as transforms or extracts, and can directly connect with simulation data or consume the results of a pipeline.

The other responsibility of the runtime is to interface with the simulation through Ascent’s main API calls. The runtime consumes configuration options like MPI communicators, exception handling, and what backends (e.g., OpenMP or CUDA) to execute code on. Additionally, the simulation’s mesh data and the actions are all passed to the runtime.

4.2.1 Parallelism in Ascent

Ascent is a hybrid-parallel library, meaning that it uses both distributed-memory (e.g., MPI) and shared-memory parallelism (e.g., CUDA and OpenMP), and Ascent is primarily tightly-coupled with simulations. In the tightly-coupled paradigm, simulations control how parallelism is used, so it is imperative that Ascent’s functionality be capable of running on the same architectures using the same types of parallelism as the simulations. Ascent supports many different parallel configurations including one MPI rank per core (i.e., no shared-memory parallelism), one rank per GPU, and one rank per node. While Ascent does internally leverage shared-memory parallelism for expressions, the majority of the shared-memory parallelism comes from Ascent’s components such as VTK-m and Devil Ray.

Distributed-Memory Parallelism

Within Ascent, all MPI ranks receive the same set of actions. Since the actions are the same, all MPI ranks create and execute the same graph, meaning that all flow filters execute on all ranks. Ascent guarantees that each filter is given full control of MPI communication, and filters are free to use MPI anyway they see fit, including creating asynchronous tasks. Some filters, such as threshold, do not need to use any MPI communication, i.e., each block of data is processed independently, but other

filters, such as particle advection, use MPI to pass particles from one rank to another. For mesh data, as discussed in Sect. 3.3, each MPI rank receives data published by the simulation, and Ascent does not redistribute the data. That said, filters can change the data distribution (e.g., resampling an unstructured grid onto a uniform grid), although this can be a costly operation.

Shared-Memory Parallelism

Ascent’s main components use portable performance abstraction layers to take advantage of the different types of shared-memory parallelism on modern supercomputers. For example, VTK-m is itself a portable performance layer designed specifically for visualization and currently supports OpenMP, CUDA, and TBB. Devil Ray, while not itself a portable performance layer, uses RAJA to execute on different architectures, supporting OpenMP, CUDA, HIP, and TBB.

4.2.2 Data Set Representations

Ascent uses an internal data abstraction, called the data object, as the input and outputs of filters. The data object is responsible for transforming data from one in-memory format to another without unnecessary copies. By using this abstraction, filters can ask for whatever data set representation they need, however, the conversions between data representations are not always one-to-one and may not always result in a shallow copy.

To support multiple data set representations, there must be a common set of supported features, but not all data models Ascent uses support the same set of features. Since Ascent uses Blueprint (see Sect. 3.2) as its data interface to simulations, all other data models must sometimes be adapted to support additional features. For example, Blueprint supports multiple topologies in the same data set but VTK-m only supports a single topology. To handle this correctly within Ascent, we wrap the data sets in a container that treats each topology as individual VTK-m data sets.

5 Success Stories

5.1 *In Situ Visualization of an Inertial Confinement Fusion (ICF) Simulation*

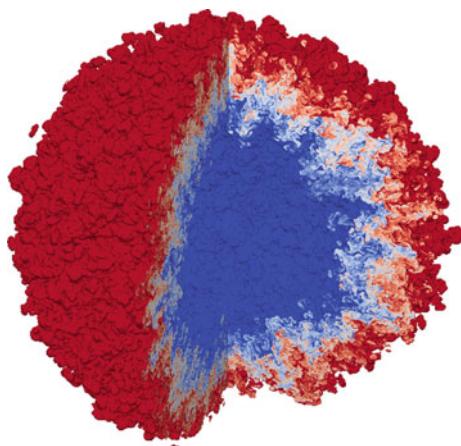
Ascent was used to visualize the results of an unprecedented 3D simulation of two-fluid mixing in a spherical geometry to better understand hydrodynamic instability and the transition to turbulence process that is important to the field of inertial confinement fusion and High Energy Density (HED) Physics. High resolution simulations of instability growth are not practical for routine use, so high resolution simulations like this help guide the development of sub-grid models that capture instability effects with much less computational cost, which are used for ICF calculations.

The simulation was run on the Lawrence Livermore National Laboratory's (LLNL) Sierra system, a 125 Petaflop peak system from IBM that has 4,320 nodes, each with 2 IBM POWER9 processors, 4 NVIDIA Tesla V100 GPUs, 320 GiB of fast memory (256 GiB DDR4 memory and 64 GiB HBM4), and 1.6TB of NVMe memory. The specific simulation was a 97.8 billion element simulation run across 16,384 GPUs on 4,096 compute nodes. The simulation application used CUDA via RAJA to run on the GPUs. The time-varying evolution of the mixing was visualized in situ with Ascent, also leveraging 16,384 GPUs. The last time step was also exported by Ascent to the parallel file system for detailed post-hoc visualization using VisIt [9]. The simulation data was accessed by Ascent directly from the GPU memory, eliminating any extra data copies. Figure 8 shows one of the many images generated in situ during this run. For this visualization, we used an isovolume filter to isolate a subset of the mixing layer, and then we used a clip filter to expose the interior of the mesh before rendering an image. For each visualization cycle, the total time used for the filters (i.e., isovolume and clip) was approximately 340 ms and the rendering time was approximately 350 ms, while the overall simulation ran for several days.

5.2 MARBL Simulation Integration

Ascent has been integrated and released with LLNL's MARBL [7] simulation code, a new next-generation multi-physics code currently under development. One of MARBL's components is a high-order finite element solver build on MFEM [3], and Ascent supports the MFEM data model. In order to leverage Ascent's visualization capabilities, Ascent refines the high-order elements to low-order (i.e., linear elements). Ascent can be activated through the simulation's input deck, and in addition to adding in situ visualization capabilities, Ascent can be used to save out the

Fig. 8 This image is of an idealized Inertial Confinement Fusion (ICF) simulation of a Rayleigh–Taylor instability with two fluids mixing in a spherical geometry. An isovolume filter was used to show only the mixing region of the heavy and light fluids, and then a clip filter was added to show the interior of the sphere



mesh and only the fields that the user specifies. Previously, MARBL only saved out full checkpoints, and only saving a subset of the data allows users to save data for post-hoc analysis at much higher temporal resolution.

As a new code, simulation validation plays an important role, and one method for simulation validation is comparing the results of experimental data to the simulated experiments. One such experiment is a radiation driven Kelvin-Helmholtz shear layer experiment [12]. The experiment captured radiographs of the instability as it was driven through the material. Comparing experimental radiographs with radiographs created from the simulation data is a useful simulation validation approach. MARBL ran a simulation of this problem using 2304 MPI ranks for 120h. Figure 9 shows a volume rendering and simulated radiograph generated as a result of this simulation.

5.3 Devil Ray Rendering

High-order finite element simulations, like MARBL, are becoming more common as supercomputing architectures continue to become more heterogeneous. One reason for this is that, by simply adjusting the polynomial order of the mesh elements, high-order simulations can tune the FLOPS/byte ratio to optimize performance on a specific architecture. Additionally, high-order finite element methods can improve the overall solution accuracy. Analysis and visualization play a key role in understanding, debugging, and communicating simulation results, but analysis and visualization frameworks have traditionally only targeted low-order meshes.

The geometry of a high-order data set is traditionally converted to a low-order approximation using element subdivision, and in order to maintain the accuracy of the solution, low-order refinement can in some cases dramatically increase the size of data representation. To make matters worse, the best refinement level is unknown, and the error propagated by the refinement is not easily understood by users. With

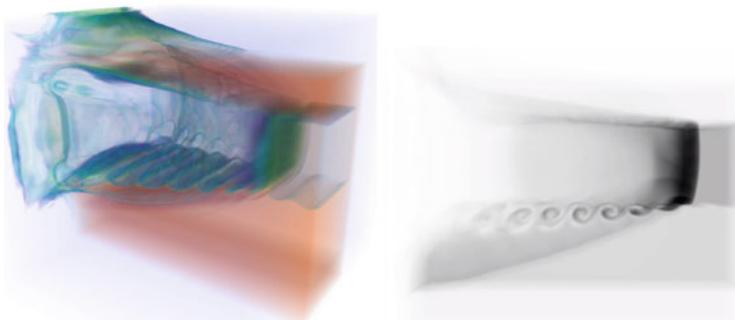


Fig. 9 A volume rendering (left) and simulated radiograph (right) created by Ascent during a Kelvin-Helmholtz simulation

the rise of in situ analysis, low-order refinement imposes additional memory and time constraints on the simulation.

By default, Ascent performs low-order refines to visualize high-order meshes, but Ascent has recently integrated Devil Ray [1], a library for natively ray tracing high-order element meshes. Ascent leverages Devil Ray to provide users an alternative to rendering images via low-order refinement. Figure 10 shows an extreme case of the differences between native support for high-order and element subdivision. The Devil Ray integration in Ascent is a first step in offering native high-order visualization support to simulations, such as MARBL.

In terms of capability, Devil Ray rendering integrated into Ascent includes volume plots, pseudocolor plot, mesh plots, isosurfacing and slicing (Fig. 11). The Devil Ray plotting capabilities are all image-based. For example, both isosurfacing and slicing are rendered through ray tracing and do not create actual geometry. Plots can be combined through image compositing.

6 Additional Resources

This section highlights additional resources for those readers who wish to learn more. As a starting point, we suggest that readers begin with Ascent and Conduit. Ascent and its dependencies have public source code repositories and documentation. The source code repositories can be found at:

- **Ascent** <https://github.com/Alpine-DAV/ascent>
- **Conduit** <https://github.com/LLNL/conduit>
- **VTK-m** <https://gitlab.kitware.com/vtk/vtk-m>
- **Devil Ray** https://github.com/LLNL/devil_ray

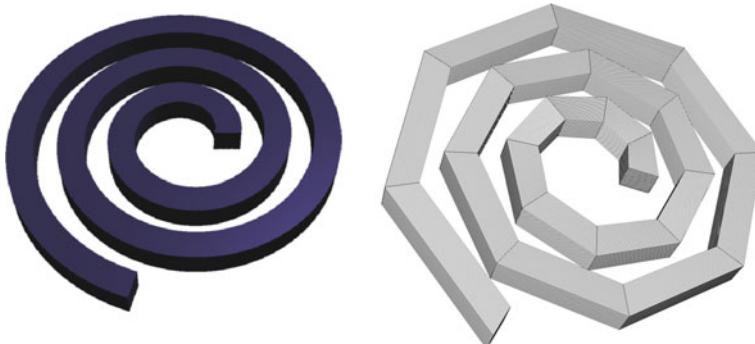


Fig. 10 Two renderings of a single 20th order hexahedral element. On the left, an image rendered by Devil Ray, and on the right, an image rendered in VisIt by subdividing the element into 9,261 low-order hexahedrons

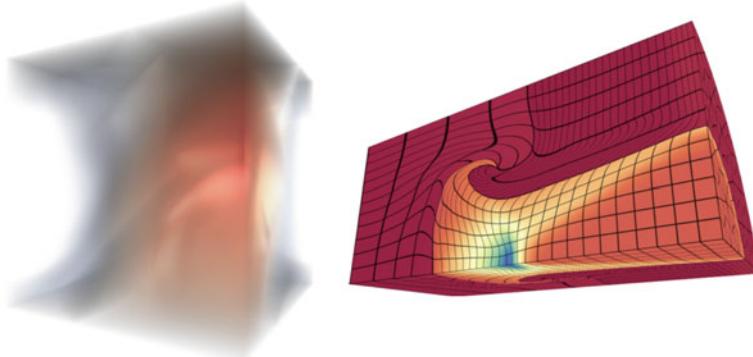


Fig. 11 Two examples of Devil Ray rendering. On the left, a volume plot of the Taylor-Green vortex, and on the right, a pseudocolor plot combined with a volume plot of the triple point problem

The Ascent, Conduit, and VTK-m documentation provide resources for both users and developers, and the documentation is a great starting point for learning more about Ascent and its enabling technologies. Additionally, the Ascent documentation contains numerous tutorials that outline basic Conduit usage, Blueprint mesh examples, and Ascent API examples. The documentation can be found at:

- **Ascent** <https://ascent.readthedocs.io>
- **Conduit** <https://llnl-conduit.readthedocs.io>
- **Conduit Blueprint** <https://llnl-conduit.readthedocs.io/en/latest/blueprint.html>
- **VTK-m** <http://m.vtk.org/documentation>

Acknowledgements This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-BOOK-814190).

References

1. Devil ray: A high-order element ray tracer. https://github.com/LLNL/devil_ray/. Accessed: 2020-1-31
2. extensible data model and format, <http://www.xdmf.org>
3. Mfem: Finite element discretization library. <https://github.com/mfem/mfem/>. Accessed: 2020-1-30
4. MFEM: Modular finite element methods (2017), <http://mfem.org>
5. Agranovsky, A., Camp, D., Garth, C., Bethel, E.W., Joy, K.I., Childs, H.: Improved Post Hoc Flow Analysis via Lagrangian Representations. In: Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV), pp. 67–75. Paris, France (2014)

6. Ahrens, J., et al.: An image-based approach to extreme scale in situ visualization and analysis. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 424–434. SC ’14. IEEE Press, Piscataway, NJ, USA (2014). <https://doi.org/10.1109/SC.2014.40>
7. Alexander, F., Almgren, A., Bell, J., Bhattacharjee, A., Chen, J., Colella, P., Daniel, D., DeSlippe, J., Diachin, L., Draeger, E., et al.: Exascale applications: skin in the game. *Philos. Trans. R. Soc. A* **378**(2166), 20190056 (2020)
8. Ayachit, U., et al.: Paraview catalyst: Enabling in situ data analysis and visualization. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, pp. 25–29. ACM (2015)
9. Childs, H., et al.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In: High Performance Visualization—Enabling Extreme-Scale Scientific Insight, pp. 357–372. CRC Press/Francis–Taylor Group (2012)
10. Dorier, M., Sisneros, R., Peterka, T., Antoniu, G., Semeraro, D.: Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In: 2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV), pp. 67–75 (2013). <https://doi.org/10.1109/LDAV.2013.6675160>
11. Harrison, C., Navratil, P., Moussalem, M., Jiang, M., Childs, H.: Efficient dynamic derived field generation on many-core architectures using python. In: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, p. 583–592. SCC ’12, IEEE Computer Society, USA (2012). <https://doi.org/10.1109/SC.Companion.2012.82>
12. Hurricane, O., Hansen, J., Robey, H., Remington, B., Bono, M., Harding, E., Drake, R., Kuranz, C.: A high energy density shock driven kelvin-helmholtz shear layer experiment. *Phys. Plasmas* **16**(5), 056305 (2009)
13. Ibrahim, S., Stitt, T., Larsen, M., Harrison, C.: 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, pp. 44–48. ISAV ’19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3364228.3364232>
14. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C.: Jupyter notebooks - a publishing format for reproducible computational workflows. In: Loizides, F., Schmidt, B. (eds.) Positioning and Power in Academic Publishing: Players, Agents and Agendas, pp. 87–90. IOS Press (2016). <https://doi.org/10.3233/978-1-61499-649-1-87>
15. Laboratory, L.L.N.: Conduit: Simplified data exchange for hpc simulations (2017., <https://llnl-conduit.readthedocs.io>)
16. Laboratory, L.L.N.: Conduit: Simplified data exchange for hpc simulations - conduit blueprint (2017). <https://llnl-conduit.readthedocs.io/en/latest/blueprint.html>
17. Larsen, M., Woods, A., Marsaglia, N., Biswas, A., Dutta, S., Harrison, C., Childs, H.: A Flexible System for In Situ Triggers. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), pp. 1–6. Dallas, TX (2018)
18. Larsen, M., et al.: 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, pp. 42–46. ISAV’17, ACM, New York, NY, USA (2017). <https://doi.org/10.1145/3144769.3144778>
19. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible io and integration for scientific codes through the adaptable io system (adios). In: Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, pp. 15–24. CLADE ’08, ACM, New York, NY, USA (2008). <https://doi.org/10.1145/1383529.1383533>
20. Meredith, J.S., Ahern, S., Pugmire, D., Sisneros, R.: EAVL: The Extreme-scale Analysis and Visualization Library. In: Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2012)
21. Moreland, K., et al.: VTK-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Comput. Gr. Appl. (CG&A)* **36**(3), 48–58 (2016)

22. Petruzza, S., Treichler, S., Pascucci, V., Bremer, P.T.: Babelflow: An embedded domain specific language for parallel analysis and visualization. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 463–473. IEEE (2018)
23. Sane, S., Childs, H., Bujack, R.: An Interpolation Scheme for VDVP Lagrangian Basis Flows. In: Eurographics Symposium on Parallel Graphics and Visualization (EGPGV), pp. 109–118. Porto, Portugal (2019)
24. Schroeder, W.J., Martin, K.M., Lorensen, W.E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In: Proceedings of Seventh Annual IEEE Visualization '96, pp. 93–100 (1996)

The SENSEI Generic *In Situ* Interface: Tool and Processing Portability at Scale



**E. Wes Bethel, Burlen Loring, Utkarsh Ayachit, David Camp,
Earl P. N. Duque, Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress,
Patrick O’Leary, David Pugmire, Silvio Rizzi, David Thompson,
Gunther H. Weber, Brad Whitlock, Matthew Wolf, and Kesheng Wu**

Abstract One key challenge when doing *in situ* processing is the investment required to add code to numerical simulations needed to take advantage of *in situ* processing. Such instrumentation code is often specialized, and tailored to a specific *in situ* method or infrastructure. Then, if a simulation wants to use other *in situ* tools, each of which has its own *bespoke* API [4], then the simulation code team will quickly become overwhelmed with having a different set of instrumentation APIs, one per *in situ* tool or method. In an ideal situation, such instrumentation need happen only once, and then the instrumentation API provides access to a large diversity of tools. In this way, a data producer’s instrumentation need not be modified if the user desires to take advantage of a different set of *in situ* tools. The SENSEI generic *in situ* interface addresses this challenge, which means that SENSEI-instrumented codes enjoy the benefit of being able to use a diversity of tools at scale, tools that include Libsim, Catalyst, Ascent, as well as user-defined methods written in C++ or Python. SENSEI has been shown to scale to greater than 1M-way concurrency on

E. W. Bethel (✉) · B. Loring · D. Camp · J. Gu · G. H. Weber · M. Wolf · K. Wu
Lawrence Berkeley National Laboratory, Berkeley, CA, USA
e-mail: ewbethel@lbl.gov

B. Loring
e-mail: loring@lbl.gov

D. Camp
e-mail: DCamp@lbl.gov

J. Gu
e-mail: jgu@lbl.gov

G. H. Weber
e-mail: ghweber@lbl.gov

K. Wu
e-mail: kwu@lbl.gov

U. Ayachit · P. O’Leary · D. Thompson · M. Wolf
Kitware, Inc., Clifton Park, NY, USA
e-mail: utkarsh.ayachit@kitware.com

P. O’Leary
e-mail: patrick.oleary@kitware.com

HPC platforms, and provides support for a rich and diverse collection of common scientific data models. This chapter presents the key design challenges that enable tool and processing portability at scale, some performance analysis, and example science applications of the methods.

1 Introduction and Overview

A fact of life in *in situ* processing is the need to add instrumentation code to data producers, such as numerical simulations, in order to invoke tools for *in situ* processing. Over the years, tools like Libsim and Catalyst have evolved to include *in situ* APIs, but these are incompatible with one another. As a result, an application that wants to use both Libsim and Catalyst would need to have tool-specific instrumentation. This problem compounds as we consider the use of more and more tools, tools that include not only visualization, but also those for statistical analysis, machine learning, preparation of derived data products, among others.

The SENSEI generic *in situ* interface project has focused on solving this problem of *tool portability* in a direct way. Its key objective is to make it possible for a SENSEI-instrumented data producer, such as a numerical simulation, to make use of any of a number of different external tools and applications for *in situ* processing, and to do so without requiring any instrumentation code changes when going from one tool to another. This concept may be thought of as “*tool portability*”, or more colloquially as “*write once, run everywhere*”. A related concept, *proximity portability*, refers to the notion of being able to run either *in situ* on the same set of nodes, or *in transit* on different set of nodes, and also without any instrumentation changes.

D. Thompson
e-mail: david.thompson@kitware.com

J. Kress · D. Pugmire · M. Wolf
Oak Ridge National Laboratory, Oak Ridge, TN, USA
e-mail: james@jameskress.com

D. Pugmire
e-mail: pugmire@ornl.gov

E. P. N. Duque · B. Whitlock · M. Wolf
Intelligent Light, Rutherford, NJ, USA
e-mail: epd@ilight.com

B. Whitlock
e-mail: bjw@ilight.com

N. Ferrier · J. Insley · S. Rizzi · M. Wolf
Argonne National Laboratory, Lemont, IL, USA
e-mail: nferrier@anl.gov

J. Insley
e-mail: insley@anl.gov

In this chapter, we focus on the design and implementation issues for the purposes of achieving tool portability. One central idea is the design of the SENSEI *in situ* interface itself (Sect. 2), which includes a solution to a challenging data modeling problem. We present several examples that illustrate tool portability (Sect. 3), explore the costs of *in situ* processing using this generic interface at scale (Sect. 4), and illustrate its application to specific science applications (Sect. 5).

The SENSEI project website¹ provides direct access to the SENSEI interface source code, documentation, code examples, and other project-related information.

2 The SENSEI Generic *In Situ* Interface Design

Given the high level objective of tool portability—being able to have a SENSEI-instrumented code connect *in situ* with various tools like Libsim, Catalyst, or custom analysis codes—we identify three design considerations.

First, if a simulation is instrumented with SENSEI, it should be able to use any of the different runtimes transparently, without any coding changes needed on the simulation side to use a different tool. In other words, in an ideal world, once a simulation has been instrumented with SENSEI, then any effort needed to leverage any other *in situ* tool should occur outside the simulation instrumentation code.

Second, if an analysis routine works with SENSEI, it should be portable, in the specific sense that it should be a straightforward process to move that piece of analytics to a different scientific simulation that uses SENSEI. The porting concerns should be at the level of data management (specifying the change in names of variable arrays), instead of wholesale rewriting of code.

Third is the desire to simplify the creation of *in situ* methods and tools for simulation scientists, data analysts, and visualization experts. This concept is related to the tool portability objective, but it is also worth mentioning separately. Given that there exist multiple *in situ* frameworks, each with its own capabilities, advantages, and expected coding patterns, it is quite challenging for simulation scientists to instrument their code to each of the frameworks separately. The same idea applies for *in situ* tool/method developers as they consider which of the *in situ* frameworks for implementing and deploying their method.

As presented in earlier work [2], the approach used to meet these design considerations focuses on two separate, but related, issues. First is solving a data model problem so that producers and consumers are able to exchange data. Second is defining an API that is suitable for use in instrumenting both data producers and consumers in a way that is representative of common design patterns and use scenarios.

S. Rizzi
e-mail: srizzi@anl.gov

¹ <http://www.sensei-insitu.org/>.

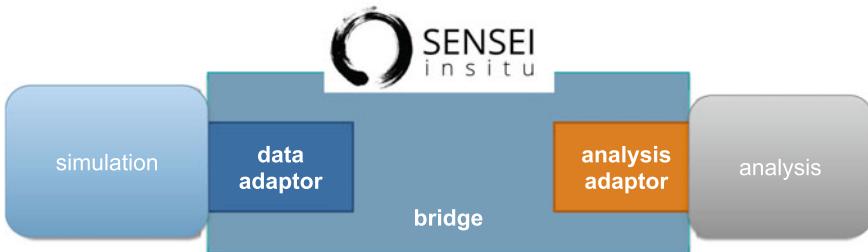


Fig. 1 The SENSEI bridge includes a *DataAdaptor*, seen by the simulation code or data producer, an *AnalysisAdaptor*, seen by the analysis code or data consumer, a bridge data model, and machinery to link the two adaptors. This image adapted from our previous work, Ayachit et al. [2]

2.1 SENSEI Data Model

A key part of the design of the common interface was a decision on a common data description model. Our choice was to extend a variant on the VTK data model. There were several reasons for this choice. The VTK data model is already widely used in applications like VisIt [5] and ParaView [16], which are important codes for the *post hoc* development of the sorts of analysis and visualization that are required *in situ*. The VTK data model has native support for a plethora of common scientific data structures, including regular grids, curvilinear grids, unstructured grids, graphs, tables, and AMR. There is also already a dedicated community looking to carry forward VTK to exascale computing [12].

Despite its many strengths, there were some key additions we added for the SENSEI model. To minimize effort and memory overhead when mapping memory layouts for data arrays from applications to VTK, we extended the VTK data model to support arbitrary layouts for multicomponent arrays through a new API called *generic arrays* [6]. Through this work, this capability has been back-ported to the core VTK data model. VTK now has native support for the commonly encountered *structure-of-arrays* and *array-of-structures* layouts utilizing zero-copy memory techniques.

2.2 SENSEI Interface

The SENSEI interface is comprised of three components, which are shown in Fig. 1. The *data adaptor* performs a mapping from the simulation data model to the VTK data model. The *analysis adaptor* performs a mapping from the VTK data model to that used by the *in situ* analysis methods. The bridge links together the data adaptor and the analysis adaptor, and provides the API that a simulation uses to trigger the invocation of the *in situ* methods. In this design and implementation, the VTK data model is the bridge data model between producer and consumer.

The data adaptor defines an API to fetch the simulation data packaged as VTK data objects. The analysis adaptor uses this API to access the data to pass to the analysis method. To instrument a simulation code for SENSEI, one has to provide a concrete implementation for this data adaptor API. The API treats connectivity and attribute array information separately, providing specific API calls for requesting each. As a result, we can avoid using compute cycles needed to map the connectivity and/or data attributes to the VTK data model unless actually needed by active analysis methods. The main parts of the originally released `sensei::DataAdaptor` API are shown in Listing 1. Subsequent releases of the `sensei::DataAdaptor` API have added methods: for exposing multiple named datasets; for fetching ghost zone and adaptive mesh refinement (AMR) covered cell masks information; and, for fetching light weight metadata useful for load balancing and planning data movement in *in transit* configurations [10].

```
namespace sensei {
class DataAdaptor : ... {
    /// provide the mesh. if structure_only is true,
    /// then only the container data object is
    /// returned without geometry or topology
    /// information.
    vtkDataObject* GetMesh(bool structure_only);
    /// add an attribute array to the mesh container,
    /// if not already added.
    bool AddArray(vtkDataObject* mesh,
                  int association,
                  const std::string& arrayname);
    /// enquire about available attribute arrays.
    unsigned int GetNumberOfArrays(int association);
    std::string GetArrayName(int association,
                           unsigned int index);
    /// release data.
    void ReleaseData();
}; }
```

Listing 1: SENSEI Data Adaptor API.

The analysis adaptor's role is to take the data adaptor and pass the data to the analysis method, doing any transformations as necessary. For a specific analysis method, the analysis adaptor is provided the data adaptor in its `Execute` method. Using the `sensei::DataAdaptor` API, the analysis adaptor can obtain the mesh (geometry, and connectivity) and attribute or field arrays necessary for the analysis method. The main elements of the analysis adaptor API is shown in Listing 2.

```

namespace sensei {
class AnalysisAdaptor : ... {
public:
/// Execute the analysis routine.
virtual int Execute(DataAdaptor* data) = 0;

/// Finalize the analysis routine
virtual int Finalize() = 0;
}; }

```

Listing 2: SENSEI Analysis Adaptor API.

2.3 Data Types Supported in the SENSEI Interface

Conceptually SENSEI expects a simulation to expose a number of “meshes” for *in situ* processing. Here a “mesh” can represent a spatially geometric data set. However, collections of non-spatially oriented data such as arrays, tables, and graphs are also supported. Irrespective of the type of data, each mesh that a simulation exposes represents a logical grouping of array based data partitioned and distributed for parallel execution. Meshes are therefore comprised of collections of distributed “blocks” such that each MPI rank has zero or more “blocks” of data. We liberally use the term “block” when referring to the subsets of a mesh which are distributed among a simulation’s MPI ranks for parallel execution. The so called “blocks” can, but do not need to, have a Cartesian structure. For instance the subsets of points allocated to each MPI rank in particle in cell (PIC) simulation are referred to as blocks of data.

The SENSEI data model makes use of VTK to internally represent simulation data. VTK is widely used in the analysis of HPC simulation data already and supports a diverse array of dataset types ranging for finite element method (FEM) datasets to graphs. VTK supports zero copy transfer of array based data and is extensible at both compile and run time. The SENSEI APIs make use of the base class in the VTK data model, `vtkDataObject`, so that any VTK dataset type including user defined types may be passed through the API without modification. A high level depiction of some of the types of data supported in the SENSEI data model are enumerated in Table 1.

2.4 SENSEI Data Producer Coding Example

To better understand the steps involved in instrumenting a simulation and analysis code with SENSEI, we present coding examples showing how to use the SENSEI interface. Here, we focus on the steps needed to instrument a data producer (e.g., a simulation) for use with SENSEI. We call this instrumentation the “bridge” code. The bridge code is not a part of SENSEI. Rather it is a concept that enables us to discuss simulation instrumentation in a general way.

Table 1 SENSEI supports a rich collection of common scientific data models, ranging from simple, like the uniform Cartesian mesh, to more complex, like AMR

Type	Description
AMR	Adaptive Mesh Refinement (AMR) is a specialization of a multi-block dataset where blocks have differing resolutions. Different organization schemes exist such as block structured overlapping and oct-tree
Multi-“block”	The general mesh type used in the SENSEI data model. All array based data passed through the simulation interface is multi-block. Blocks are used to partition subsets of the data to MPI ranks for parallel execution
Uniform Cartesian	A block type where data exists on regular Cartesian mesh. Geometry is fully implicit
Stretched Cartesian	A block type where data exists on a stretched Cartesian mesh. Geometry is defined by 3 coordinate axes
Curvilinear	A block type with hexahedral elements in a regular ordering such that element indexing is logically Cartesian. Geometry is fully explicit
Unstructured/FEM	A block type with collections of potentially mixed types of finite element method (FEM) cells with an arbitrary ordering. Geometry is full explicit
PIC/Point cloud	A block type where data exists at points in space. Implemented as unstructured mesh
Molecular	A block type specifically designed for molecular dynamics with representations of atoms and bonds between them
Tabular	A block type where data is organized as a collection of rows and columns
Graph	A block type where data is organized on nodes and edges without spatial information
Array collection	A block type consisting of an arbitrary set of arrays without any spatial information

The bridge code does three things: initializes SENSEI, including passing user provided XML that selects the data consumer; periodically invokes *in situ* processing through the SENSEI APIs as the simulation state evolves; and finalizes SENSEI. Listing 3 shows an example in C++. Conditionals protect each of the three bridge code blocks, as the code is only executed if and when the simulation determines it would like to do *in situ* processing.

```

int main(... {
    // initialize the simulation
    ...
    // initialize SENSEI
    if (doInSitu) {
        ca = sensei::ConfigurableAnalysisAdaptor::New();
        ca->Initialize(userXMLFile);
    }

    // simulation main loop
    for (int timestep=first, timestep < last; ++timestep) {

        // advance simulation
        ...

        if (doInSitu) {
            // create and initialize the data adaptor
            DataAdaptor *da = DataAdaptor::New();
            da->Initialize(...);
            // invoke in situ processing
            ca->Execute(da);
            // clean up
            da->Delete();
        }
    }

    // SENSEI shutdown and cleanup
    if (doInSitu) {
        ca->Finalize();
        ca->Delete();
    }

    // simulation specific cleanup
    ...
}

```

Listing 3: Data producer, data bridge setup and use.

2.5 SENSEI Data Consumer Coding Example

Next, in Listing 4 we present the view from an *in situ* method where we set up the analysis adaptor. This particular example is from the SENSEI histogram endpoint, which is part of the SENSEI code distribution. When instrumenting for an *in situ* analysis method, one has to provide a `sensei::AnalysisAdaptor` subclass that implements the `Execute(sensei::DataAdaptor*)` method. In the simplest case, the analysis method's data model is based on the VTK data model, in which case the `AnalysisAdaptor` subclass obtains the VTK data object using the

`sensei::DataAdaptor` and then does the necessary computation. This listing does not show computation of the histogram, only setting up the “inbound” bridge.

```
namespace sensei {
    bool Histogram::Execute(sensei::DataAdaptor* data) {
        ...
        // request light-weight mesh without connectivity info
        vtkDataObject* mesh = data->GetMesh(/*structure_only*/true);
        // request the array to histogram
        data->AddArray(mesh, this->Association, this->ArrayName);
        ...
        // * compute histogram using the array available on mesh
        // * cell or point data locally and then reduce local
        // * result across ranks using MPI Reduce.
    } }
```

Listing 4: *In Situ* data consumer, data bridge setup and use.

If the analysis method uses a different data model other than the VTK data model, then the `Execute` method needs to obtain the raw array pointers from the VTK data object and then do any needed transformations.

3 SENSEI Tool Portability

One of the main strengths of the SENSEI design and implementation is the idea of *tool portability*. The design objective is to be able to instrument a data producer code once with SENSEI, and then use any number of different *in situ* or *in transit* methods without any coding changes. This section explores the different ways that SENSEI achieves tool portability by presenting examples of use with a diverse set of *in situ* tools, ranging from user-written Python, to *in situ* endpoints from both the SENSEI and Ascent projects.

3.1 Configurable Analysis Adaptor

Reviewing briefly, in SENSEI there are data producers that are instrumented with SENSEI to invoke a `DataAdaptor` that is specific to the data producer code. There are also data consumers, or endpoints, which have associated `AnalysisAdaptors`. The purpose of the `DataAdaptor` and `AnalysisAdaptor` is to perform any necessary transformations between a native data model and the bridge data model.

In addition, SENSEI provides a *configurable analysis adaptor*, which uses an XML file to select and configure one or more back ends at run time. Run time selection of the back end via XML means one user can access Catalyst, Libsim, or

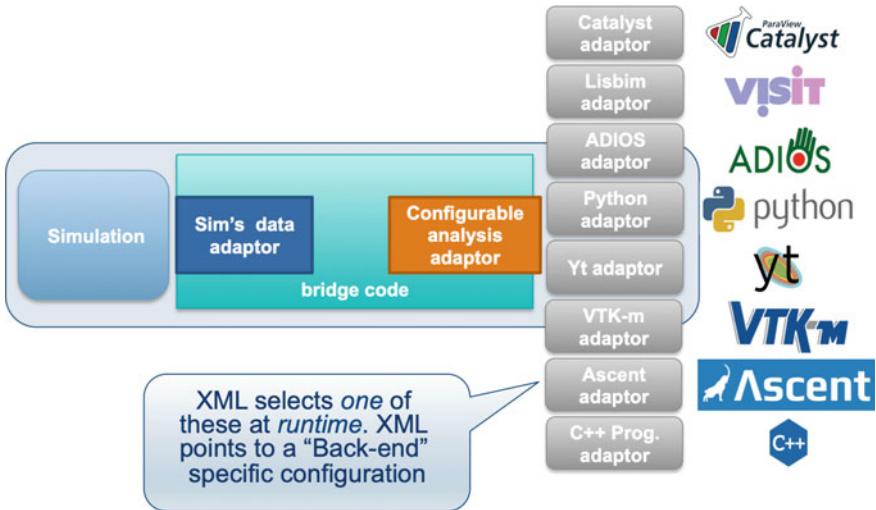


Fig. 2 Leveraging SENSEI’s configurable analysis adaptor, a single data producer has access to any number of potential *in situ* or *in transit* methods. The runtime choice of which *in situ* or *in transit* method or endpoint, along with its associated parameters, is specified in a human-readable XML configuration file. Image courtesy B. Loring

a Python-based method with no changes to the simulation code. In Fig. 2, we see a simulation data producer’s bridge code pushes data through the configurable analysis adaptor to the back end that was selected at runtime via the SENSEI configuration file.

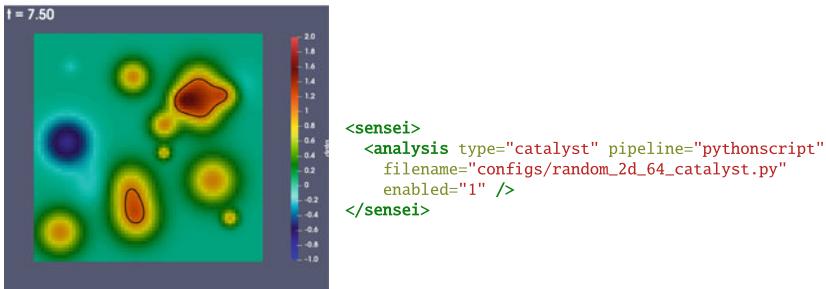
3.2 Connecting SENSEI to Libsim, Catalyst, Ascent, or ADIOS

To illustrate tool portability, Fig. 3 presents an example of a single data producer, the oscillators miniapplication, which is part of the SENSEI software distribution, coupled to three different *in situ* endpoints that perform visualization. Figures 3a, b, and c show sample images and the associated XML configuration file used to produce the image with Libsim, Catalyst, and Ascent, respectively. The runtime selection of endpoint is performed by SENSEI’s configurable analysis adaptor. This example reinforces the idea of tool portability, one of SENSEI’s design objectives, whereby no instrumentation code changes are needed on the data producer side when changing between *in situ* endpoints.

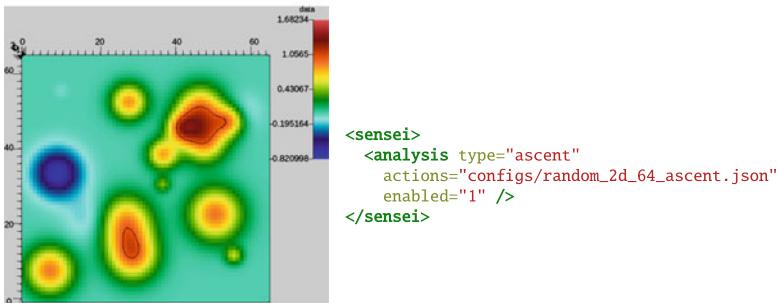
For *in transit* configurations, where data must be explicitly moved from producer ranks to consumer ranks, SENSEI can take advantage of several different potential data transport mechanisms. Continuing the example of connecting the oscillators



(a) Sample image and XML that was used to configure the run with Libsim.



(b) Sample image and XML that was used to configure the run with Catalyst.



(c) Sample image and XML that was used to configure the run with Ascent.

Fig. 3 Example of SENSEI's tool portability, where the `oscillators` miniapplication is used with three different *in situ* infrastructures: Libsim, Catalyst, and Ascent. In all cases, there are no coding changes needed to the `oscillators` miniapplication. Instead, the only difference is in the configuration file, which specifies the specific *in situ* method to be run and using what parameters. Note that these three backends all make use of their own separate configuration files as well. Image courtesy B. Loring

```
<sensei>
  <transport type="adios1" filename="random_2d_64.bp"
    method="FLEXPATH" enabled="1" />
</sensei>
```

Listing 5: This XML configures runs to use ADIOS as a data transport. The filename in this example is used to establish a connection between sender and receiver. SENSEI has the ability to switch between several different potential data transport tools at runtime via XML-based configuration files.

miniapplication to one of several potential endpoints, Listing 5 shows the SENSEI configuration file that connects the data producer to the ADIOS data transport. That data transport could then be connected to any of the endpoints shown earlier in Fig. 3 to produce exactly the same visual results as when those endpoints are invoked *in situ*.

Using this idea to implement *in transit* processing, and building on the adaptor and endpoint strategy, one can construct *in situ* and *in transit* workflows by “daisy chaining” together transport layers and endpoints that perform specific types of processing. This has been demonstrated at scale on HPC systems with several different scientific simulations and different types of endpoints [1, 2, 10].

3.3 Coupling with User-Written Python Tools

Due to the growing collection of Python-based tools and methods for diverse activities ranging from visualization to analysis and machine learning, the SENSEI project includes the ability to invoke Python-based methods for use *in situ*, including parallel Python-based methods and use of such methods at scale on HPC platforms. This section provides a high-level overview of how to invoke user-supplied Python code *in situ*, and in parallel, from a SENSEI-instrumented application, and is a consolidation of a more detailed discussion of design principles and implementation details discussed elsewhere [11].

Focusing on the user-supplied Python analysis code only, the basic idea is that the user-supplied Python code needs to contain three functions: `Initialize`, `Execute`, and `Finalize`. The `PythonAnalysis` forwards calls from SENSEI’s C++ `AnalysisAdaptor` API to those three user-provided Python functions. Those three functions are contained in the user Python file that is passed as an argument to the `PythonAnalysis` class. During initialization, the `PythonAnalysis` class reads the user Python script file on Rank 0, and then broadcasts that script to all other ranks. No specific action is required on the part of the user Python code for this to happen. The function signatures are shown in Listing 6.

The runtime selection of the user-supplied Python method is accomplished via SENSEI’s `ConfigurableAnalysis`. The `ConfigurableAnalysis` allows users

```
def Initialize():
    # your initialization code here
    return

def Execute(dataAdaptor):
    # your in situ analysis code here
    return

def Finalize():
    # your tear down code here
    return
```

Listing 6: The user-supplied Python file must contain three functions: `Initialize`, `Execute`, and `Finalize`, which are invoked by the SENSEI AnalysisAdaptor at runtime.

to select one of the analysis back ends at run time via an XML configuration file. Example XML is shown in Listing 7, which contains the name of the user-supplied Python script, along with some initialization values that are specific to the variables in that script.

```
<sensei>
  <analysis type="python" script_file="userPythonFile.py"
    enabled="1">
    <initialize_source>
      threshold=1.
      mesh='mesh'
      array='data'
      cen=1
    </initialize_source>
  </analysis>
</sensei>
```

Listing 7: The XML initialization file used by SENSEI’s `ConfigurableAnalysis` to invoke the user-supplied Python code, as well as to provide some initialization values specific to the user script. Having initialization values in the configuration file helps to avoid having hard-coded parameters inside the Python code itself

The user-supplied XML file shown in Listing 7 includes Python code that can be thought of as initialization steps. The idea is that this *initialization source* is injected by SENSEI and executed as source code in the interpreter. This channel is only used for initialization, and as a result, is only run once at start up. This channel can be used to set global variables that control execution of the user-defined analysis script. Once the *initialization source* has been run, the user-provided `Initialize` function, if present, is invoked.

During a simulation run, the simulation periodically invokes the analysis back end and will provide it with a data adaptor instance so the analysis method can

access the data it needs from the simulation. In the case of a user-supplied Python code running *in situ*, when the C++ implementation's `Execute` override is called (by the simulation), it creates a SWIG wrapped instance of the data adaptor passed to it, builds an argument list containing the wrapped adaptor instance, and invokes the user-supplied Python `Execute` function. The Python analysis code uses the wrapped data adaptor to query metadata and then selectively access data objects containing the desired set of arrays. The data adaptor returns a VTK-wrapped `vtkDataObject` instance. The user Python code makes use of VTK's `numpy_support` module to access simulation data. A complete example is show in Listing 8.

The parallel user Python code may need to make use of MPI for tasks like interprocess communication. SENSEI uses an isolated MPI communication space, which can be overridden by the simulation if desired. The communicator is accessible in the Python script via a global variable named `comm`.

Since many parallel simulations make use of ghost zones, the corresponding analysis methods will require access to them for their computations. SENSEI has adopted the ghost zone convention now used by both ParaView [9] and VisIt [17]. SENSEI's `DataAdaptor` provides methods for querying the presence of ghost zones and accessing mask arrays identifying them.

We leverage this capability to perform a conditional *in situ* computation of a time-varying data producer running in parallel on an HPC system. We configured the `oscillator` miniapplication with 256 randomly positioned and initialized harmonic oscillators on a 16384^2 plane. This configuration serves as a proxy for a simulation of a chemical reaction on a 2D substrate where the output represents the reaction rate. Data generated by the miniapplication at simulation time 1 is shown on the left of Fig. 4, including an isoline at 1.0. In the original study [11], we ran this miniapplication at four concurrency levels (512, 1024, 2048 and 4096 cores) for 100 timesteps and invoked the *in situ* method, in this case our custom Python code shown in Listing 8, at each timestep. Note that this code uses SENSEI's ghost zone mask array to support selective computations. In each invocation, we calculate the area of the domain where the reaction rate exceeds a given threshold, here set to 1.0, and accumulate the value over all timesteps. At the end of the run we use `matplotlib` to generate the x-y plot shown in the right of Fig. 4, which shows the time-evolving area computation.

3.4 *In Situ Analysis of AMR Data*

Adaptive mesh refinement (AMR) is a computational technique introduced by Berger and Colella [3]. It is used for solving systems of partial differential equations whereby computational resources are targeted to areas of the simulated domain where numerical errors have become unacceptably large. As the simulation evolves, it internally computes an estimate of numerical errors that result from the current mesh discretization. When these errors become too large, the cells with high errors are flagged to be replaced with a new discretization of higher spatial resolution, which

```

import numpy as np, matplotlib.pyplot as plt
from vtk.util.numpy_support import *
from vtk import vtkDataObject, vtkCompositeDataSet

# default values of control parameters
threshold = 0.5
mesh = ''
array = ''
cen = vtkDataObject.POINT
out_file = 'area_above.png'
times = []
area_above = []

def pt_centered(c):
    return c == vtkDataObject.POINT

def Execute(adaptor):
    # get the mesh and arrays we need
    dobj = adaptor.GetMesh(mesh, False)
    adaptor.AddArray(dobj, mesh, cen, array)
    adaptor.AddGhostCellsArray(dobj, mesh)
    time = adaptor.GetDataTime()

    # compute area above over local blocks
    vol = 0.
    it = dobj.NewIterator()
    while not it.IsDoneWithTraversal():
        # get the local data block and its props
        blk = it.GetCurrentDataObject()

        # get the array container
        atts = blk.GetPointData() if pt_centered(cen) \
            else blk.GetCellData()

        # get the data and ghost arrays
        data = vtk_to_numpy(atts.GetArray(array))
        ghost = vtk_to_numpy(atts.GetArray('vtkGhostType'))

        # compute the area above
        ii = np.where((data > threshold) & (ghost == 0))
        vol += len(ii[0])*np.prod(blk.GetSpacing())

        it.GoToNextItem()

    # compute global area
    vol = comm.reduce(vol, root=0, op=MPI.SUM)

    # rank zero writes the result
    if comm.Get_rank() == 0:
        times.append(time)
        area_above.append(vol)

def Finalize():
    if comm.Get_rank() == 0:
        plt.plot(times, area_above, 'b-', linewidth=2)
        plt.xlabel('time')
        plt.ylabel('area')
        plt.title('area Above %0.2f' % (threshold))
        plt.savefig(out_file)
    return 0

```

Listing 8: A complete, working example of user-written Python code run *in situ* by SENSEI to produce the results shown in Figure 4. This code uses SENSEI's ghost zone mask array to perform a conditional computation.

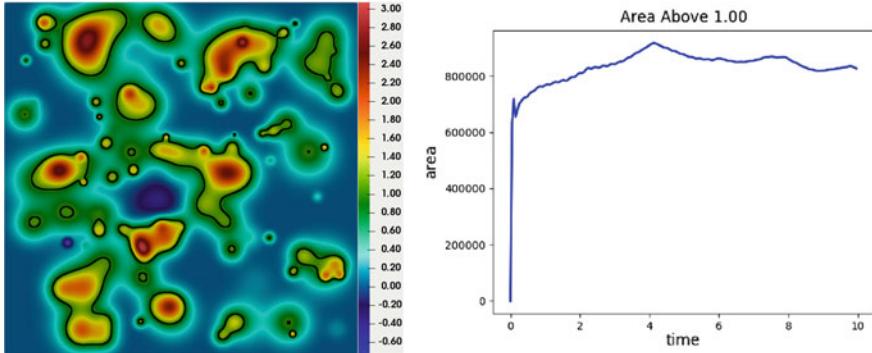


Fig. 4 The reaction rate on a planar substrate as computed in the miniapplication is shown here at simulation time 1, including an iso-line of 1 in black (left). At each time step the *in situ* analysis (Listing 8) computes the area of the substrate where the reaction rate is greater or equal to 1. The area is accumulated and plotted at the end of the run (right). This image is reprinted from our previous work Loring et al. [11]

is intended to reduce numerical error due to discretization. Cells with acceptable errors remain at their existing resolution. In overlapping block-structured AMR, the domain is decomposed into groups of blocks organized by refinement level. The blocks in higher levels fully overlap those in lower levels. Metadata accompanying the set of blocks is used to describe the hierarchical nesting of the blocks.

The SENSEI data and analysis adaptor APIs and data model includes support for AMR-capable data producers. Among these capabilities are APIs for fetching mask arrays, which indicate where mesh cells are covered by more refined cells, and for fetching metadata, which describes the hierarchical structure of the AMR data [10]. These extensions enable seamless movement and processing of AMR data by the supported *in situ* and *in transit* methods and endpoints.

Figure 5 shows two examples of *in situ* processed AMR data. The left panel shows a pseudocolored image produced using SENSEI's Python back-end in conjunction with the Yt visualization module, and the image reveals the hierarchical structure of an AMR advection miniapplication that ships with the AMReX framework [11]. In the right panel, blue lines on an extracted isosurface show the refined nature of the mesh. In both cases, the mask arrays obtained through SENSEI's data adaptor API are critical for correct processing of the refined data. Cells that are covered by refined cells in higher levels as indicated by the mask need to be discarded during *in situ* processing so that only the highest resolution data is used.

The SENSEI analysis adaptors for *in situ* and *in transit* methods and endpoints with AMR processing capabilities handle translation of AMR data from SENSEI's data model. As in the case of other SENSEI instrumented data producers, AMR simulations make use of runtime provided XML to select one of the *in situ* or *in transit* data processing, analysis, or movement methods. No changes to the simulation are required to switch between the various *in situ* and *in transit* methods and endpoints.

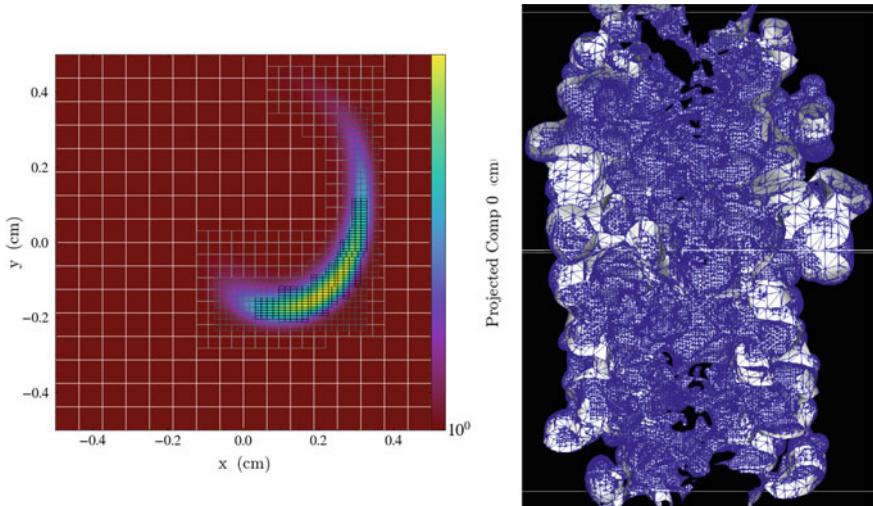
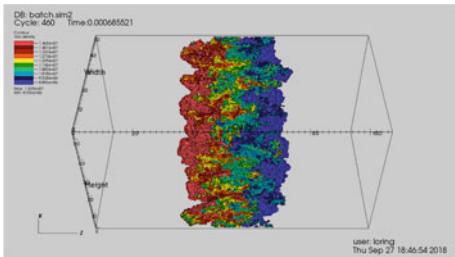


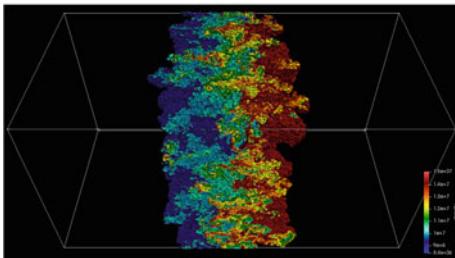
Fig. 5 Adaptive mesh refined (AMR) data has a hierarchical structure where mesh cells in regions of the simulation where numerical error is unacceptably large are refined, increasing the spatial resolution, to reduce the error. SENSEI’s data model and adaptor APIs are designed for use with AMR data producers. *Left:* In this *in situ* pseudocolored rendering, using SENSEI’s Yt back-end, overset lines show the hierarchical mesh structure. Data produced by an AMReX advection mini-application. This image is reprinted from our previous work Loring et al. [11]. *Right:* In this image an isosurface extracted with SENSEI’s Catalyst back-end is rendered. The blue lines show the refined mesh structure. Data produced with the AMReX IAMR compressible Navier-Stokes simulation. Image courtesy B. Loring

Figure 6 shows renderings of results produced by IAMR, a compressible Navier-Stokes solver implemented on top of AMReX. These IAMR runs were executed using 2048 MPI ranks on NERSC’s Cori, a Cray XC40 system. The simulation was configured using a Rayleigh-Taylor instability initial condition and set to use 3 levels with a base mesh of $256^2 \times 512$ giving an effective resolution of $1024^2 \times 2048$. The top most panel shows 10 isosurfaces rendered with Libsim, the middle panel shows the same isosurfaces rendered with Catalyst, and the bottom most panel shows them again rendered with Ascent. The SENSEI XML used to switch between the different renderers is shown next to the corresponding image. In each case, the XML file points to a library-specific configuration file that contains the configuration information in the rendering library’s native format. These library specific configurations, in the case of Libsim and Catalyst, can be generated using their respective GUIs.



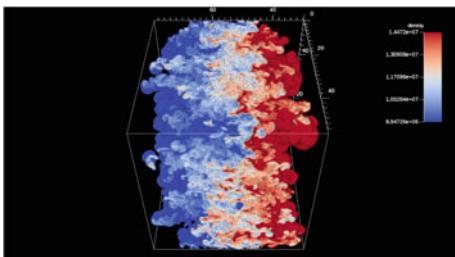
(a) Rendering the IAMR Rayleigh-Taylor simulation with Libsim

```
<sensei>
  <analysis type="libsim"
    session_file="rt_contour.session"
    enabled="1" />
</sensei>
```



(b) Rendering the IAMR Rayleigh-Taylor simulation with Catalyst

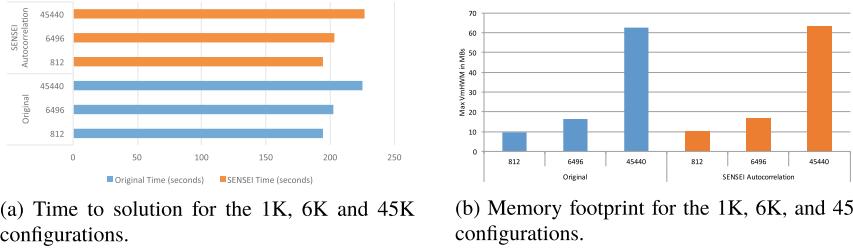
```
<sensei>
  <analysis type="catalyst"
    script_file="rt_contour.py"
    enabled="1" />
</sensei>
```



(c) Rendering the IAMR Rayleigh-Taylor simulation with Ascent

```
<sensei>
  <analysis type="ascent"
    actions="rt_contour.json"
    enabled="1" />
</sensei>
```

Fig. 6 This SENSEI data producer, the AMReX IAMR Rayleigh-Taylor simulation instrumented with the SENSEI interface, is shown here running in parallel on a large HPC platform with 3 levels of refinement. Three runs were made, the first using the SENSEI Analysis Endpoint that invokes a Libsim-based renderer (top), the second one that invokes a Catalyst-based renderer (middle), and the third using one that invokes an Ascent-based renderer (bottom). Switching between the different rendering back-ends required no changes to the simulation code. Instead, runtime provided XML, shown to the right of each figure, was used to effect the change. Images courtesy B. Loring



(a) Time to solution for the 1K, 6K and 45K configurations.

(b) Memory footprint for the 1K, 6K, and 45K configurations.

Fig. 7 Performance analysis to measure the potential impact to a code of invoking a method directly as a subroutine call compared to invoking the method through the SENSEI interface. In this particular test configuration, which entails using *in situ* processing of a structured mesh using an autocorrelation operation, there are no significant differences between the two when measuring and comparing time to solution (Fig. 7a) and memory footprint (Fig. 7b). This image is reprinted from our previous work [1]

4 SENSEI *In Situ* Performance Analysis at Scale

There are several potential different dimensions of performance analysis of *in situ* systems and methods. Of these, we focus on two specific questions in this section. The first is to examine the degree to which use of *in situ* methods impacts the performance of a simulation running at scale on an HPC system. The second is to compare the performance of a typical scientific visualization task when run in *post hoc* and *in situ* configurations so as to gain an understanding of potential gains that might result from using an *in situ* approach.

4.1 Performance Impact of *In Situ* Processing

One key question is “how much does the use of *in situ* methods impact simulation code” in terms of memory footprint and runtime. Ayachit, et al. [1] approach this problem by setting up a test matrix that consists of the several configurations aimed to revealing performance differences when invoking methods directly as compared to when invoking them using *in situ* infrastructure. The study involves the use of several different *in situ* analysis methods that have embarrassingly parallel scaling characteristics. The test battery includes runs at varying concurrencies (1K-, 6K-, and 45K-way parallel) on a large HPC platform, and include measurements of runtime and memory use.

The results of those tests, shown in Fig. 7, reveal there was no appreciable difference in either runtime or memory footprint when invoking the method directly as a subroutine compared to invoking the method through the SENSEI infrastructure. In other words, the use of *in situ* methods at scale had no appreciable difference when compared to the application just making a subroutine call. This result means the *in*



Fig. 8 From a 1M-way parallel PHASTA run on `mira` at Argonne National Laboratory, this image is a zoomed-in view of a 2D slice from 3D volume. This numerical simulation models the flow over a vertical tail-rudder assembly for a geometry that exactly matches the configuration of an ongoing wind tunnel experiment. This image is reprinted from our previous work [1]

situ interface did not “get in the way” of the application, or otherwise incur any significant cost in terms of runtime or memory footprint.

Some of the characteristics of the SENSEI interface and the study that enabled this type of result are as follows. First, this particular study focused on processing structured, 3D meshes. Such data types require relatively little metadata, and are amenable to *zero copy* sharing between the simulation and *in situ* method. The results would have likely been somewhat different with a different type of data model that incurs a larger cost in terms of metadata, such as AMR meshes. At the time of that study, the SENSEI interface provided support for *zero copy* data sharing between the simulation and *in situ* method for structure mesh data models. More recent work enables shallow- and zero-copy data sharing for other types of data whenever possible and feasible. Some of the science examples studied include unstructured meshes, where portions of the data required shallow- or deep-copies, including a large-scale CFD run at greater than 1M-way parallel, results of which are shown in Fig. 8.

This particular study from Ayachit, et al. [1] focuses on a relatively narrow set of configurations to explore performance impact at scale for *in situ* processing. Other studies have examined different dimensions of performance in *in transit* processing. For example, Morozov and Lukić [13] compare time to solution *in situ* and *in transit* work processes. Kress, et al. [8] examine cost estimation for *in transit* configurations of some common visualization rendering scenarios. This particular area, performance analysis of *in situ* and *in transit* processing, is a vibrant area of research with a long history in distributed computing.

4.2 Cost Savings of In Situ Over Post Hoc

Another dimension of performance analysis for *in situ* processing is to examine time-to-solution for a workflow consisting of a simulation producing numerical results that are then processed either *in situ* or in a *post hoc* configuration. Ayachit, et al. [1] include such a study, which shows significant differences in time-to-solution for *in situ* and *post hoc* configurations run at varying levels of concurrencies and workload

configurations. In their study, they run a data producer over 100 timesteps at a range of concurrencies: 1K-, 6K-, and 45K-way parallel. The results of this computation are then analyzed using a set of simple, embarrassingly parallel methods (histogram, point-wise autocorrelation) and visualized with ParaView/Catalyst.

In the *post hoc* configuration, data from each time step is written to disk using a file-per-processor configuration. Then, these data files are read back in from disk and processed by the analysis and visualization methods. The *post hoc* configuration uses 1/10 the number of ranks used to produce the data, since this ratio is typical of many *post hoc* workflows where there are typically far fewer ranks used for analysis than were used to compute the data.

In the *in situ* configuration, the analysis and visualization are performed at the same concurrency as the simulation. This configuration is also representative of many common *in situ* configurations: using a differing number of ranks for producer and consumer would likely entail data movement or redistribution, which is essentially an *in transit* use scenario.

The findings from this study, presented in detail in Ayachit, et al. [1], show that the expensive cost of disk I/O, both for writing and reading, is a significant impediment. It is no surprise that avoiding disk I/O altogether results in a huge performance gain. In addition, running the analysis and visualization at the same scale as the simulation results in significantly lower runtimes for that portion of the workload. The disparity between *in situ* and *post hoc* configurations grows with scale: as concurrency increases, the difference in performance between these two modalities also increases. This study makes a strong case for performance gains that result when using *in situ* processing.

5 SENSEI *In Situ* Applications to Science Problems

The next two sections present results where the SENSEI interface facilities *in situ* processing for specific science applications. The first, in Sect. 5.1, shows how *in situ* processing helps to validate mesh configurations for the Nek5000 code, where the idea is to detect problematic configurations during debug runs prior to full-scale production runs. The second, in Sect. 5.2, shows use of *in situ* processing to produce data extracts for *post hoc* analysis, where the data extract is significantly smaller in size than the underlying computational mesh.

5.1 *In Situ* Mesh Validation in Combustion Simulations

Nek5000 is a massively-parallel high-order spectral element computational fluid dynamics (CFD) solver written in Fortran and used for more than 30 years [14]. Domain scientists use it for large-scale simulations of fluid flow, thermal convection, combustion, magnetohydrodynamics, and electromagnetics. During a typical simulation,

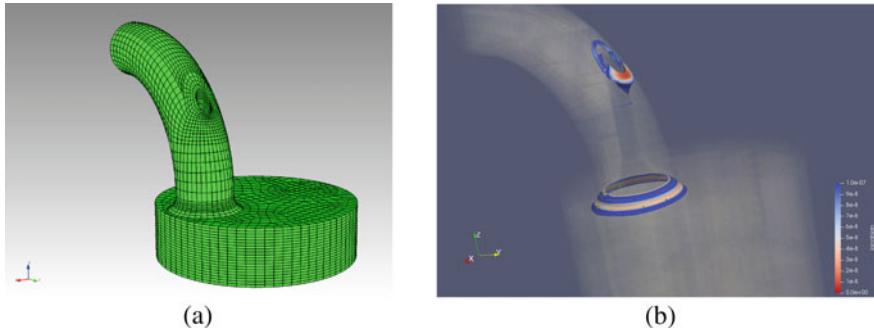


Fig. 9 **a** Simulation mesh. **b** *In situ* visualization highlighting very small and vanishing Jacobians. These images are representative reprinted from our previous work [15]

spectral (mesh) elements may deform, sometimes causing the Jacobian of some of the elements to become zero or negative. In those cases the simulation can not proceed.

Users of Nek5000 frequently encounter the problem of needing to identify the exact location of problematic individual mesh elements within the whole mesh. Employing the standard *post hoc* approach to address this problem is both time consuming and requires vast storage space. To alleviate this problem, Shudler et al. [15] demonstrated Nek5000 instrumented with SENSEI as a tool to enable users to find problematic regions of the simulation mesh *in situ*.

The left side of Fig. 9 shows the input mesh for a Nek5000 simulation of an experimental combustion engine. The hexahedral elements of the mesh may be valid (i.e., non-vanishing Jacobians) at the start of a production run, but may become highly skewed or distorted (vanishing Jacobians) during the mesh motion. The goal of this *in situ* workflow with SENSEI is to determine when and where vanishing Jacobians, if any, occur. The desired outcome is to provide a capability to users where they can validate the mesh before running any production-level simulations at scale. The right side of Fig. 9 shows an *in situ* rendering with SENSEI and Catalyst where the problematic areas of the mesh are highlighted in red (low values of the Jacobian). *In situ* detection and visualization of vanishing Jacobians helps users to locate the problematic regions of the mesh more quickly by providing visual feedback as well as a list of the problematic elements that can be fixed in preparation for large-scale production runs.

5.2 In Situ Processing and Analysis in Wind Energy Applications

One area of study in wind energy applications is the numerical modeling of wind turbines as individuals and as groups in wind farms. As individuals, some studies focus on stresses and other factors of the wind turbine elements, including the blades,

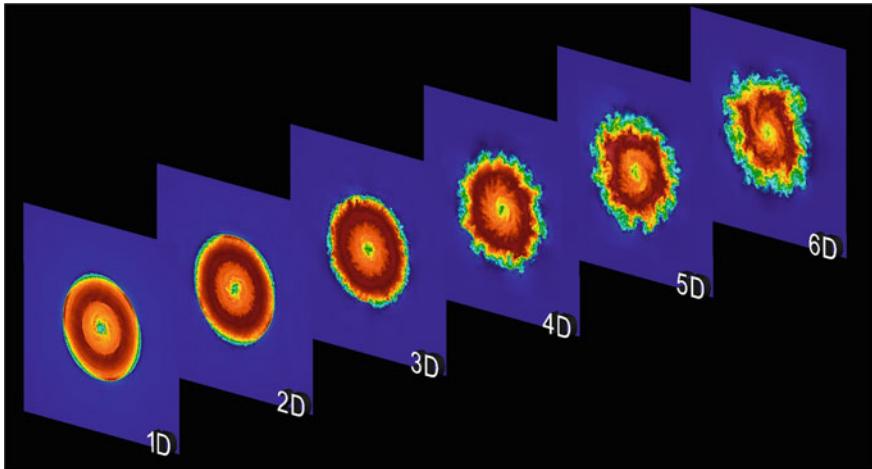


Fig. 10 Downstream cross flow cut-planes at various positions of the NREL WindPACT-1.5MW wind turbine. This image is reprinted from our previous work [7]

the nacelle, and the tower. As groups, in farms, some studies seek to analyze the interplay of turbulence and its impact when the downstream wake from one wind turbine intersects other wind turbines.

A recent study from Kirby, et al. [7] explores the use of *in situ* methods in a numerical modeling and analysis workflow. Here, the W²A²KE3D code is used to model the NREL WindPACT-1.5MW wind turbine (as well as other models, which are presented in cited work). Figure 10 demonstrates *in situ* extracts visualized in VisIt showing the wake evolution through seven cut-planes across the wake as a function of rotor diameter at fixed locations downstream of the wind turbine. Here, the *in situ* operation is to extract and produce 7 cut-planes at 400×400 resolution, each containing five flow variables and three coordinates. Additionally, a center plane of resolution 2000×400 with the same variables is extracted and is shown in the top of Fig. 11. An isocontour of velocity magnitude is shown the bottom of Fig. 11, which reveals the complex tip vortex phenomenon evolving downstream of the wind turbine.

For this study, one of the benefits reported by the authors is a significant cost savings. For their particular workflow, which entails performing analysis of wake turbulence, they realize a data reduction factor of approximately 246.2 that results when using *in situ* methods to produce data extracts at each timestep as opposed to doing I/O at full spatiotemporal resolution.

For problems such wind turbine and wind farm modeling, the authors suggest that *in situ* workflows “provide opportunities for increased productivity as the data extraction and visualization is co-produced with simulation results at run-time.” The productivity increase results from a decrease in data sizes, and provides the ability

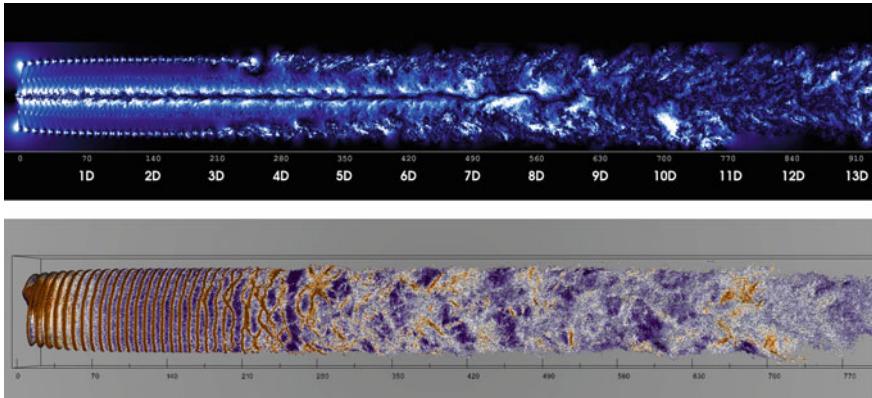


Fig. 11 (Top) Visualization of instantaneous tangential flow velocity computed by the NREL WindPACT-1.5MW wind turbine model. The image shows the downstream wake propagation, which is annotated by rotor diameter lengths. (Bottom) An isocontour of the velocity magnitude demonstrating the vortex structure evolution of the NREL WindPACT-1.5WM wind turbine. The data extracts were created *in situ* using Libsim, and were rendered in a post-processing step using VisIt. These images are representative printed from our previous work [7]

to perform analysis at high spatiotemporal resolution with only a fraction of the I/O and computational cost required by a *post hoc* workflow.

6 Conclusion

The SENSEI generic *in situ* interface has made it possible for a parallel simulation code to be instrumented once but then take advantage of any number of different potential *in situ* or *in transit* methods or tools for analysis and visualization. The primary design feature that makes this code coupling possible is the adaptor model whereby implementation-specific code that maps from native to bridge data models. The feature provides a high degree of portability between data producers and data consumers. The examples in this chapter show SENSEI being used with a diverse set of visualization and analysis endpoints, including Libsim, Catalyst, Ascent, and user-written Python code. This latter capability, being able to invoke user-written Python code both *in situ* and in parallel, opens the doors to a vast world of third-party Python tools for analysis, computer vision, and machine learning.

This system has been studied extensively at scale, including at over 1M-way concurrency. The findings show that when there is a high degree of alignment between data models, as would be the case with something like a 3D structured mesh, that the *in situ* operations are highly efficient due to SENSEI's ability to use *zero copy* wherever possible. SENSEI's rich support for a wide diversity of different scientific

data models means that it is readily usable by nearly all scientific simulation codes in existence today.

Acknowledgements This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract Nos. DE-AC02-05CH11231 and DE-AC01-06CH11357, through the grant “Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery,” program manager Dr. Laura Biven. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Argonne National Laboratory’s work was supported by and used the resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy, Office of Science User Facility supported under contract DE-AC02-06CH11357.

References

1. Ayachit, U., Bauer, A., Duque, E.P.N., Eisenhauer, G., Ferrier, N., Gu, J., Jansen, K.E., Loring, B., Lukić, Z., Menon, S., Morozov, D., O’Leary, P., Ranjan, R., Rasquin, M., Stone, C.P., Vishwanath, V., Weber, G.H., Whitlock, B., Wolf, M., Wu, K.J., Bethel, E.W.: Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’16, pp. 79:1–79:12. IEEE Press, Piscataway, NJ, USA (2016). <http://dl.acm.org/citation.cfm?id=3014904.3015010>
2. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: The SENSEI generic in situ interface. In: Proceedings of the 2nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization, ISAV ’16, pp. 40–44. IEEE Press, Piscataway, NJ, USA (2016). <https://doi.org/10.1109/ISAV.2016.13>
3. Berger, M., Colella, P.: Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.* **82**(1), 64–84 (1989)
4. Childs, H., Ahern, S., Ahrens, J., Bauer, A.C., Bennett, J., Bethel, E.W., Bremer, P.T., Brugger, E., Cottam, J., Dorier, M., Dutta, S., Favre, J., Fogal, T., Frey, S., Garth, C., Geveci, B., Godoy, W.F., Hansen, C.D., Harrison, C., Hentschel, B., Insley, J., Johnson, C., Klasky, S., Knoll, A., Kress, J., Larsen, M., Lofstead, J., Ma, K.L., Malakar, P., Meredith, J., Moreland, K., Navratil, P., O’Leary, P., Parashar, M., Pascucci, V., Patchett, J., Peterka, T., Petruzza, S., Podhorszki, N., Pugmire, D., Rasquin, M., Rizzi, S., Rogers, D.H., Sane, S., Sauer, F., Sisneros, R., Shen, H.W., Usher, W., Vickery, R., Vishwanath, V., Wald, I., Wang, R., Weberr, G.H., Whitlock, B., Wolf, M., Yu, H., Ziegler, S.B.: A terminology for in situ visualization and analysis systems. *Int. J. High Perform. Comput. Appl.* **0**(0) (2020). <https://doi.org/10.1177/1094342020935991>
5. Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J., Navratil, P.: VisIt: an end-user tool for visualizing and analyzing very large data. In: Bethel, E.W., Childs, H., Hansen, C. (eds.) High performance visualization—enabling extreme-scale scientific insight, Chapman & Hall/CRC Computational Science, pp. 357–372. CRC Press/Francis–Taylor Group, Boca Raton, FL, USA (2012). <http://www.crcpress.com/product/isbn/9781439875728>. LBNL-6320E
6. David Lonie: vtkArrayDispatch and Related Tools. <http://www.vtk.org/doc/nightly/html/VTK-7-1-ArrayDispatch.html>. <http://www.vtk.org/doc/nightly/html/VTK-7-1-ArrayDispatch.html>, last accessed Aug, 2016
7. Kirby, A.C., Yang, Z., Mavriplis, D.J., Duque, E.P., Whitlock, B.J.: Visualization and data analytics challenges of large-scale high-fidelity numerical simulations of wind energy

- applications. In: 2018 AIAA Aerospace Sciences Meeting (2018). <https://doi.org/10.2514/6.2018-1171>. <https://arc.aiaa.org/doi/abs/10.2514/6.2018-1171>
- 8. Kress, J., Larsen, M., Choi, J., Kim, M., Wolf, M., Podhorszki, N., Klasky, S., Childs, H., Pugmire, D.: Comparing the efficiency of in situ visualization paradigms at scale. In: International Conference on High Performance Computing, pp. 99–117. Springer (2019)
 - 9. Lipsa, D., Geveci, B.: Ghost and blanking (visibility) changes. <https://blog.kitware.com/ghost-and-blanking-visibility-changes/> (2015). Last accessed: Aug 2018
 - 10. Loring, B., Gu, J., Ferrier, N., Rizzi, S., Shudler, S., Kress, J., Logan, J., Wolf, M., Bethel, E.W.: Improving performance of m-to-n processing and data redistribution in in transit analysis and visualization. In: EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV). Norrköping, Sweden (2020)
 - 11. Loring, B., Myers, A., Camp, D., Bethel, E.: Python-based in situ analysis and visualization. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization—ISAV ’18. ACM Press (2018). <https://doi.org/10.1145/3281464.3281465>
 - 12. Moreland, K., Sewell, C., Usher, W., Lo, L., Meredith, J., Pugmire, D., Kress, J., Schroots, H., Ma, K.L., Childs, H., Larsen, M., Chen, C.M., Maynard, R., Geveci, B.: VTK-m: accelerating the visualization toolkit for massively threaded architectures. IEEE Computer Graphics and Applications (CG&A) **36**(3), 48–58 (2016)
 - 13. Morozov, D., Lukic, Z.: Master of puppets: cooperative multitasking for in situ processing. In: Proceedings of High-Performance Parallel and Distributed Computing (HPDC) (2016)
 - 14. Offermans, N., Marin, O., Schanen, M., Gong, J., Fischer, P., Schlatter, P., Obabko, A., Peplinski, A., Hutchinson, M., Merzari, E.: On the strong scaling of the spectral element solver Nek5000 on petascale systems. In: Proceedings of the Exascale Applications and Software Conference 2016, EASC ’16, pp. 5:1–5:10. ACM, New York, NY, USA (2016). <https://doi.org/10.1145/2938615.2938617>
 - 15. Shudler, S., Ferrier, N., Insley, J., Papka, M.E., Patel, S., Rizzi, S.: Fast mesh validation in combustion simulations through in-situ visualization. In: Childs, H., Frey, S. (eds.) Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2019). <https://doi.org/10.2312/pgv.20191105>
 - 16. Utkarsh Ayachit: The ParaView Guide: A parallel visualization application. Kitware, Inc. (2015)
 - 17. Whitlock, B.: Representing ghost data (2012). http://www.visitusers.org/index.php?title=Representing_ghost_data. Last accessed: June 2020

In Situ Solutions with CinemaScience



**David H. Rogers, Soumya Dutta, Divya Banesh, Terece L. Turton,
Ethan Stam, and James Ahrens**

Abstract Cinema is a flexible in situ visualization ecosystem that combines data extracts with viewers and analysis capabilities to support in situ, post hoc and hybrid approaches for data processing. With data extracts that include metadata, images, meshes, and other data types, Cinema databases generated in situ are a central component of post hoc analysis workflows. These workflows support visualization and exploration of data, verification and validation tasks, and leverage computer vision and statistical techniques for post hoc analysis. This chapter describes the Cinema approach, its database specification, and demonstrates its use through example workflows.

1 Introduction

As mentioned in the introductory chapter, data analysis and visualization workflows are traditionally based on a post-processing model. Data is saved at regular intervals and then visualized post hoc with a standard visualization application. With this model, data sizes—over all and for a single time step—can be quite large. This impacts the amount of data that can be saved, usually requiring significant temporal

D. H. Rogers · S. Dutta · D. Banesh · T. L. Turton (✉) · E. Stam · J. Ahrens
Los Alamos National Lab, Los Alamos, NM, USA
e-mail: tlturton@lanl.gov

D. H. Rogers
e-mail: dhr@lanl.gov

S. Dutta
e-mail: sdutta@lanl.gov

D. Banesh
e-mail: dbanesh@lanl.gov

E. Stam
e-mail: stam@lanl.gov

J. Ahrens
e-mail: ahrens@lanl.gov

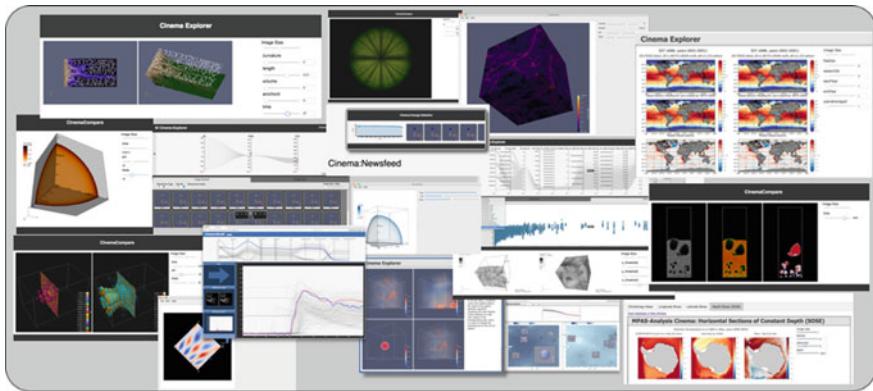


Fig. 1 Cinema is a novel way to capture, store and interact with data extracts from a wide variety of sources. It is well suited for hybrid in situ coupled with post hoc data workflows, and has been integrated into the most common visualization and analysis applications in use at extreme scale

down-sampling, and limits the ability of the scientist to effectively render and explore the data in a post hoc workflow (Fig. 1).

One consequence of moving analysis algorithms and data reduction into the in situ workflow is that the data output may change as a result of an in situ computation. In particular, in situ workflows may result in the generation of small subsets of data—*data extracts*. For example, if an algorithm finds an interesting feature at a specific time and location, the algorithm may create a data extract that allows closer inspection of a relevant portion of the data. That data extract could be a visualization, a subset of the data, a statistical representation of the data, or some other form of data extract.

In contrast to long-standing practice where a simulation outputs a standard data format (which may be specific to that simulation), these data extracts can be widely different from each other in time and scope (spatial dimensions, variables saved, mesh-based versus image-based, etc.). Thus, output from algorithm-driven in situ pipelines can be a heterogeneous set of data extracts for which complementary human-in-the-loop post hoc analysis tools and pipelines are needed to enable the domain scientist to generate scientific insights from these reduced data extracts.

Cinema provides a novel way of interacting with related sets of data extracts produced in situ. This approach, first introduced in [2], is a way of capturing, recording, analyzing and interacting with related sets of extracts from scientific data. A Cinema workflow is based on a well-defined database for these extracts, as detailed in the Cinema specification [10]. Cinema databases provide a very compact data representation that can provide many of the benefits of interacting with extremely large data, while also defining a flexible infrastructure for analyzing and interacting with the data. Cinema databases work together with Cinema writers, Cinema viewers and Cinema based analysis algorithms to form the Cinema ecosystem.

The goal of this chapter is to motivate the use of Cinema as part of an in situ workflow and to describe it sufficiently so as to allow the interested reader to understand which aspects of the Cinema ecosystem best fit their needs. This will be accomplished by a survey of Cinema functionality starting with an overview of the Cinema ecosystem, Sect. 2. The Cinema database is described in Sect. 2.2 with an overview of alternate data types in Sect. 2.5. The set of writers available for in situ export of Cinema databases is discussed in Sect. 2.3. Standard viewers can be found in Sect. 2.4 along with prototype work. Analysis capabilities are discussed in Sect. 3, showcasing examples leveraging computer vision techniques, Sect. 3.1, and statistical methods, Sect. 3.2. A task-based workflow, typical for in situ production of Cinema databases, is described in Sect. 4. Although Cinema is a general approach to data capture and analysis applicable to both experimental and simulated data, this chapter focuses on the application of Cinema to scientific simulation data, related to the context of this book.

The Cinema ecosystem is constantly evolving and adding new functionality to meet the needs of scientists. Throughout the paper, examples and references to publications demonstrate how scientists are leveraging the Cinema ecosystem to enable scientific insight. A full list of the many publications utilizing Cinema can be found on the CinemaScience website [11].

2 The Cinema Ecosystem

Cinema is an ecosystem of capabilities organized around a database of extracts. The Cinema ecosystem, shown in Fig. 2, organizes material from various sources into a database that associates data parameters with data extracts. A Cinema database includes metadata about data written to permanent storage. These databases can be written by any application, operated on by algorithms, and interactively viewed with a set of viewers and viewer components. The simplicity and flexibility of the database means that it is quite easy to integrate and experiment with Cinema as an in situ analysis workflow.

Cinema provides the following capabilities:

- A novel **image-based data extract**, the *cinema composable image* [10], which provides interactive data exploration for extreme-scale data. Interactions include: smoothly varying camera positions, so the user can interact with the data, the ability to interactively compose images so that elements can be turned on and off, and interactive post hoc recoloring of data to tune visualizations after the simulation is complete. More details can be found in Sect. 2.1.
- **Writers** that extract data to a well-specified database format, discussed in Sect. 2.2.
- **Viewers** that allow interaction with data in a movie-like or interactive application-like manner. This is shown in Sect. 2.4.
- **Algorithms** that query, analyze and filter sets of results in entirely new ways, discussed in Sect. 3.

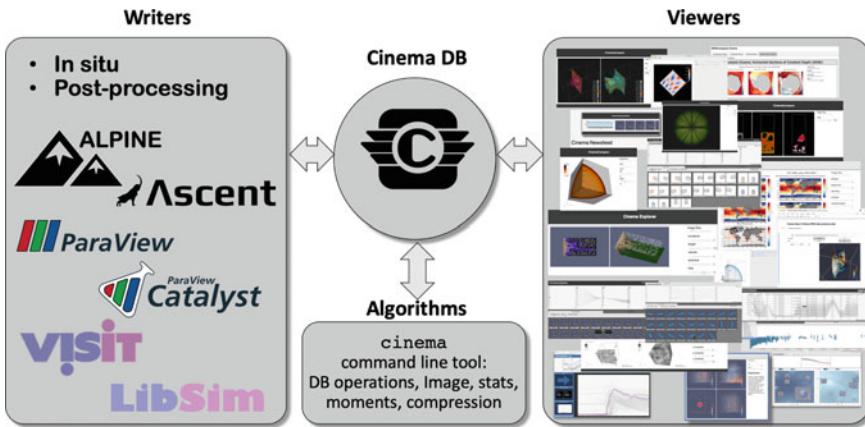


Fig. 2 Cinema is an ecosystem of capabilities organized around a database of extracts. This includes metadata about data written to permanent storage. Most common is a database of images that captures a set of rendered views of the data. These databases can be written by any application, operated on by algorithms, and interactively viewed with a set of viewers and viewer components. The simplicity and flexibility of the database means that it is easy to get started and take advantage of some of the power of Cinema. (Versions of this image have been previously used in internal ECP [14] documents and, for example, in [8, 27])

2.1 Simple Use Case: Cinema Image Databases

The application of Cinema as an image database is a simple use case of the ecosystem that illustrates its functionality. For example, a user might render images from specific camera positions for all time steps of an extreme-scale simulation. These images provide a compact representation and Cinema can be used to organize, and interact with the images via its viewers. For example, the simulation can be ‘rotated’ through a series of images captured at different spatial locations. The progression of the simulation over time can be viewed through images captured temporally. Figure 3 shows the results of exporting a Cinema image database from ParaView and viewing the images with a Cinema viewer. Sliders in the viewer allow users to rotate the data and scroll through time interactively.

The *cinema composable image*, an extension of rendered images, allows a viewer to interactively compose elements of an image and recolor them post hoc. Therefore, in addition to manipulating parameters such as camera position, a Cinema viewer can also turn elements on and off, and interactively recolor images. Figure 4 shows how different elements of a cinema composable image can be combined together to provide a more complete picture of a data set while allowing components to be viewed and hidden interactively.

The following sections of the chapter delve more deeply into the database concept, writers, and viewers that comprise the Cinema system.

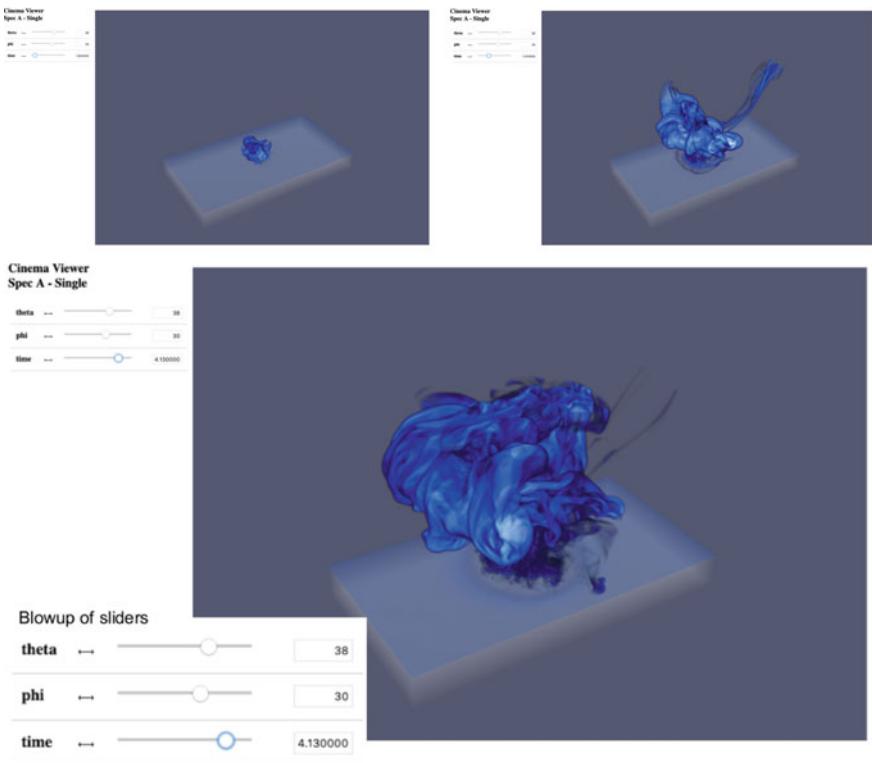


Fig. 3 A Cinema web-based viewer showing a Cinema database of images from the Deep Water Impact Ensemble Data Set [22]. The viewer enables an interactive exploration of data. In these images, the user operates a time-based slider to scroll through time steps of the simulation. At the top is an image from the beginning and middle of an asteroid impact simulation. At the bottom is an image from late in the simulation. Due to constraints on loading large data sets, this type of interaction would be impossible with raw full sized data. This interaction is a powerful feature of Cinema, demonstrating that even extreme-sized data can be explored interactively via images

2.2 The Cinema Database

At a high level, Cinema maps metadata to data extracts saved on disk or other permanent storage. A Cinema database does not encode specific meaning in the metadata—that is left up to the user. This was a deliberate choice when creating the specification, to retain simplicity at Cinema’s core, and allow flexibility for user-written applications. For consistency, tools (viewers, writers) operate on parameters and extracts across the ecosystem.

Cinema’s database specification provides simplicity and adaptability. Rows represent database entries, e.g. time steps in a simulation, and columns are the parameters and data extracts available for each entry. The minimal Cinema database is a directory that contains a required `data.csv` file. Optional data files and directories can

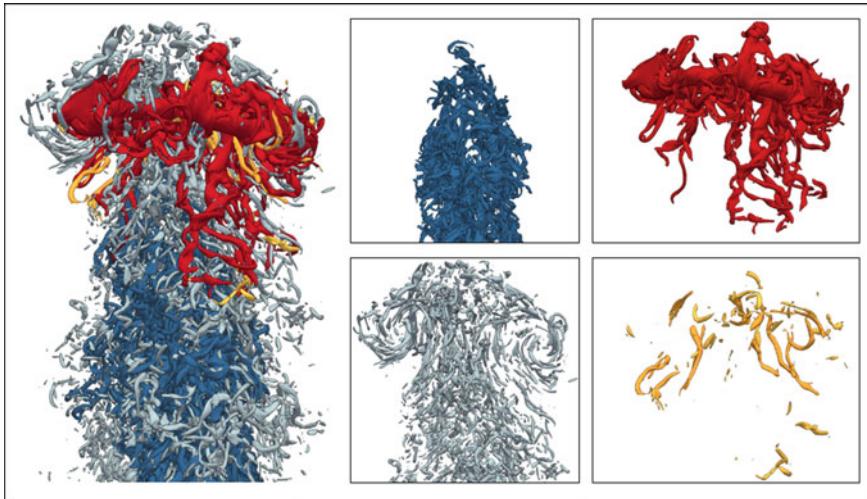


Fig. 4 These images represent elements of a cinema composable image, demonstrating how these can be composited by a Cinema viewer. The renderings show four clusters of roughly 5,000 highly turbulent regions of a computational fluid dynamics simulation. On the left is a composite image showing all four regions together. On the right are the four separate regions, each stored in a different component of a cinema composable image. Each of the clusters can be viewed independently, but also, through the cinema composable image specification, they can be combined together (composed) to show a complete picture of the data. These components can be interactively composited, producing the effect of turning elements ‘on’ and ‘off’ in any view of the data. This makes the resulting Cinema database highly interactive and explorable (Images courtesy of J. Lukasczyk, Arizona State University)

include data extracts such as images, small mesh files, other CSV files, text files, or other relevant files. The specification allows for other files to be present in the main directory, so that applications can add additional information that core Cinema tools will ignore. Typically, each row in a Cinema database maps a set of parameters to one or more data extracts. The database specification, [10], contains full details of the current specification for the interested user.

2.2.1 Fully Populated Versus Sparse Metadata

One possibility for a Cinema database is that the metadata has a fully populated cross-product of values. This can be achieved by, for example, writing an image from a set of camera positions for each time step. Table 1 is an example of a fully populated metadata database. A data set such as shown in Fig. 3 is an example of a fully populated cross-product Cinema database that can be read by a Cinema viewer designed to view a fully populated metadata database, such as Cinema:View.

A second option is that the columns are not full cross-products. This is a frequent occurrence in scientific data sets, where all values for all parameters may not be

Table 1 This example `data.csv` file is for a Cinema database containing images for a simple phi/theta camera move over two time steps. Time varies over $[0, 1]$, phi over $[0, 45]$ and theta over $[45, 90]$, and the PNG images each have a unique filename

Time	phi	theta	File
0	0	45	000.png
0	0	90	001.png
0	45	45	002.png
0	45	90	003.png
1	0	45	004.png
1	0	90	005.png
1	45	45	006.png
1	45	90	007.png

Table 2 This is a sparse database, in which data values (columns) may be missing, valued as Nan, or duplicates of other values. This is commonly seen in tables of values from, for example, experimental scientific data sets. Cinema viewers are still able to view data sets like this, through UI widgets other than continuous sliders

Time	Isovar	Isovalue	File
0			000.png
1			001.png
2	Temperature	100.0	002.png
3	Temperature	150.0	003.png

defined, and all combinations of the variables may not be defined. Consider this example, in which the variable `isovar` is defined on time step 2, but not time step 0 or 1. In this case, the Cinema database would look like the one in Table 2. The Cinema ecosystem provides a set of standard viewers (Sect. 2.4) to handle either case—sparse or full cross-product databases.

2.3 Cinema Writers

As part of an in situ workflow, Cinema image database export is available in common open source scientific visualization applications and infrastructures. ParaView [1] provides post-processing Cinema export and in situ export is available through the ParaView Catalyst [4, 15] in situ library. VisIt [9] also provides post-processing export capability. Ascent [17], a new flyweight infrastructure under development as part of the Exascale Computing Project [14], also contains Cinema export functionality. As an example, the MPAS-Ocean [24] simulation was instrumented with Catalyst’s in situ capability to export a Cinema database [20]. ParaView was used post-processing to generate Cinema databases from the Nyx [3] cosmology simu-

lation, used in [8] and as seen in Figs. 5 and 7. Common to each of these export capabilities is the ability to choose a static view, i.e., a single camera angle, or a phi-theta view with a user-specified number of steps in (ϕ, θ) . By default, the ϕ and θ steps are regularly spaced.

It is useful to note that Cinema databases can also be built post hoc. Simple Python or bash scripts can be used to organize already existing images or output plots into Cinema databases. While this chapter focuses on the *in situ* use of Cinema, Cinema databases built post hoc have been used for both simulation and experimental data sets. The Foresight framework [16] has used Cinema databases to allow scientists to interactively explore the impact of compression on simulation data. Cinema has also been used for a variety of experimental analysis workflows that output images. These include shock physics experiments [21], experimental diffraction images [29], and Bragg peak detection and tracking as discussed in [27].

Lastly, we note that most while most of the *in situ* writers work in a tightly coupled mode, Cinema can also be implemented in a loosely coupled or in transit environment such as described in various chapters in this book.

2.4 Cinema Viewers

A core concept in Cinema is the *viewer*—flexible applications that can read in any specification-compliant Cinema database and enable analysis workflows. Cinema includes three standard viewers, Cinema:Scope, Cinema:View and Cinema:Explorer that will meet the needs of most users. There is also a library of individual viewer components that can be used to build workflow-specific viewers. This section overviews each of the standard viewers, providing example use cases. The Cinema ecosystem constantly evolves to meet user needs and prototype viewer development is also discussed.

2.4.1 Cinema:Scope

Cinema:Scope is a cross-platform application built on Qt and C++. It has slider controls mapped to the database parameters. By default, Cinema:Scope loads the first set of images in a Cinema database and provides intuitive mouse controls that are mapped to `phi` and `theta`. The mouse control mapping and image set can be changed within the application. Figure 5 shows an example of a cosmology simulation [3] viewed within Cinema:Scope. The database parameters are `time`, `phi`, and `theta` where `phi` and `theta` are mapped to the mouse controls.

Cinema:Scope provides the post hoc interactive exploratory functionality needed by the scientist while avoiding the computationally expensive part in the post hoc workflow. It gives the user the feel of using a full visualization application such as ParaView or VisIt but without the overhead of rendering each image post hoc. This functionality is one of the attractive features of Cinema:Scope.

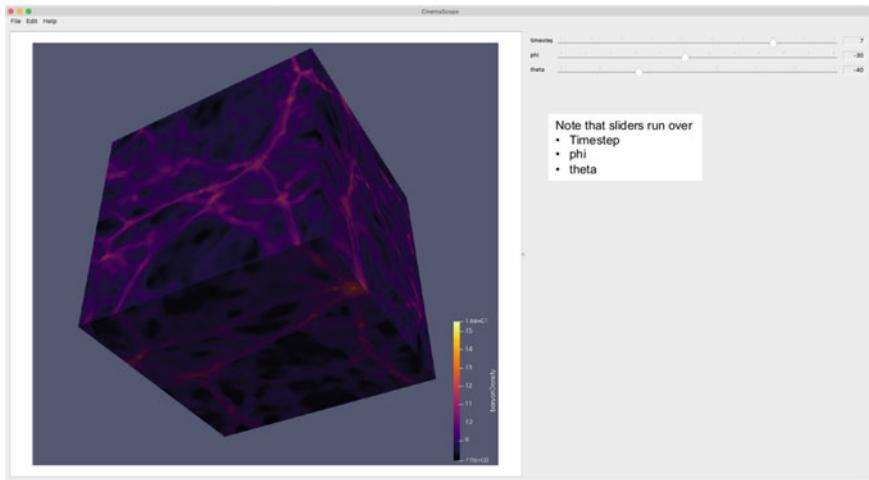


Fig. 5 CinemaScope is used to view a Nyx cosmology simulation showing the formation of dark matter halos over time. The three sliders, timestep, phi, theta, are mapped to the mouse controls to enable intuitive movement through the Cinema image database

2.4.2 Cinema:View

Cinema:View is a basic browser-based viewer used to visually explore images in a Cinema database. Cinema:View is based on JavaScript and D3. It features slider controls and can be used to view a single Cinema image database or multiple databases with common parameter sets. Figure 6 uses Cinema:View to view three ways to detect voids or bubbles in an MFIX-Exa [26] bubbling fluid bed simulation.

2.4.3 Cinema:Explorer

Also browser-based using JavaScript and D3, Cinema:Explorer leverages parallel coordinates to explore data within a Cinema database. The use of parallel coordinates for Cinema databases was first explored in [31]. Parallel coordinates are a common approach to exploring high dimensional data.

Cinema:Explorer includes a parallel coordinates view, an image spread view, and a scatterplot view. The view panel information is linked so that a selection in the parallel coordinates panel brings up the associated images in the image spread view and the scatterplot. This can be used, for example, to identify outliers or explore correlations in large data sets.

The screenshot in Fig. 7 shows Cinema:Explorer being used to query a large image database using the parallel coordinates interactive view. For this example, a Nyx cosmology Cinema database has had computer vision algorithms applied to generate image-based statistics. Cinema:Explorer displays the database parameters and statis-

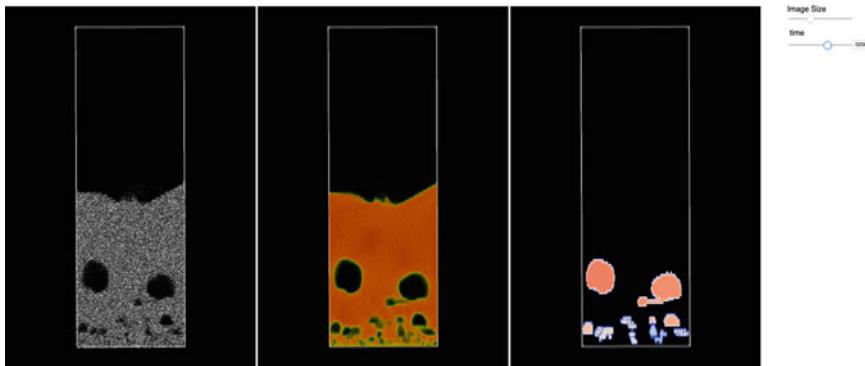


Fig. 6 Cinema:View is used to view the detection of voids (bubbles) in an MFIX-Exa bubbling fluid bed simulation. From left to right, a subset of the original data, downsampled to 5% of the data (see the chapter on *Sampling In situ*); a density field is used to find bubbles; a bubble finding algorithm using the density field calculation shows only the voids. The three views of the bubbling bed simulation can be compared over time using the slider. Image size can also be changed to fit the user browser. MFIX-Exa data courtesy of A. Almgren and J. Blaschke, Lawrence Berkeley National Laboratory; bubble images courtesy of A. Biswas, Los Alamos National Laboratory

tical quantities in parallel coordinates. Standard parallel coordinate techniques can be used to hide/show axes or select ranges on the axes. In this example, the user has selected images early in time and with low entropy. Cinema:Explorer shows all images in the database that match this query. Cinema:Explorer allows scientists to quickly view queries on data ranges across all samples extracted, making it possible to explore and compare large sets of data very quickly. This example illustrates a typical Cinema-enabled workflow: saving parameters and data extracts such as pre-rendered visualizations *in situ*, extending the analysis post hoc, and using the viewer to select, query, and explore the information within the database.

2.4.4 Jupyter-Based Viewers

Many simulation scientists use the Python scientific analysis stack and notebooks are becoming an increasingly common workflow analysis tool for simulations. NERSC has a dedicated JupyterHub to connect notebooks to HPC resources. To accommodate Python users, a prototype Jupyter notebook-based viewer is available at https://github.org/cinemascience/cinema_jnc. Figure 8 shows a WarpX [28] simulation of a plasma-driven accelerator in the Cinema:JNC viewer. Currently, this has similar functionality to Cinema:View, allowing the user to use sliders to view a Cinema database through time and spatial angles. This viewer could be included in a Jupyter-based workflow, leveraging Python’s data analysis capabilities for post hoc analysis. Including multiple data types in a Cinema database, as described in Sect. 2.5, allows

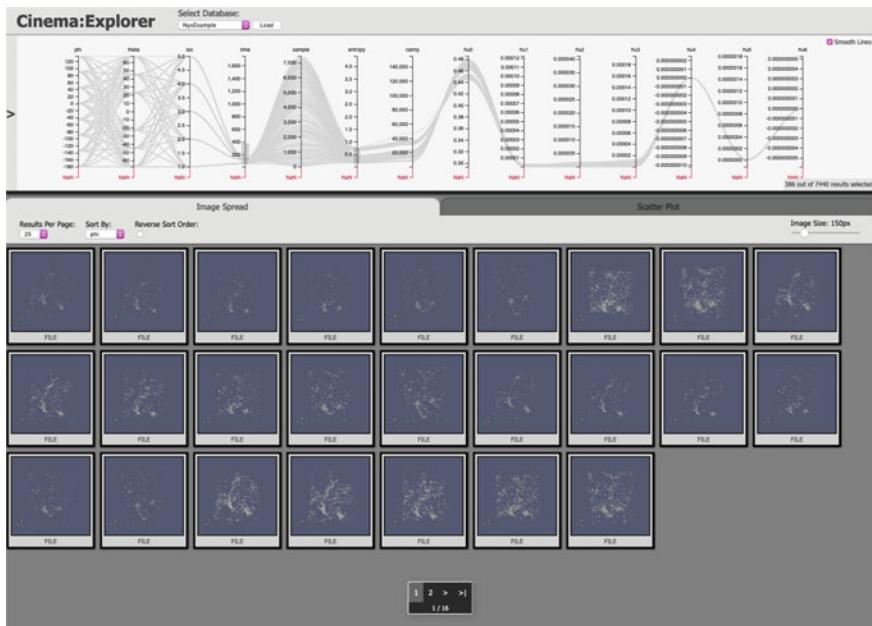


Fig. 7 In this image, Cinema:Explorer is used to query a large image database, using the parallel coordinates interactive view. For this example, the Nyx cosmology Cinema database has had computer vision algorithms applied to generate image-based statistics. Cinema:Explorer displays the database parameters and statistical quantities in the parallel coordinates view. Standard parallel coordinate techniques can be used to hide/show axes or to select ranges on the axes. Using the axes, the user has selected images early in time and with low entropy. The results of the query can be seen in the image spread view

workflows that, for example, use Python/VTK [25] based pipelines within a Jupyter notebook-based analysis framework.

2.4.5 Cinema Components

In addition to the standard viewers, Cinema also has a library of components that can be combined to create a viewer specific to the needs of the scientist.

These components include a parallel coordinates plot, image spread, scatter plot, query generator, and glyph-style plots. An example use case of an analysis-specific viewer built with individual components (beyond those used in Cinema:Explorer) can be found in [29]. In addition to these components, users can develop and combine new components as needed. Cinema:Bandit [21] is one such instance of a viewer built for a specific scientific application.

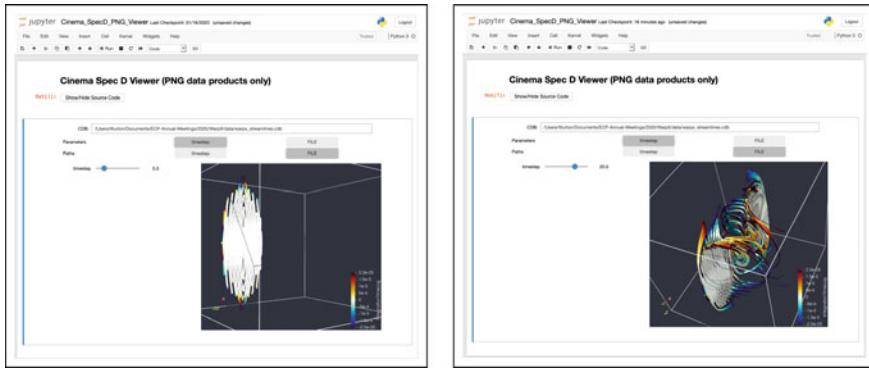


Fig. 8 Two views showing field streamlines in a WarpX plasma accelerator simulation. The prototype Jupyter notebook based viewer, Cinema:JNC, is used to explore the WarpX Cinema database. An early time step is on the left, evolving to a later time step on the right. Cinema database courtesy of R. Bujack, Los Alamos National Laboratory

2.5 Data Types Beyond Images

Cinema works on any data type and supports multiple data extracts and mixed data types for each parameter set (row in the database). For example, an in situ analysis could identify a specific feature of interest. The Cinema database export might save both a visualization of that feature and a small VTK-based mesh containing that

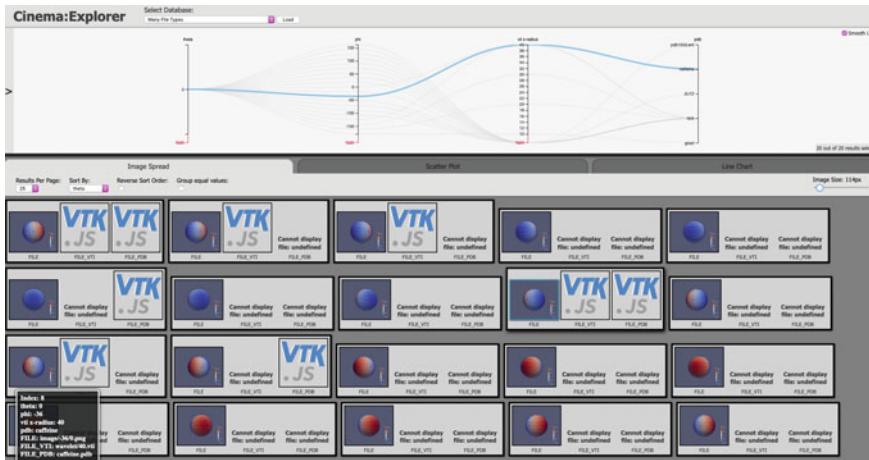


Fig. 9 A simple sphere Cinema database demonstrates the multiple data types in Cinema:Explorer. For each $[\phi-\theta]$ set of parameters, there is an image. Additionally, some of the $[\phi-\theta]$ parameter sets have other data types such as a vtk file or pbd file. Clicking on the thumbnail for each extract (vtk or pbd file) will bring up the data in a viewer for that specific data type, see Fig. 10



Fig. 10 Different file types: ParaView pbd file on the left and a ParaView vti file on the right. These are displayed in a modal view when selected by clicking on the file name in the Cinema:Explorer image spread

feature. This is an effective way to downsample the data by not saving the full simulation mesh. That VTK-based file is then included in the Cinema database, associated with the same set of parameters (e.g., view angles and time) that identify the corresponding image.

An example of multiple data types can be seen using the Cinema:Explorer viewer. In Fig. 9, a Cinema database of a simple sphere contains the visualization of the sphere for different ϕ and θ values. Some of the rows also have an associated VTK or PBD file. Clicking on file types brings up the vti or pbd data for interactive viewing in a modal view, see Fig. 10. Rows where the vti or pbd datafile is not available have an informational message displayed.

3 Analysis Algorithms

As discussed in this chapter's introduction, Cinema is a hybrid approach that produces heterogeneous data extracts *in situ*. These extracts can become the input for post hoc analysis workflows. The light-weight nature of the Cinema database approach enables flexible real-time exploration. This exploration may be conducted through the Cinema:View viewer, the parallel coordinates plot in Cinema:Explorer and in the case of image databases with associated saved parameters, through computer vision and statistical algorithms applied to the images and saved data parameters. Large image databases with several columns of associated numerical information can easily be produced through the methods discussed in Sect. 2.3. Standard tools to help users explore and understand these data products are an essential facet of the Cinema ecosystem. Two examples of algorithms for Cinema image database exploration—a computer vision framework and a set of statistical methods—illustrate how users may employ such techniques for in-depth analysis.

As shown in Fig. 2, the image-based *Analysis Algorithms* are part of an *iterative workflow*. The results of analysis algorithms, either computer vision or the statistical methods, are collated back into the Cinema database as additional images or columns

of numerical data. Therefore, a user who needs to compute a series of steps on their data can accomplish this goal through a sequential set of commands.

It is important to note that when analyzing images, the format in which the image is saved becomes highly pivotal. For RGB color-mapped images, the user must be aware of the potential effects of the colormap on their data and subsequent post-processing analyses. Alternatively, the user may opt to save their data as *cinema composable images*, where the simulation data is directly projected into the 32-bit pixels of the image. This allows for a more direct application of the Analysis Algorithms to the underlying data.

3.1 Computer Vision Framework

There exists a wide body of image-based computer vision techniques that can be exploited in a Cinema database workflow. However, a computer vision framework for data analysis and visualization must ensure that each component of the framework helps the user identify and examine features that directly correspond to attributes of the simulation. Therefore, the methods currently included in this framework help to identify common physical properties of interest in scientific data. For example, a gradient-based edge-detection algorithm is included to identify regions with sharp discontinuities. An examination of the Western Boundary of the Gulf Stream discussed in [27] is an example of an application of this edge-detection technique.

Another capability of the computer vision framework is contour detection, to locate closed regions at and above (or below) a given threshold, i.e., superlevel or sublevel sets. These extracted features may then be matched temporally or spatially based on attributes and metrics such as location, pixel intensities, area or derived quantities such as Hu moments. Given a particular matching metric, these matched features can be tracked temporally to identify events such as splits, merges, deaths or births. An example of this computer vision based workflow is discussed below and the interested reader can find further details in [6, 27].

3.1.1 Case Study: MPAS-Ocean Eddy Tracking

A Cinema-enabled computer vision workflow for eddy analysis is described in [6]. This is an example of a typical analysis workflow leveraging the Cinema ecosystem. This system allows the user to identify mesoscale ocean eddies, track their movement and visualize these results over time through count and tracking graphs. The input to the application is a Cinema database of MPAS-Ocean floating point images, to which the feature analysis algorithms are applied. The Cinema database images have the relevant physical variables, in this example, kinetic energy, embedded within the images allowing the user access to the simulation data at the resolution of the saved images.

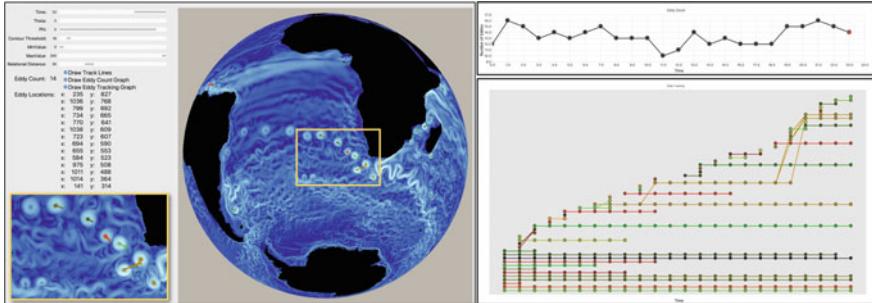


Fig. 11 Visualization of eddy feature tracking in the Agulhas Retroreflection Region of an MPAS-Ocean Simulation. The computer vision framework includes a control panel (upper left, see Fig. 12 for more detail) with sliders to select images and set algorithm parameters; visualization panels for overview and zoomed display; and output information such as the eddy count (upper right) and tracking chart (lower right). (Image is adapted from our previous work [6])

An interface such as the one shown, allows certain areas of domain expertise to be inherently captured. What is considered an “eddy” is intuitively understood by the scientist even though a rigorous mathematical definition of an eddy eludes ocean scientists. The eddy tracking application, shown in Fig. 11, allows the user to select contour detection as the computer vision technique and flexibly set thresholds to identify a set of eddies spatially. The application then tracks and creates a timeline of eddy progression. To track big eddies or small eddies, to include weaker eddies or only allow stronger eddies, and to determine over which region the eddy analysis will occur are all actions enabled by the interface. The user can choose to output a running count of the number of eddies found and a tracking graph that indicates the death/birth/merge events in eddy formation (upper and lower right of Fig. 11, respectively). This combination of low-cost Cinema database images and optimized computer vision algorithms enables an interface for scientists where exploration of the data is possible in real-time. The real-time Cinema-based approach can be compared to mesh-based analysis methods such as geometric eddy detection algorithms. Running those types of algorithms on large data sets requires far more time, limiting the exploratory capabilities for the user.

3.2 Statistical Methods

Statistical tools in the Cinema ecosystem are another data analysis approach within a Cinema workflow. Statistical tools are a ubiquitous choice for analysis of numerical data. The data may be derived from a Cinema image database, obtained during in situ processing, extracted from simulation data or added to the Cinema database from external sources. `cinema` [12] is a command line python-based tool that allows users to extract typical image properties such as mean, standard deviation, Shannon

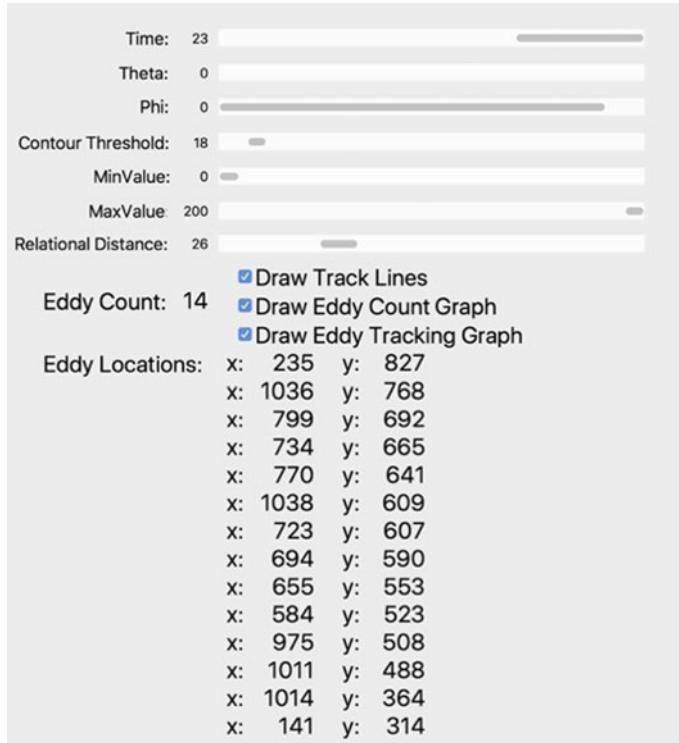


Fig. 12 Close-up of the computer vision framework control panel. At the top are sliders for the database parameters: time, theta, and phi. Underneath are algorithm parameters such as thresholds and ranges. User interface options allow the user to customize the views. Brief overview of the algorithm output is also included. (Image is adapted from our previous work [6])

entropy and joint entropy into numerical quantities added to the Cinema database. Maack et al. [18] leveraged the Cinema framework to apply statistical metrics to find features of interest.

Another approach enabled by Cinema is statistical change point detection (see for example, [23] for a discussion of the concept of change point detection). Within a physical system, *change* often denotes an interesting time step or event. Visually scanning thousands of images may be too time-intensive for a domain scientist to find events of interest. Change point detection can be applied to properties of simulation images to identify time steps or parametric values of interest. Both [5, 7] are examples of the application of change point detection within the body of Cinema literature. The change point detection algorithm available as part of the Cinema ecosystem, [13], allows the user to identify locations in a sequence of numbers where *change* has occurred [19], as defined through a linear regression model.

In Fig. 13, the Cinema statistical change point detection algorithm is applied to a simulation of the Deep Water Impact Ensemble Data Set [22], the same simulation

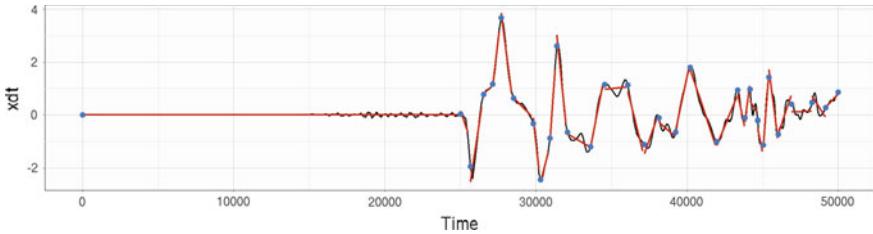


Fig. 13 Statistical change point detection for the ‘*xdt*’ parameter of the Deep Water Impact Ensemble Data Set [22]. Change detection parametric values are: $B = 4$, $\alpha = 0.8$, $\delta^2 = 1$. The change points are noted in blue while the red lines drawn show the best fit line to the data between change points

shown in Fig. 3. Instead of using images stored in a Cinema database, the parameters in the database can be used for the statistical change point detection analysis. In this analysis, a representative slice is taken in the z-plane through the center of the data and the data values on that slice are averaged. This process is repeated for each of the 476 saved time steps spanning about 50,000 steps in simulation time. Change detection is then applied to a parameter of interest from the simulation parameters. The selected parameter, *xdt*, is the x component of the velocity in each cell, in centimeters per second. Change point detection is applied to this parameter over time to identify points of interest.

During the first 25000 time units of the simulation, as the asteroid is descending towards earth, there is very little activity and therefore, no change points detected. Intense and frequent change points are seen as the asteroid strikes the ocean surface. The algorithm identifies points in time where moderate to large amounts of change occur. Even in the latter part of the simulation, regions of time with smaller amount of change are still grouped together. The algorithm enables the flexibility to adjust the change detection parameters so as to find smaller or larger amounts of change, as desired by the user for their particular analysis and data set.

4 Task-Based Workflow Examples

To streamline in situ Cinema database production, a task-based workflow has been developed that works in a distributed parallel environment. In this workflow, the user first specifies the parameters that will be used to generate the Cinema database and the range of values for each parameter. The user also can select if the parameter ranges will be divided regularly or randomly while creating specific parameter combinations. For example, the user can specify the ranges of the viewing angle parameters (e.g., $240 \leq \phi \leq 360$ and $50 \leq \theta \leq 100$) and how many visualization samples are needed from this range. Then the framework will sample the parameter ranges either regularly or randomly as specified by the user to produce combinations of (ϕ, θ) values. For

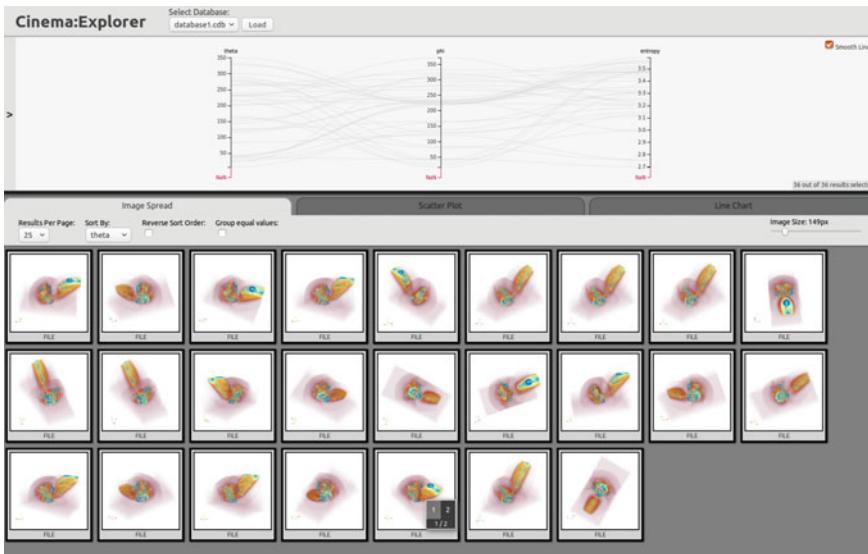


Fig. 14 Visualization of a Cinema database produced by our task-based workflow using the Asteroid impact data set [22]. The database produced visualization of Temperature field using volume visualization technique

each such (ϕ, θ) combination, a visualization extract will be added to the Cinema database. The user can also specify the type of visualization algorithm such as surface rendering or volume rendering that will be used to produce the visualization data extracts.

In this workflow, a *unit task* is characterized as the production of a visualization extract for a specific parameter combination (ϕ , θ , etc.). Since the visualization and rendering in parameter space is high-dimensional in nature, and can be quite large, a thorough exploration of such space while producing a Cinema database may need a very large number of such unit tasks. Therefore, the task-based workflow first generates the list of tasks that will be needed in order to produce the complete Cinema database as specified by the user. Those tasks are then distributed among the different compute nodes of a high performance cluster (HPC). Finally, the tasks are executed in parallel. At the end of the task-based workflow, a Cinema database is created with all the visualization extracts. Finally, the workflow installs a Cinema database viewer customized for the resultant Cinema database so that the results can be readily explored interactively. Figure 14 shows an example Cinema database generated using this task-based workflow and viewed using the Cinema:Explorer viewer, Sect. 2.4.

The current version of the task-based workflow is implemented in Python using the `mpi4py` library for distributed processing. The visualizations are produced using ParaView [1] in off-screen rendering mode. User control is through a JSON file specifying the range of parameters and types of visualizations needed. The task-

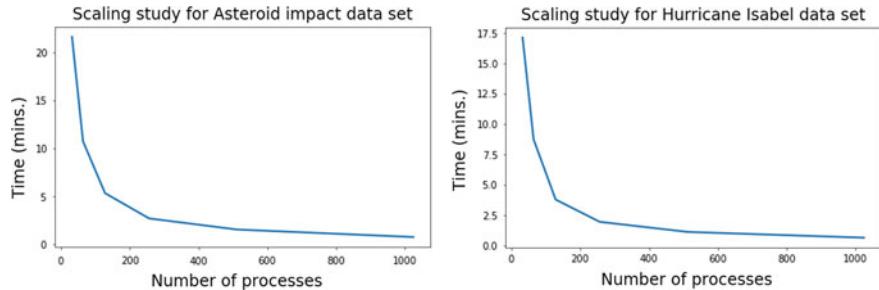


Fig. 15 Strong scaling study of Cinema’s task-based workflow for (left) the asteroid impact data set and (right) the Hurricane Isabel data set

based workflow will then generate the list of tasks and execute them in parallel on the HPC machine.

Performance testing of the task-based workflow was done on Snow, an Institutional Computing (IC) Commodity Technology System Phase I cluster (CTS-1) located at Los Alamos National Laboratory. Snow has two integrated Scalable Units (SU) and each unit forms a building-block to assemble the CTS-1 cluster. Each SU has 184 compute nodes plus other nodes for services, I/O, etc. Each node in the SU has 36 Processor cores: $2 \times (\text{E5 } 2695\text{v4 } 2.1\text{ GHz}, 18 \text{ cores}, 45\text{ MB cache})$, 128GB Memory, and Intel OmniPath OP HFI, Single-port, PCIe-gen3 x16 Network Interconnect.

In Fig. 15, the result of a strong scaling study of the task-based workflow is shown for two sets of data: the asteroid impact data [22] and Hurricane Isabel data [30]. The asteroid data used volume rendering for visualization and the Hurricane Isabel data used an isocontour visualization technique. Each test case consisted of 10000 tasks and the number of processing cores were varied from 32 to 1024. Since the tasks were distributed among different processing nodes and the tasks are independent, it can be observed that with increased number of processing cores, the computation time goes down and the workflow scales as expected for both the data sets. This study demonstrates the practicality of a task-based workflow for in situ generation of image-based Cinema databases.

5 Conclusion

Cinema provides a powerful ecosystem for extracting, storing and interacting with scientific and experimental data at any scale—from small tables of data to extreme-scale simulations running on the largest supercomputers. As an open data definition, a Cinema database is a data set that any application can read and write, making it simple to start using Cinema, adding capabilities as needed. In contrast to monolithic applications, Cinema provides an opportunity to participate by re-using small

components from open source libraries. Cinema-enabled data analysis and visualization workflows have been used across a wide range of data. Already included in many standard applications and frameworks for large scale science, Cinema is a good option for complex scientific data analysis.

Acknowledgements Many collaborators have contributed to the development of Cinema. We thank everyone who contributed and would like to acknowledge our LANL Data Science at Scale team members: A. Biswas, C. Biwer, R. Bujack, L-T. Lo, D. Orban, J. Patchett, C. Tauxe, J. Q. Wofford; our science collaborators: G. Gisler, M. Petersen, J. Schoonover, Z. Lukic, J-L. Vay, J. Musser, A. Almgren; our industry partners at Kitware and Intelligent Light and other collaborators: J. Lukasczyk, C. Harrison, M. Larsen, J. Woodring, G. Aldrich, G. Streletz. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. The Hurricane Isabel data was provided by Wei Wang, Cindy Bruyere, Bill Kuo, and others at NCAR with Tim Scheitlin at NCAR converting the data into the Brick-of-Float format. This research was released under LA-UR-20-20649.

References

1. Ahrens, J., Geveci, B., Law, C.: Paraview: An end-user tool for large data visualization. *The Visualization Handbook*, vol. 717 (2005)
2. Ahrens, J., Jourdain, S., O’Leary, P., Patchett, J., Rogers, D.H., Petersen, M.: An image-based approach to extreme scale in situ visualization and analysis. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 424–434. IEEE Press (2014)
3. Almgren, A.S., Bell, J.B., Lijewski, M.J., Lukic, Z., Andel, E.V.: Nyx: a massively parallel AMR code for computational cosmology. *Astrophys. J.* **765**(1), 39 (2013). <https://doi.org/10.1088/0004-637x/765/1/39>
4. Ayachit, U., Bauer, A., Geveci, B., O’Leary, P., Moreland, K., Fabian, N., Mauldin, J.: Paraview catalyst: enabling in situ data analysis and visualization. In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp. 25–29. ACM (2015)
5. Banesh, D., Petersen, M., Wendelberger, J., Ahrens, J., Hamann, B.: Comparison of piecewise linear change point detection with traditional analytical methods for ocean and climate data. *Environ. Earth Sci.* **78**(21), 623 (2019)
6. Banesh, D., Schoonover, J.A., Ahrens, J.P., Hamann, B.: Extracting, visualizing and tracking mesoscale ocean eddies in two-dimensional image sequences using contours and moments. In: Rink, K., Middel, A., Zeckzer, D., Bujack, R. (eds.) *Workshop on Visualisation in Environmental Sciences (EnvirVis)*. The Eurographics Association (2017). <https://doi.org/10.2312/enviris.20171103>
7. Banesh, D., Wendelberger, J., Petersen, M., Ahrens, J., Hamann, B.: Change point detection for ocean eddy analysis. In: *Workshop on Visualisation in Environmental Sciences (EnvirVis)*. The Eurographics Association (2018). <https://doi.org/10.2312/enviris.20181134>

8. Bujack, R., Rogers, D., Ahrens, J.: Reducing occlusion in cinema databases through feature-centric visualizations. In: Leipzig Symposium on Visualization in Applications (LEVIA) (2018). <https://datascience.dscc.org/wp-content/uploads/2019/01/ReducingOcclusioninCinemaDatabasesthroughFeature-CentricVisualizations.pdf>
9. Childs, H., et al.: VisIt: an end-user tool for visualizing and analyzing very large data. In: High Performance Visualization—Enabling Extreme-Scale Scientific Insight, pp. 357–372. CRC Press/Francis–Taylor Group (2012)
10. Cinema: Cinema specification (2018). https://github.com/cinemascience/cinema/blob/master/specs/dietrich/01/cinema_specD_v012.pdf. Accessed Jan 2020
11. Cinema-Developers: Cinema publications (2018). (List of Cinema publications, available on CinemaScience website.) <https://cinemascience.github.io/publications.html>
12. Cinema-Developers: Cinema science (2018). <http://cinemascience.org/>. Accessed 11 Dec 2019
13. Cinema-Developers: Cinema change point detection (2019). https://github.com/cinemascience/cinema_change_detection. Accessed 22 Jan 2020
14. ECP: Exascale Computing Project (2017). <https://www.exascaleproject.org/>. Accessed Jan 2020
15. Fabian, N., Moreland, K., Thompson, D., Bauer, A.C., Marion, P., Gevecik, B., Rasquin, M., Jansen, K.E.: The ParaView coprocessing library: a scalable, general purpose in situ visualization library. In: 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV), pp. 89–96. IEEE (2011)
16. Grossset, P., Biwer, C.M., Pulido, J., Mohan, A.T., Biswas, A., Patchett, J., Turton, T.L., Rogers, D.H., Livescu, D., Ahrens, J.: Foresight: analysis that matters for data reduction (2020). To appear in SC '20: International Conference on High Performance Computing, Data, and Analytics., November 2020
17. Larsen, M., Ahrens, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., Harrison, C.: 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, ISAV'17, pp. 42–46. ACM, New York, NY, USA (2017). <https://doi.org/10.1145/3144769.3144778>
18. Maack, R., Rogers, D., Gillmann, C.: Exploring cinema databases using multi-dimensional image measures. In: Leipzig Symposium on Visualization in Applications (LEVIA) (2019)
19. Myers, K., Lawrence, E., Fugate, M., Bowen, C.M., Ticknor, L., Woodring, J., Wendelberger, J., Ahrens, J.: Partitioning a large simulation as it runs. *Technometrics* **58**(3), 329–340 (2016)
20. O’Leary, P., Ahrens, J., Jourdain, S., Wittenburg, S., Rogers, D.H., Petersen, M.: Cinema image-based in situ analysis and visualization of mpas-ocean simulations. *Parallel Comput.* **55**(C), 43–48 (2016). <https://doi.org/10.1016/j.parco.2015.10.005>
21. Orban, D., Banesh, D., Banesh, C., Biwer, C., Biswas, A., Saavedra, R., Sweeney, C., Sandberg, R., Bolme, C.A., Ahrens, J., Rogers, D.: Cinema:bandit: a visualization application for beamline science demonstrated on xfel shock physics experiments. *J. Synchrotron Rad.* **27**(1) (2020). <https://doi.org/10.1107/S1600577519014322>
22. Patchett, J.M., Gisler, G.R.: Deep water impact ensemble data set. Technical Report, Los Alamos National Laboratory (2017). LA-UR-17-21595
23. Ray, B.K., Tsay, R.S.: Bayesian methods for change-point detection in long-range dependent processes. *J. Time Series Anal.* **23**(6), 687–705 (2002)
24. Ringler, T., Petersen, M., Higdon, R.L., Jacobsen, D., Jones, P.W., Maltrud, M.: A multi-resolution approach to global ocean modeling. *Ocean Model.* **69**, 211–232 (2013). <https://doi.org/10.1016/j.ocemod.2013.04.010>
25. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit, 4th edn. Kitware, Clifton, New York (2006)
26. Syamlal, M., Musser, J., Almgren, A., Bell, J., Hrenya, C., Hauser, T., Liu, P.: MFIX-Exa: a CFD-DEM code for exascale computer architectures. Abstract and Presentation in Computational Modeling and Validation for Fluidization Processes at AIChE Annual Meeting (2018)
27. Turton, T.L., Banesh, D., Overmyer, T., Sims, B.H., Rogers, D.H.: Enabling domain expertise in scientific visualization with cinemascience. *IEEE Comput. Graph. Appl.* **40**(1), 90–98 (2020). <https://doi.org/10.1109/MCG.2019.2954171>

28. Vay, J.L., et al.: Warp-X: a new exascale computing platform for beam-plasma simulations (2018). [arXiv:1801.02568](https://arxiv.org/abs/1801.02568) [physics.acc-ph]
29. Vogel, S.C., Biwer, C.M., Rogers, D.H., Ahrens, J.P., Hackenberg, R.E., Onken, D., Zhang, J.: Interactive visualization of multi-dataset rietveld analyses using cinema:debye-scherrer. *J. Appl. Crystallogr.* **51** (2018). <https://doi.org/10.1107/S1600576718003989>
30. Wang, W., Bruyere, C., Kuo, B., Scheitlin, T.: IEEE visualization 2004 contest data set (2004). <http://sciviscontest.ieeevis.org/2004/data.html>. NCAR
31. Woodring, J., Ahrens, J.P., Patchett, J., Tauxe, C., Rogers, D.H.: High-dimensional scientific data exploration via Cinema. In: 2017 IEEE Workshop on Data Systems for Interactive Analysis (DSIA), pp. 1–5 (2017). <https://doi.org/10.1109/DSIA.2017.8339086>

New Research Results and Looking Forward

Deep Learning-Based Upscaling for In Situ Volume Visualization



Sebastian Weiss, Jun Han, Chaoli Wang, and Rüdiger Westermann

Abstract Complementary to the classical use of feature-based condensation and temporal subsampling for in situ visualization, learning-based data upscaling has recently emerged as an interesting approach that can supplement existing in situ volume visualization techniques. By upscaling we mean the spatial or temporal reconstruction of a signal from a reduced representation that requires less memory to store and sometimes even less time to generate. The concrete tasks where upscaling has been shown to work effectively are geometry upscaling, to infer high-resolution geometry images from given low-resolution images of sampled features; upscaling in the data domain, to infer the original spatial resolution of a 3D dataset from a down-scaled version; and upscaling of temporally sparse volume sequences, to generate refined temporal features. In this book chapter, we aim at providing a summary of existing learning-based upscaling approaches and a discussion of possible use cases for in situ volume visualization. We discuss the basic foundation of learning-based upscaling, and review existing works in image and video super-resolution from other fields. We then show the specific adaptations and extensions that have been proposed in visualization to realize upscaling tasks beyond color images, discuss how these approaches can be employed for in situ visualization, and provide an outlook on future developments in the field.

S. Weiss · R. Westermann (✉)
Technical University of Munich, Garching bei München, Germany
e-mail: westermann@tum.de

S. Weiss
e-mail: sebastian13.weiss@tum.de

J. Han · C. Wang
University of Notre Dame, Notre Dame, IN, USA
e-mail: jhan5@nd.edu

C. Wang
e-mail: chaoli.wang@nd.edu

1 Introduction

For in situ volume visualization, two commonly employed approaches are feature-based data reduction and spatial or temporal subsampling. In the former approach, each dataset is condensed in situ to a few important features, and these features are used to analyze the data. Since extracted features usually require far less memory than the original data, memory bandwidth and capacity limitations can be overcome. On the other hand, features not selected are lost, and some feature extraction techniques require global data access operations not well supported by the data distribution scheme on the parallel computing architecture. In the latter approach, volumetric data are downscaled spatially or only every n th timestep is stored, thus reducing the amount of memory needed. Then, spatial or temporal features can get lost, since classical interpolation schemes cannot reconstruct these features from the reduced data in general.

Complementary to these approaches, learning-based data upscaling has recently emerged as an interesting approach that can supplement existing in situ volume visualization techniques. By *upscaling* (which is also referred to as *super-resolution*), we refer here to the spatial or temporal reconstruction of a physical field from a reduced representation. The concrete tasks where learning-based upscaling has been demonstrated for scientific data are

- 1 *Image upscaling*: Upscaling in the visualization image domain by inferring high-resolution images from given low-resolution images of sampled 3D features.
- 2 *Spatial upscaling*: Upscaling in the spatial domain by inferring a higher resolution of a 3D dataset from a downsampled version for reconstructing spatial features.
- 3 *Temporal upscaling*: Upscaling in the temporal domain by inferring temporally dense volume sequences from sparse sequences for refining temporal features.

Option (1) requires focusing on specific features that are visualized at low image resolution. Yet, resulting images require less data access operations to generate, can be generated at higher speed, and decrease the memory required to store and transmit them, e.g., for use over low bandwidth channels in remote visualization environments. With options (2) and (3), storing a dataset becomes faster, and more datasets can be stored in a given a certain disk capacity. Depending on the size of the downsampled dataset, it can be even feasible to stream an entire dataset to a remote client where the visualization occurs. This can be an interesting option to monitor the running simulation.

The challenge in upscaling is to infer the structure of a coarsely sampled dataset from a low-resolution spatial or temporal sampling, beyond what can be predicted from the given samples by classical upscaling filters like bilinear or bicubic interpolation. In general, this seems impossible without any further assumptions about the structures that are contained in the dataset. Recent works in learning with artificial neural networks (ANNs), however, have demonstrated that such networks have the capabilities to learn such assumptions.

In abstract terms, upscaling seeks a mapping function from inputs to outputs, called the generator. In the in situ scenario, the generator learns to map a multi-

dimensional field, e.g., a 2D color or geometry image, or a 2D or 3D and possibly time-varying scalar or vector field, to a higher spatial or temporal resolution, optionally including additional parameters, i.e., channels, that are inferred from the given input samples.

With ANNs, the generator internally builds a so-called *latent space representation* that encodes the nonlinear mapping function. The generator tries to learn an identity mapping, which gives for every low-resolution input the corresponding high-resolution output. Since the dimension of the latent space is not sufficient to achieve such a mapping for every input, the network learns to encode relevant features that have a significant impact on the inference quality. Thus, the generator learns assumptions about the occurrence of structures by using corresponding pairs of low- and high-resolution fields in the training process. Learned assumptions can then be transferred to a new low-resolution input, to generate a high-resolution variant that adheres to the structures seen at training time.

Learning-based image and video super-resolution have achieved remarkable results, by training networks using corresponding pairs of low- and high-resolution color images [9, 26]. Learned assumptions can then be transferred to a new low-resolution input, to generate a high-resolution variant in which structures that have been seen at training time are well recovered. Similar approaches have been used in the context of numerical fluid simulation, in particular, to add turbulent sub-structures to low-resolution input simulations [6, 57].

In this book chapter, we aim to bring the readers' attention to the possibilities of learning-based upscaling in the context of in situ volume visualization. Even though learning-based upscaling has not yet been integrated into existing in situ visualization systems, especially in this context, we see it as an interesting technique that can effectively complement existing approaches. In particular, recent works on spatial and temporal upscaling of physical fields indicate that networks do not learn a specific field, but rather that networks can generalize and learn properties of structures that occur in such fields.

On the other hand, as recent works have shown, a network cannot infer the missing data samples accurately in all but very simple scenarios. In particular, when structures appear which have never been seen by the network in the training process, or the sampling frequency is so low that structures are missed entirely, an accurate reconstruction cannot be expected in general. While some networks tend to hallucinate new structures in this case, other network variants prefer a smooth continuation similar to classical smoothing filters.

One way to address this problem is to build specialized networks for certain types of simulations to learn the specific features (and their spatial and temporal relationships) that can occur. To achieve this, further research is required to explore the limits of predictability using network-based inference, including in particular a thorough investigation of their specialization capabilities in different application areas. In this context, it will be important to shed light on the use of additional information sources, besides low-resolution versions of the data, that can be generated efficiently in a running simulation and can help to improve the prediction. For instance, to investigate whether certain feature indicators can be derived and stored together with the low-

resolution version, so that the inference step can learn to combine both sources in a meaningful way and improve the inference results.

Another limitation of current network architectures for upscaling tasks is performance, which is essentially bound by the many data access operations a network needs to perform to reconstruct a full resolution dataset from the low-resolution version. We are confident that with faster deep-learning hardware and performance-optimized network architectures the performance and scalability of upscaling networks will increase significantly over the next years. Furthermore, in recent work it has been shown that ANNs can even learn the importance of samples for data reconstruction [54], by training an adaptive sampling network and a reconstruction network end-to-end. This work, in particular, gives evidence that networks can be trained to convert a dataset into a sparse, yet feature-preserving representation from which another network has learned to generate a visualization in turn, i.e., without having to reconstruct the full resolution field. Such an approach can significantly reduce the number of data access operations and, thus, increase the visualization performance accordingly.

2 Background and Related Work

2.1 Artificial Neural Networks

In the following, we will briefly discuss some of the basic machine learning mechanisms underlying neural network-based upscaling, together with a short description of the specific architectures used in upscaling for visualization. For a thorough introduction to and an exhaustive summary of the developments in neural network-based learning, we refer readers to the overviews by Schmidhuber [45] and Goodfellow et al. [13].

In recent years, ANNs have gained tremendous attention due to their superior performance to alternative methods in many pattern recognition and machine learning tasks. The power of these networks comes from their ability to form representations of categories by “learning” from large sets of samples in which these categories are present. The learned representations are then used to recognize, classify and infer about properties of objects and events in unknown samples. An ANN aims to solve problems in a similar way as a human brain. It consists of interconnected nodes akin to the vast network of neurons in the brain. Artificial neurons are aggregated into layers, and different layers perform different kinds of transformations on their inputs. A *deep neural network* (DNN) is an ANN with multiple layers between the input and output layers [32]. These extra hidden layers result in a *deep* network, enabling the composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing *shallow* network [2]. As an essential branch of machine learning, deep learning has been successfully applied to many applications such as computer vision, speech recognition, natural language

processing, and computer graphics, achieving results comparable or even superior to human experts. In recent years, fully connected network layers were replaced by deep hierarchies using only local convolutional update operations, and massively parallel graphics processing units (GPU) equipped with high-performance memory interfaces made learning with large datasets and many layers practicable. *Convolutional neural networks* (CNNs) have proven very successful for *supervised* image recognition and classification. The hidden layers of a CNN include several stages of local convolutional and pooling layers, sometimes followed by one or several fully-connected layers. A neuron $z_j^{(i+1)}$ at *convolutional layer* $i + 1$ detects local combinations of features by aggregating information from the previous layer i via convolution operations and non-linear state updates:

$$a_j = W_j * z^i + b_j, \quad (1)$$

$$z_j^{(i+1)} = f(a_j), \quad (2)$$

where W_j is a linear convolution operator with trainable weights, and b_j is a bias used to allow for absolute shifts in the output values. A non-linear function f is used to compute a neuron's final state, its so-called *activation*, to enable the network to perform a non-linear mapping of its inputs without affecting the receptive fields of the convolutional layers. Typical activation functions are $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$ or $\text{ReLU}(x) = \max(0, x)$. A *pooling layer* aggregates features into smaller and smaller feature maps via down sampling, i.e., it merges semantically similar features into one by computing a summary (e.g., maximum or average). The *fully-connected layer* learns global features from local ones by connecting neurons to all activations in the previous layer. This is used, for instance, in classification tasks to generate the final classification mask from the activations of all neurons in the final convolutional layer. Today, deep learning [30, 48] refers to the use of high-throughput architectures for learning hierarchical representations of categories, which are formed via non-linear models for transforming the representation at one level into a more abstract one.

For network learning, a *loss function* is used to measure the difference between the predicted network output and the desired result, e.g., a class label in classification tasks or a high-resolution ground truth image in image-based upscaling. Minimizing the loss function is usually achieved with an optimization method such as *stochastic gradient descent* (SGD). SGD first calculates the gradient of the loss function with respect to the network's weights and biases, and then updates the parameters according to a given *learning rate*. The network's weights and biases are the only parameters that can be modified to make the loss as low as possible. Because the loss is calculated through the composition of the neurons' activation functions, it is a continuous and piecewise differentiable function of the parameters. This gives rise to minimization via an iterative process of gradient descent. As such, the learning process reduces to calculating the gradient of a network function with respect to its weights. This is what is performed in the training step.

A simple yet often surprisingly effective class of loss functions considers regular vector norms, i.e., L_1 or L_2 , over the data domain. *Perceptual losses* [10, 12, 25]

have been widely adopted in image and video upscaling to guide networks towards additional image details instead of smoothed mean values. The idea is that two images are similar if they have similar activations in the latent space of a trained network. Let ϕ be the function that extracts the layer activations when feeding an input image into the network, and let O^{EST} and O^{GT} , respectively, be the inferred output and the ground-truth image. The perceptual loss function aims at minimizing the distance

$$\mathcal{L}_P = \sum_s \|\phi(O_s^{GT}) - \phi(O_s^{EST})\|_2^2. \quad (3)$$

Autoencoders [29] can learn latent features from data in an *unsupervised* manner, e.g., to use these features for dimensionality reduction. An autoencoder consists of two networks: an *encoder* and a *decoder*. The encoder encodes an input data sample to a compressed representation in the latent space. The decoder decodes the latent representation back to reconstruct the data sample as close as possible. *Generative adversarial networks* (GAN) [14] are explicitly designed to optimize for generative tasks. A GAN consists of two networks: a *generator* and a *discriminator*, which contest with each other in a zero-sum game. The generator maps from a latent space to a particular data distribution of interest. The discriminator discriminates between instances from the true data distribution and candidates produced by the generator. In adversarial training, this discriminator is then used in the loss function of the generator network. A popular loss function in adversarial training is the *binary cross-entropy loss*. Let z be the input over all timesteps and $G(z)$ the generated results, and let $D(x)$ be the discriminator that takes the generated results as input and produces a single scalar score. Then the discriminator is trained to distinguish fake from real structures by minimizing

$$\mathcal{L}_{GAN,D} = -\log(D(x)) - \log(1 - D(G(z))). \quad (4)$$

The generator is trained to minimize

$$\mathcal{L}_{GAN,G} = -\log(D(G(z))). \quad (5)$$

Recurrent neural networks (RNN) [15, 42] can capture the dynamics of a sequence or a time series and are widely used in speech recognition, text synthesis, and hand-writing recognition. As the most popular deep RNN architecture, the *long short-term memory* (LSTM) [22] addresses the vanishing gradient problem in traditional RNNs by adding forget gates in the units. To strengthen the temporal coherence of time series predictions, one can use loss functions that explicitly penalize differences between predictions and different timesteps. To cope with motion in images, let $W_{i,j}$ be a warping operator that aligns the output at time i with the output at time j with respect to some motion field. Then, the temporal loss can be defined as

$$\mathcal{L}_{temp} = \sum_t \|O^{EST_t} - W_{t-1,t}(O^{EST_{t-1}})\|_2^2. \quad (6)$$

2.2 Related Work in Upscaling

Upscaling Image and Video Data. In recent years, deep learning approaches have been used successfully for single-image super-resolution tasks [8, 31, 47, 49, 51], i.e., the upscaling of images and videos from a lower to some higher resolution. Many previous works let the networks learn to optimize for losses between the inferred and ground-truth images based on direct vector norms [27, 28]. GANs were introduced to prevent the undesirable smoothing of direct loss formulations [33, 43], and instead use a second discriminator network that discriminates real from generated samples and guides the generator. Convolutional architectures [8] with residual blocks [21] are popular generator architectures that offer training stability as well as high-quality inference. Losses based on the feature-space differences of image classification networks, e.g., a pre-trained VGG network [25] or the discriminator in a GAN setting, have shown to mimic well the human’s capability to assess the perceptual similarity between two images. In the context of synthetic images, existing work focuses on enhancing path-traced images. A common application is image denoising, e.g., for Monte-Carlo raytracing [4, 58].

In contrast to image super-resolution, video super-resolution tasks introduce the time dimension, and as such, require temporal coherence and consistent image content. While many methods use multiple low-resolution frames [24, 34, 50], the FRVSR-Net [44] reuses the previously generated high-resolution image to achieve better temporal coherence. By using a spatiotemporal discriminator, the TecoGAN [7] network produced results with spatial detail without sacrificing temporal coherence. Multiframe approaches benefit from aligning the frames via warping, which requires an estimation of the image-space motions. As this is usually not readily available for natural videos, optical flow estimation networks are a popular choice [3]. EDVR [52] uses alignment modules on a U-Net [41] to align features of different frames. Recurrent architectures employ feedback loops to address sequence, time, and video prediction tasks [5]. Chaitanya et al. [4] used recurrent connections to propagate a latent state over time inside the network, while other methods use the previously generated high-resolution output as input [44].

Upscaling Scientific Data. Thanks to the tremendous advances of deep learning solutions, visualization researchers have begun to explore the capabilities of DNNs for upscaling and reconstruction of 2D/3D steady and time-varying scientific data, including both scalar and vector fields. Closely related to scientific visualization, Zhou et al. [60] presented a CNN-based solution that upscales a volumetric dataset using three hidden layers designed for feature extraction, non-linear mapping, and reconstruction, respectively. Han et al. [17] took a two-stage approach for vector field reconstruction via deep learning. The first stage initializes a low-resolution vector field based on the input streamline set. The second stage refines the low-resolution vector field to a high-resolution one via CNN. The use of neural network-based inference of data samples in the context of in situ visualization was demonstrated by Han and Wang [18], by letting networks learn to infer missing timesteps between 3D simulation results. Guo et al. [16] designed a deep learning framework that produces

coherent spatial super-resolution of 3D vector field data. With a recommended scaling factor of 4 or 8, they can downsample vector field data by 64 or 512 times at simulation time and upsample these reduced data back to their original resolution with good quality.

Beyond scientific visualization, several works aim to upscale physical fields resulting from volumetric flow simulations. For example, Xie et al. [57] presented tempoGAN to synthesize spatial super-resolution volume sequences. Temporal coherence is ensured by wrapping velocity and vorticity fields into the synthesized volumes. Xiao et al. [56] proposed a CNN-based flow correction method for a fast preview of the smoke animation results based on low-resolution simulations, which was achieved through the use of a grid-layer feature vector along with a special loss function. Werhahn et al. [55] designed a multi-pass method to upsample 3D spatiotemporal functions with GANs. They decomposed generative problems on the Cartesian field functions into multiple smaller subproblems for efficient learning. Bai et al. [1] proposed a dynamic upsampling approach to generate high-resolution turbulent smoke flows using a dictionary-based neural network.

3 Upscaling Scenarios—Image-Based Upscaling

Rendering an accurate image of a volumetric field typically requires a large number of data samples, and reducing this number lies at the core of research in volume rendering. In the following, we shed light on the use of CNNs for learning the upscaling of a low-resolution rendering of an isosurface to a higher resolution [53], with reconstruction of spatial detail and shading. What makes this approach interesting for in situ visualization is the reduced number of data samples that need to be accessed to obtain the final high-resolution image. Since the high-resolution image is inferred from a low-resolution image with only 1/16 (about 6%) of the pixels in the high-resolution version, the number of data access operations that are required to generate the high-resolution image decreases correspondingly. Due to this, in situ environments where a dataset is so large that volume rendering must be performed while the data is being generated, increasing rendering performance can be expected. In addition, the low-resolution image can be streamed in turn to a remote visualization client, reducing bandwidth requirements by 16:1 and requiring no additional compression stage on the machine where the data is being generated.

The input and output of the image-based upscaling (IU) network in [53] are 2D images of an isosurface in a 3D scalar field. In contrast to classical image- and video-upscaling approaches, however, there are differences in what these images represent and what the network learns to infer. First, the upscaling task does not work on color images. Instead, it uses a geometry image of the isosurface, including depth and gradient information. As illustrated in Fig. 1, this leads to improved learning of geometric surface properties and avoids color bleeding effects when different color maps are used in the training process. Second, instead of learning a mapping from a low-pass filtered version of a given high-resolution image to that image, the network

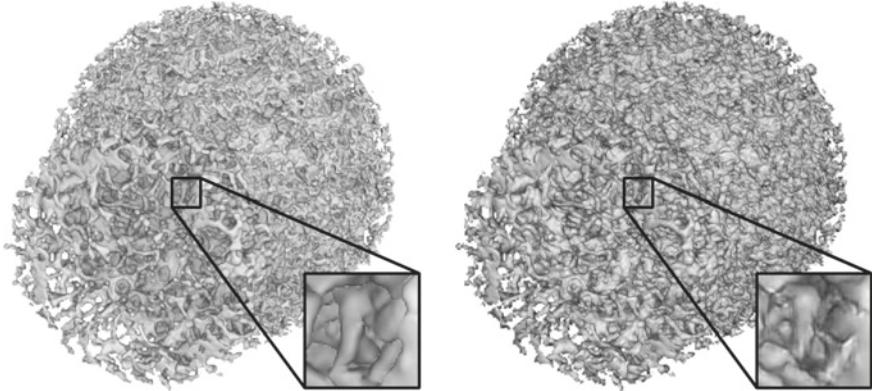


Fig. 1 Left: IU using depth and normal maps with screen-space shading. Right: upscaling of color images introduces color bleeding from color mappings seen during training. This image is adapted from our previous work [53]

is trained to learn a mapping from a low-resolution sampling of the isosurface in the high-resolution dataset to a version that is sampled at high image resolution. Third, the network learns to generate additional attribute channels from the given inputs. In particular, it infers global illumination values from the given geometry images by learning the relation between low-resolution geometry and global illumination in the high-resolution data.

Processing Pipeline. In every frame t , the network receives the low-resolution input I_t^{LR} , which comprises multiple 2D fields, the so-called channels. The binary mask $M_t^{LR} \in \{-1, +1\}^{H \times W}$ indicates for every pixel whether or not the isosurface is hit. The normal map $N_t^{LR} \in [-1, +1]^{3 \times H \times W}$ stores the screen-space normal vectors at the rendered surface points. The depth map $D_t^{LR} \in [0, 1]^{H \times W}$ stores the distance of each point to the viewing plane.

From these input fields, the network infers the high-resolution image of the isosurface O_t^{EST} , including additional output channels that were not given in the low-resolution input, such as high-resolution ambient occlusion (AO) values. Therefore, in the training process, the network is fed with ground-truth AO maps (generated via volumetric raycasting) and learns to predict them according to the low-resolution geometry. Screen-space Phong shading with AO is finally applied as a post-processing step. Optionally, a map of 2D displacement vectors $F_t^{LR} \in [-1, +1]^{2 \times H \times W}$ —indicating the screen-space flow from the previous view to the current view—is computed internally to let the network smooth the differences between subsequent predictions.

Architecture Details. The network architecture builds upon a fully convolutional frame-recurrent neural network (FRVSR-Net) consisting of a series of residual blocks [44]. In a residual network, some layers feed their output not only into the next layer but also directly into the layers many stages away. By using these so-called skip connections, the layers can learn the residual between the true output and the

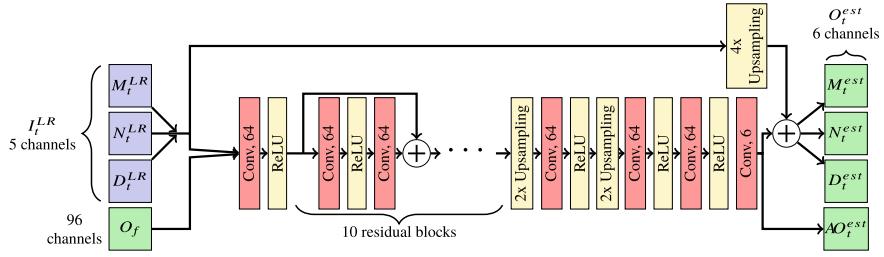


Fig. 2 The IU network architecture. \oplus indicates component-wise addition of the residual. All convolutions use 3×3 kernels with stride 1. Bilinear interpolation is used for the upsampling layers. This image is reprinted from our previous work [53]

prediction. Furthermore, larger gradients can be propagated to initial layers, thus avoiding the problem of vanishing gradients in deep networks. The FRVSR-Net is a rather small network that provides a good tradeoff between quality and inference speed, i.e., the inference of FullHD images at 60 fps is possible, which is difficult to achieve with more complex architectures like the U-Net [41]. An illustration of the network’s building blocks and its topology is given in Fig. 2.

The residual architecture network improves the network’s capability to generalize to new data, as it can focus on generating the residual content [21]. Hence, the input channels are bilinearly upsampled and added to the channels of the output, producing M_t^{EST} , N_t^{EST} , and D_t^{EST} . The only exception is AO_t^{EST} , which is inferred from scratch, as there is no low-resolution input AO map.

Training and Loss Function Characteristics. For training and validation, images of isosurfaces in a supernova simulation (dataset Ejecta) on a 256^3 grid are used. The dataset is rendered at different timesteps to provide the network with a variety of different geometric structures, ranging from very small details to rather smooth low-frequency parts. The input images were subdivided into smaller parts so that multiple inputs can be processed at once and benefit from batch processing in the optimizer.

Weiss et al. [53] compare networks with different weighted combinations of the individual losses described in Sect. 2.2. Figure 3 shows a visual comparison of the surface structures (without AO) that are inferred from the low-resolution input image using these networks.

The evaluation indicates that a network which is trained only with L_1 -losses and a minor objective on temporal coherence gives the best results. This network only sees shaded colors in the temporal coherence loss during the training process and is thus forced to focus primarily on the reconstruction of geometry. Adding a perceptual loss on the normal and AO fields does not lead to any visual differences. This is because the perceptual loss network VGG-19 is trained on color images and does not explicitly consider the relation between geometry and shaded output. Further experiments using an adversarial loss indicate that the GAN produces more high-frequency details

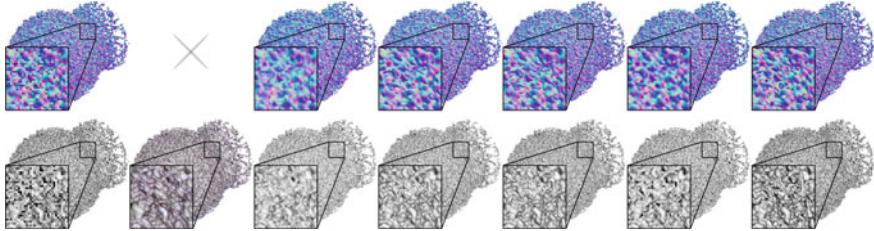


Fig. 3 Comparing networks with different loss function configurations. Top row: the normal map. Bottom row: the shaded output color. Since the “Shaded” network acts directly on color images, a normal map is not used. Left to right: low-resolution input, shaded, L_1 -color, L_1 -geometry (final), perceptual, GAN, and ground truth. This image is adapted from our previous work [53]

that actually decrease the quality of the reconstruction. Furthermore, it significantly increases both training time and memory requirements of the discriminator.

Quality and Performance Evaluation. The following comparison sheds light on the quality and performance of network-based upscaling. The comparison involves images of isosurfaces in Ejecta that were never seen during training, as well as images of isosurface in other datasets (see Fig. 4): A numerical simulation of a Richtmyer–Meshkov (RM) instability at 1024^3 and of a Rayleigh–Bernard process at $1200 \times 1200 \times 80$, CT scans of human anatomies (Skull, Thorax) at a resolution of 256^3 , and a flow simulation (Flow) at 256^3 . For all datasets, even though some of them exhibit geometric features that are different from those in Ejecta, the inference results are very close to the ground truth. The results indicate that the network can effectively infer meaningful details in line with the geometric surface properties, and can furthermore predict a highly accurate distribution of AO values from the inferred geometry.

The quantitative assessment of the quality of IU using the peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) on the normal map in Table 1 confirm the high reconstruction fidelity of network-based inference.

Compared to volumetric raycasting on the GPU without the simulation of AO values, the performance of isosurface image upscaling typically ranges from equal to a factor of 2 faster, including the inference of AO values. Once the renderer also needs to simulate AO, it is outperformed by the upscaling network by two orders of magnitude, due to the computational complexity of AO simulation using ray-based approaches.

4 Upscaling Scenarios—3D Spatial Upscaling

Besides upscaling in image space, spatial upscaling (SU) can also be performed directly in the data space. A possible integration with in situ implementation is as follows. The simulation first outputs, for example, 40% of samples from the entire

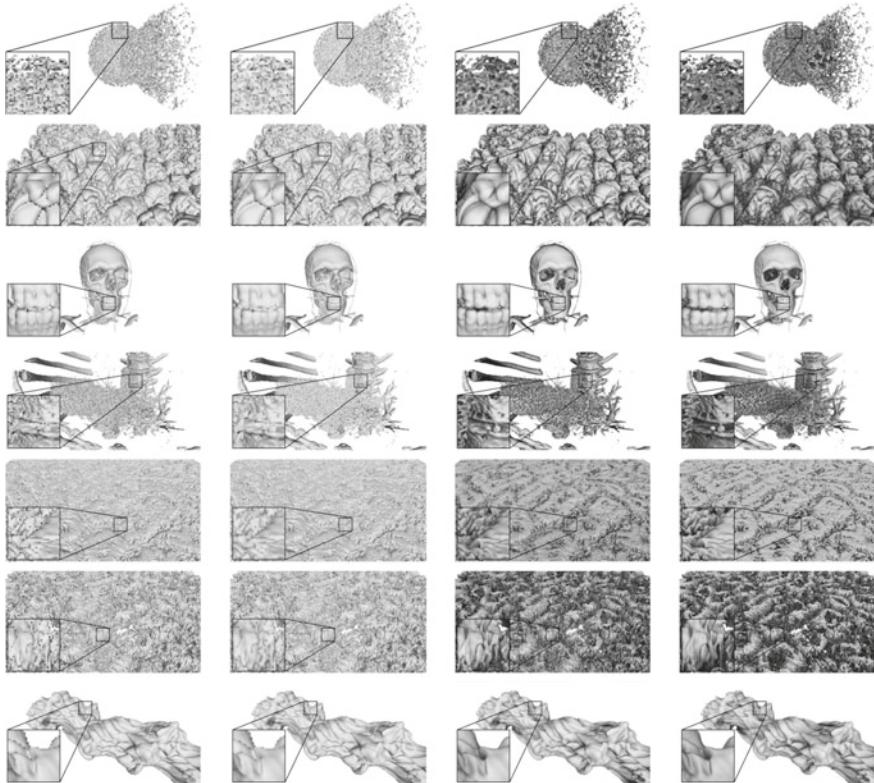


Fig. 4 Comparing upscaling quality on Ejecta, RM, Skull, Thorax, two isosurfaces of a Rayleigh-Bernard process and a single jet. The network was trained only on Ejecta, but on different views than shown. The other datasets were never seen during training. Left to right: input, bilinear, IU, and ground truth with AO. This image is adapted from our previous work [53]

Table 1 Comparing average PSNR and SSIM values of the normal map for different methods

Dataset	PSNR (dB)				SSIM			
	Nearest	Linear	Cubic	IU	Nearest	Linear	Cubic	IU
Clouds	59.18	65.51	56.03	69.88	0.92	0.94	0.88	0.96
Ejecta	60.71	64.99	58.03	69.43	0.91	0.92	0.87	0.95
RM	25.21	27.26	24.02	28.50	0.74	0.76	0.69	0.80
Thorax	43.76	46.74	41.69	49.36	0.75	0.76	0.68	0.78
Skull	25.46	27.36	24.55	29.10	0.91	0.92	0.87	0.95

volume sequence as high-resolution volumes, as the training data. The simulation then resumes and only outputs subsequent low-resolution volumes, as the testing data. For network training, the corresponding low-resolution volumes can be obtained by downsampling the high-resolution ones using the bicubic kernel with a downscaling factor of four (thus, each volume is 1/64 of the original size). Once trained offline, the network can perform real-time inference to predict high-resolution volumes from low-resolution ones.

Given a time-varying volumetric dataset, Han and Wang [19] designed a network architecture similar to tempoGAN [57] for SU of each volume in the sequence. The network includes a generator G and two discriminators (spatial discriminator D_s and temporal discriminator D_t). With this architecture, the network produces spatiotemporally coherent spatial super-resolution of the given volume sequence using adversarial learning. Formally, the goal is to estimate a mapping function \mathcal{F} from a low-resolution volume sequence \mathbf{V}^{LR} to a high-resolution volume sequence \mathbf{V}^{EST} , while taking into account temporal coherence. Namely, $\mathbf{V}^{EST} = \mathcal{F}(\mathbf{V}^{LR})$. The network is then trained by minimizing the loss function that considers (1) *adversarial loss* [35] which trains G with the goal of fooling D_s and D_t , (2) *content loss* [23, 40] which mixes the adversarial loss with a more traditional loss, such as L_2 distance, and (3) *feature loss* [59] which constrains G to produce similar features between synthesized and ground-truth volumes at different scales.

Preliminary Results. For same-variable inference, Fig. 5 compares the isosurfaces extracted from the synthesized volumes via bicubic interpolation and SU. A single isovalue is picked and the results for a single timestep are shown. Clearly, SU generates closer results with respect to the ground truth. Figure 6 compares volume rendering of the synthesized volumes via bicubic interpolation, CNN, and SU. The CNN-based baseline model utilizes a post-upsampling framework [33]. It is clear that CNN generates the worst result while SU yields the best result. Besides same-variable inference, the framework can perform different-variable inference. That is, a variable X of a dataset is used for training. For inference, X is used to infer another variable Y of the same dataset ($X \rightarrow Y$). Such an example is shown in Fig. 7, where

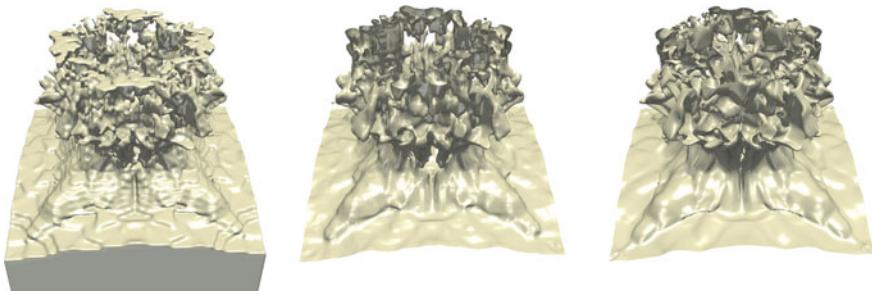


Fig. 5 Comparing same-variable inference results of the Ionization (He+) dataset at timestep 73 using isosurface rendering. The chosen isovalue is -0.84 (the value range is normalized to $[-1, 1]$). Left to right: bicubic, SU, and ground truth

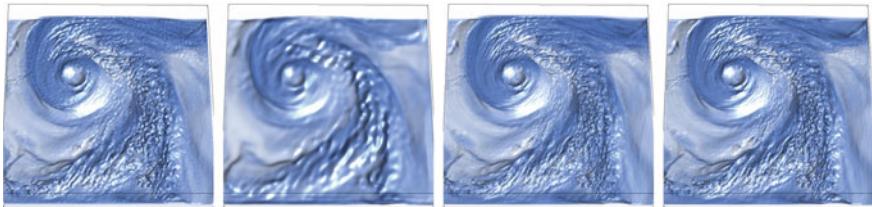


Fig. 6 Comparing same-variable inference results of the Hurricane (QVAPOR) dataset using volume rendering. Left to right: bicubic, CNN, SU, and ground truth

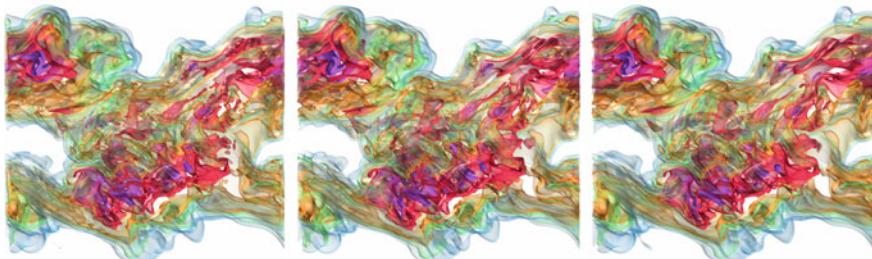


Fig. 7 Comparing different-variable inference results of the Combustion ($MF \rightarrow YOH$) dataset using volume rendering. Left to right: bicubic, SU, and ground truth

the MF variable is used for training and the YOH variable is used for inference. Again, SU produces higher-quality and more detailed visual results than bicubic interpolation.

5 Upscaling Scenarios—Temporal Upscaling

Temporal upscaling (TU) is particularly useful as a large-scale scientific simulation often runs a long sequence but could only afford to store the volume sequence rather sparsely (e.g., every 100th timestep). The upscaling aims to synthesize the intermediate timesteps by providing temporally resolved details to support a more detailed analysis and visualization of dynamic temporal features. A possible integration with in situ implementation is as follows. The simulation also first outputs, for example, 40% of samples from the entire volume sequence as the training data. The simulation then resumes and only sparsely outputs subsequent volumes (says every tenth or hundredth timestep), as the testing data. For network training, it takes pair-wise timesteps at two ends as input and aims to predict intermediate timesteps in between through forward and backward predictions (where intermediate timesteps are known for loss computation). Once training offline, the work can perform real-time inference to predict intermediate timesteps given a pair of timesteps in the testing set (where no intermediate timesteps are known).

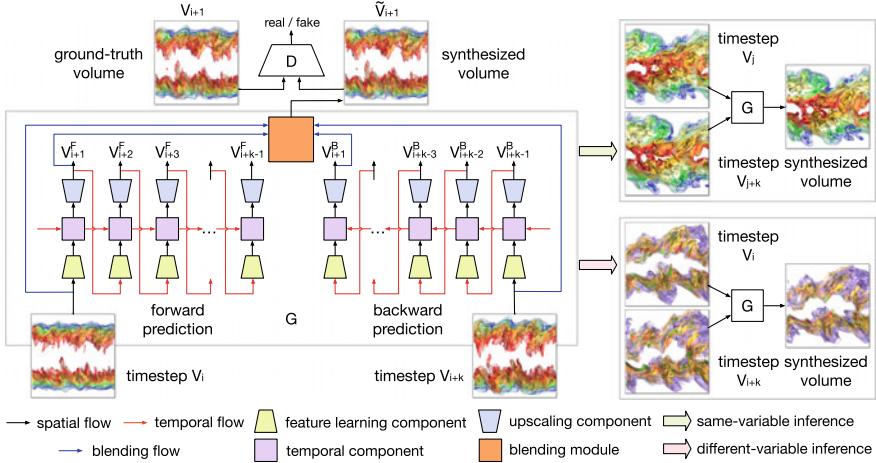


Fig. 8 The TU network has a generator G (including the predicting and blending modules) and a discriminator D . During inference, the network performs either same-variable inference or different-variable inference using G . This image is adapted from our previous work [18]

The TU solution given by Han and Wang [18] uses a *recurrent generative network* that combines RNN and GAN, to generate the intermediate volumes between a given pair of volumes (i.e., two-end timesteps). Given a pair of volumes (V_i, V_{i+k}) from timesteps i and $i+k$ (where $k > 1$), a function \mathcal{F} is sought that satisfies $\mathcal{F}(V_i, V_{i+k}) \approx \mathbf{V}$, where $\mathbf{V} = \{V_{i+1}, V_{i+2}, \dots, V_{i+k-1}\}$ are the intermediate volumes between V_i and V_{i+k} .

Network Design and Loss Function. As sketched in Fig. 8, the network includes a generator G and a discriminator D . G uses two modules: the *predicting module* (\mathcal{F}_{PRED}) and *blending module* (\mathcal{F}_{BLD}), to estimate \mathcal{F} . \mathcal{F}_{PRED} is a volume prediction network that produces a forward prediction \mathbf{V}^{FW} through V_i and a backward prediction \mathbf{V}^{BW} through V_{i+k} , respectively. \mathcal{F}_{PRED} includes three components: (1) *feature learning component* which extracts feature representations from the volumes, (2) *temporal component*, which bridges the spatial and temporal information among different volumes, and *upscale component*, which recovers the volumes from the spatiotemporal features. \mathcal{F}_{BLD} takes V_i, V_{i+k} , and V_{i+j} ($0 < j < k$) from \mathbf{V}^{FW} and \mathbf{V}^{BW} as input, and blends them into the predicted volume $\tilde{\mathbf{V}}$. The discriminator D distinguishes $\tilde{\mathbf{V}}$ from \mathbf{V} . Given an input, D produces a score to indicate whether or not the input is from the real data. That is, $D(\mathbf{V}) \approx 1$ and $D(\tilde{\mathbf{V}}) \approx 0$. Note that the goal of G is to fool D so that D cannot distinguish $\tilde{\mathbf{V}}$ as fake volumes. In this regard, D serves the role of a binary classifier as the score from D can guide G in synthesizing high-quality volumes. Similar to spatial upscaling, the network is trained by minimizing the loss function that considers adversarial loss, content loss, and feature loss.

Architecture Details. With a traditional residual block, the resolution of the input [21] cannot be changed. Therefore, Han and Wang [18] opt to enhance the resid-

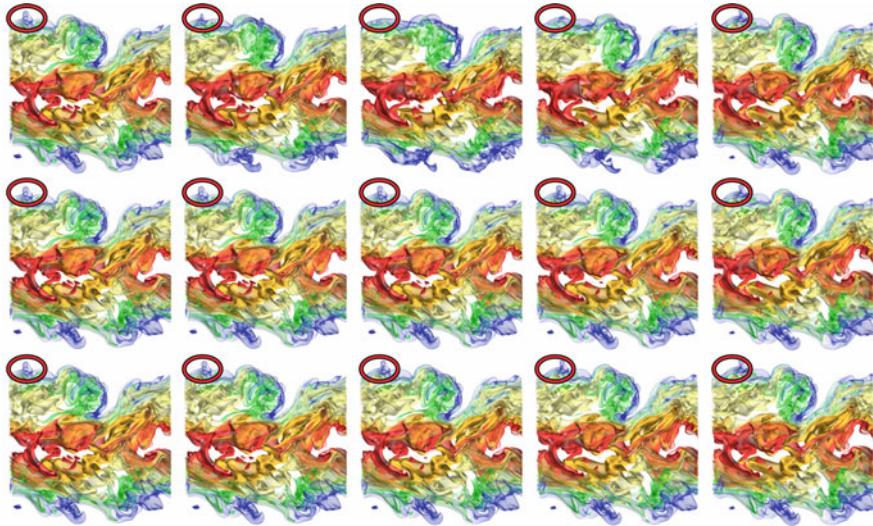


Fig. 9 Comparing same-variable inference results of the Combustion (MF) dataset using volume rendering. Top to bottom: linear, TU, and ground truth. Left to right: five consecutive timesteps. The synthesized results show that TU solution can preserve temporal coherence well while linear interpolation fails to do so (refer to the evolution of a volumetric feature highlighted in ellipses)

ual block by allowing downscaling or upscaling the input. For the feature learning component, each residual block is designed to contain two parts, which are bridged by skip connection. The first part has four convolutional layers, followed by spectral normalization [37] and ReLU [38]. The second part has one convolutional layer, followed by spectral normalization and ReLU. For the temporal component, to enable the network to predict volumes, ConvLSTM [46] is applied to transfer the spatial features into spatiotemporal features. The advantage of ConvLSTM over traditional LSTM [22] lies in its weight-sharing mechanism in convolution. Weight-sharing uses fewer parameters to train ConvLSTM, thus saving memory and speeding up training. For the upscaling component, the spatiotemporal features from ConvLSTM are taken as input and a synthesized volume is output. Although common, using deconvolutional layers to recover resolution from max-pooling or convolutional layers would incur a high computational cost. Therefore, right after the last spectral normalization layer, Han and Wang [18] add the *voxel shuffle* layer, a sub-voxel convolutional layer, for upscaling. Assuming a feature of size $[L, W, H]$ needs to be upscaled with a factor f , voxel shuffle applies a periodic shuffle operation to rearrange the elements of a $[cf^3, L, W, H]$ tensor to a tensor of $[c, fL, fW, fH]$, where c is the number of channels.

Qualitative Results. Figure 9 shows the comparison of volume rendering images of synthesized volumes generated using linear interpolation and inferred using TU. The ground-truth results are displayed for reference. The comparison shows that linear interpolation fails to capture the temporal evolution of features while TU can.

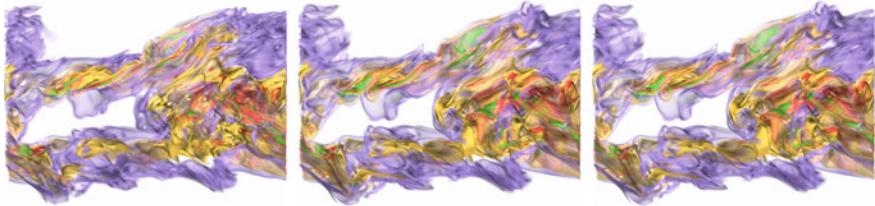


Fig. 10 Comparing different-variable inference results of the Combustion ($\text{MF} \rightarrow \text{HR}$) dataset using volume rendering. Left to right: linear, TU, and ground truth

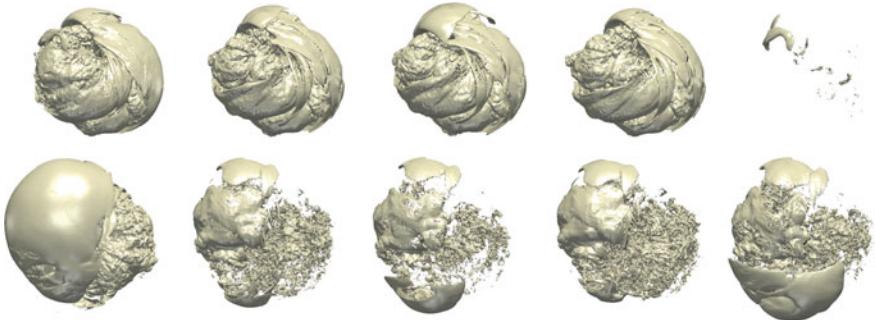


Fig. 11 Comparing same-variable inference results of the Supernova (E) dataset using isosurface rendering. The chosen isovalue is 0.255. Top row: timestep 35. Bottom row: timestep 51. Left to right: linear, TU, ground truth, and the two-end ground-truth timesteps (i.e., 33 and 37 for timestep 35, 49 and 53 for timestep 51)

For example, the ground-truth rendering image sequence shows that the volume feature highlighted in ellipses actually shifts from left to right (note that the feature at the two-end timesteps do not overlap spatially). However, due to the non-overlap of this feature at the two-end timesteps, linear interpolation would interpret this as two separate features: the feature on the left shrinks and disappears while the feature on the right appears and grows. TU can learn temporal evolution for accurate inference of intermediate timesteps. The different-variable inference results shown in Fig. 10 also confirms the effectiveness of TU over linear interpolation.

Figure 11 compares the isosurfaces extracted from the synthesized volumes via linear interpolation and TU. A single isovalue is picked and the results for two different timesteps are shown. Compared with the ground-truth results, it can be seen that at timestep 33, linear interpolation generates a similar isosurface; at timestep 51, linear interpolation fails to construct the isosurface. However, in both cases, TU clearly generates closer results with respect to the ground truth.

Quantitative Results. For quantitative comparison, PSNR and SSIM are used to evaluate the quality of synthesized volumes at the data and image levels, respectively. In addition to linear interpolation, two baseline deep learning solutions based on RNN and CNN are implemented. For the training of RNN, the architecture of TU

Table 2 Comparing average PSNR and SSIM values for different methods

Dataset (variable)	PSNR (dB)				SSIM			
	Linear	RNN	CNN	TU	Linear	RNN	CNN	TU
Combustion (HR)	25.61	26.13	25.72	25.81	0.66	0.70	0.69	0.72
Combustion (MF)	25.12	25.86	25.43	25.62	0.71	0.73	0.73	0.74
Supernova (E)	22.34	24.31	23.81	23.74	0.61	0.64	0.63	0.66
Vortex	26.62	27.42	26.85	26.90	0.73	0.75	0.75	0.75

is followed but the discriminator is excluded. For the training of CNN, the same TU architecture is leveraged but ConvLSTM is removed. Table 2 reports the average PSNR and SSIM values over the entire volume sequence for linear interpolation, RNN, CNN, and TU. At the data level, RNN performs the best in terms of PSNR. Such a result is expected, as RNN is a PSNR-oriented solution. In contrast, TU is also constrained by adversarial and feature losses. At the image level, TU performs the best in terms of SSIM.

6 Concluding Remarks and Future

The current results of deep learning-based data upscaling indicate that such methods have the potential to overcome some of the limitations in current high-performance simulation environments. The trained networks infer well the geometric properties of isosurfaces in 3D scalar fields, and they can even predict missing spatial and temporal features in static and time-resolved fields. As such, there is evidence that deep learning-based upscaling methods can (1) reduce the number of samples that need to be reconstructed (and transmitted in a remote environment) when monitoring a running simulation in situ, (2) reduce the resolution at which data needs to be stored, and (3) reduce the number of timesteps that need to be stored.

In computer vision, a single neural network model trained on various types of images could effectively upscale unseen images from multiple categories. However, this is not the case for scientific data since the training data is limited, and different scientific datasets may not follow a single data distribution (e.g., Gaussian distribution). Still, we believe that one can train a model on a certain type of datasets and later apply it to upscale or infer a different dataset of the same type (e.g., a different variable sequence or ensemble run). For example, Han et al. [20] recently designed V2V, a framework for variable to variable transfer using GAN.

For future work regarding the use of learning-based upscaling for in situ visualization, we envision in particular the following approaches.

When using 2D image-based upscaling in combination with 3D upscaling, it might be possible to let networks infer on the raw data from multiple (low-resolution) views

of selected features in this data, facilitating a feature-based encoding of physical fields. An interesting research question is which features and how many of them are required by a network to infer the original dataset, and whether networks can locally infer the data from these features.

Even more important seems the question whether ANNs can convert the data to a compact feature-preserving representation (a code) that can be permanently stored, and directly decoded by the visualization tool into a visual representation, without having to decode the initial data. In this context, it will be interesting to revisit data compression techniques in light of the so-called “task-dependency principle” from psychology. This principle suggests that a code is optimal if it considers the behavioral goals of a user of this code, which is a perceptual investigation of the information encoded in the data when performing visual data analysis tasks. Following the task-dependency principle, codes should allocate resources according to how the user makes use of the encoded information, and the encoding of data that are irrelevant should be allocated minimal resources. This principle is fundamental to data visualization, since it asks for the reconstruction of data from a perceptual point of view, rather than a signal processing standpoint that argues in terms of numerical accuracy. It is an interesting question whether ANNs can generate such task-dependent, i.e., perception-aware, codes that can be intertwined with a visualization tool in the envisioned way. Answers to this question might be obtained by looking at recent works related to scene representation networks [36] and differentiable rendering [39], where reconstruction networks and networks that learn to generate rendered imagery from the resulting latent space representations have been trained end-to-end.

References

1. Bai, K., Li, W., Desbrun, M., Liu, X.: Dynamic upsampling of smoke through dictionary-based learning (2019). [arXiv:1910.09166](https://arxiv.org/abs/1910.09166)
2. Bengio, Y.: Learning deep architectures for AI. *Found. Trends Mach. Lear.* **2**(1), 1–127 (2009)
3. Caballero, J., Ledig, C., Aitken, A.P., Acosta, A., Totz, J., Wang, Z., Shi, W.: Real-time video super-resolution with spatio-temporal networks and motion compensation (2016). [arXiv:1611.05250](https://arxiv.org/abs/1611.05250)
4. Chaitanya, C.R.A., Kaplanyan, A.S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., Aila, T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.* **36**(4), 98:1–98:12 (2017)
5. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014). [arXiv:1406.1078](https://arxiv.org/abs/1406.1078)
6. Chu, M., Thuerey, N.: Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. Graph.* **36**(4), 69:1–69:14 (2017)
7. Chu, M., Xie, Y., Leal-Taixé, L., Thuerey, N.: Temporally coherent gans for video super-resolution (TecoGAN) (2018). [arXiv:1811.09393](https://arxiv.org/abs/1811.09393)
8. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: Proceedings of European Conference on Computer Vision, pp. 184–199 (2014)
9. Dong, C., Loy, C.C., He, K., Tang, X.: Image super-resolution using deep convolutional networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(2), 295–307 (2016)

10. Dosovitskiy, A., Brox, T.: Generating images with perceptual similarity metrics based on deep networks. In: Proceedings of Annual Conference on Neural Information Processing Systems, pp. 658–666 (2016)
11. Eckert, M.-L., Um, K., Thuerey, N.: ScalarFlow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Trans. Graph.* **38**(6), 239:1–239:16 (2019)
12. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2414–2423 (2016)
13. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Proceedings of Annual Conference on Neural Information Processing Systems, pp. 2672–2680 (2014)
15. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **31**(5), 855–868 (2009)
16. Guo, L., Ye, S., Han, J., Zheng, H., Gao, H., Chen, D.Z., Wang, J.-X., Wang, C.: SSR-VFD: Spatial super-resolution for vector field data analysis and visualization. In: Proceedings of IEEE Pacific Visualization Symposium, pp. 71–80 (2020)
17. Han, J., Tao, J., Zheng, H., Guo, H., Chen, D.Z., Wang, C.: Flow field reduction via reconstructing vector data from 3D streamlines using deep learning. *IEEE Comput. Graph. Appl.* **39**(4), 54–67 (2019)
18. Han, J., Wang, C.: TSR-TVD: temporal super-resolution for time-varying data analysis and visualization. *IEEE Trans. Vis. Comput. Graph.* **26**(1), 205–215 (2020)
19. Han, J., Wang, C.: SSR-TVD: spatial super-resolution for time-varying data analysis and visualization. *IEEE Trans. Vis. Comput. Graph.* (Under Minor Revision) (2020)
20. Han, J., Zheng, H., Xing, Y., Chen, D.Z., Wang, C.: V2V: a deep learning approach to variable-to-variable selection and translation for multivariate time-varying data. *IEEE Trans. Vis. Comput. Graph.* **27**(2) (2021). In Press
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of International Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
22. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
23. Isola, P., Zhu, J.-Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of International Conference on Computer Vision and Pattern Recognition, pp. 1125–1134 (2017)
24. Jo, Y., Oh, S.W., Kang, J., Kim, S.J.: Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3224–3232 (2018)
25. Johnson, J., Alahi, A., Li, F.-F.: Perceptual losses for real-time style transfer and super-resolution. In: Proceedings of European Conference on Computer Vision, pp. 694–711 (2016)
26. Kappeler, A., Yoo, S., Dai, Q., Katsaggelos, A.K.: Video super-resolution with convolutional neural networks. *IEEE Trans. Comput. Imaging* **2**(2), 109–122 (2016)
27. Kim, J., Lee, J.K., Lee, K.M.: Deeply-recursive convolutional network for image super-resolution. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1637–1645 (2016)
28. Kim, J., Lee, J.K., Lee, K.M.: Accurate image super-resolution using very deep convolutional networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1646–1654 (2016)
29. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. *AICHE J.* **37**(2), 233–243 (1991)
30. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)

31. Lai, W.-S., Huang, J.-B., Ahuja, N., Yang, M.-H.: Deep Laplacian pyramid networks for fast and accurate superresolution. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 624–632 (2017)
32. LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
33. Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., Shi, W.: Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of International Conference on Computer Vision and Pattern Recognition, pp. 4681–4690 (2017)
34. Liu, D., Wang, Z., Fan, Y., Liu, X., Wang, Z., Chang, S., Huang, T.: Robust video super-resolution with learned temporal dynamics. In: Proceedings of International Conference on Computer Vision, pp. 2526–2534 (2017)
35. Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: Proceedings of International Conference on Computer Vision, pp. 2813–2821 (2017)
36. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: representing scenes as neural radiance fields for view synthesis (2020). [arXiv:2003.08934](https://arxiv.org/abs/2003.08934)
37. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: Proceedings of International Conference for Learning Representations (2018)
38. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of International Conference on Machine Learning, pp. 807–814 (2010)
39. Nimier-David, M., Vicini, D., Zeltner, T., Wenzel, J.: Mitsuba 2: a retargetable forward and inverse renderer. *ACM Trans. Graph.* **38**(6), 203:1–203:17 (2019)
40. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of International Conference on Computer Vision and Pattern Recognition, pp. 2536–2544 (2016)
41. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. In: Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234–241 (2015)
42. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986)
43. Sajjadi, M.S.M., Schölkopf, B., Hirsch, M.: EnhanceNet: single image super-resolution through automated texture synthesis. In: Proceedings of IEEE International Conference on Computer Vision, pp. 4501–4510 (2017)
44. Sajjadi, M.S.M., Vemulapalli, R., Brown, M.: Frame-recurrent video super-resolution. In: Proceedings IEEE Conference on Computer Vision and Pattern Recognition, pp. 6626–6634 (2018)
45. Schmidhuber, J.: Deep learning in neural networks: an overview. *Neural Netw.* **61**, 85–117 (2015)
46. Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., Woo, W.-C.: Convolutional LSTM network: a machine learning approach for precipitation Nowcasting. In: Proceedings of Advances in Neural Information Processing Systems, pp. 802–810 (2015)
47. Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., Wang, Z.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1874–1883 (2016)
48. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
49. Tai, Y., Yang, J., Liu, X.: Image super-resolution via deep recursive residual network. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3147–3155 (2017)
50. Tao, X., Gao, H., Liao, R., Wang, J., Jia, J.: Detail-revealing deep video super-resolution. In: Proceedings of IEEE International Conference on Computer Vision, pp. 4482–4490 (2017)
51. Tong, T., Li, G., Liu, X., Gao, Q.: Image super-resolution using dense skip connections. In: Proceedings of IEEE International Conference on Computer Vision, pp. 4809–4817 (2017)

52. Wang, X., Chan, K.C., Yu, K., Dong, C., Loy, C.C.: EDVR: Video restoration with enhanced deformable convolutional networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (2019)
53. Weiss, S., Chu, M., Thuerey, N., Westermann, R.: Volumetric isosurface rendering with deep learning-based super-resolution. *IEEE Trans. Vis. Comput. Graph.* (Accepted) (2019)
54. Weiss, S., Işık, M., Thies, J., Westermann, R.: Learning adaptive sampling and reconstruction for volume visualization (2020). [arXiv:2007.10093](https://arxiv.org/abs/2007.10093)
55. Werhahn, M., Xie, Y., Chu, M., Thuerey, N.: A multi-pass GAN for fluid flow super-resolution. *Proc. ACM Comput. Graph. Inter. Tech.* **2**(1), 10:1–10:21 (2019)
56. Xiao, X., Wang, H., Yang, X.: A CNN-based flow correction method for fast preview. *Comput. Graph. Forum* **38**(2), 431–440 (2019)
57. Xie, Y., Franz, E., Chu, M., Thuerey, N.: tempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. Graph.* **37**(4), 95:1–95:15 (2018)
58. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising (2016). [arXiv:1608.03981](https://arxiv.org/abs/1608.03981)
59. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 586–595 (2018)
60. Zhou, Z., Hou, Y., Wang, Q., Chen, G., Lu, J., Tao, Y., Lin, H.: Volume upscaling with convolutional neural networks. In: Proceedings of Computer Graphics International, pp. 38:1–38:6 (2017)

Scalable CPU Ray Tracing for In Situ Visualization Using OSPRay



**Will Usher, Jefferson Amstutz, Johannes Günther, Aaron Knoll,
Gregory P. Johnson, Carson Brownlee, Alok Hota, Bruce Cherniak,
Tim Rowley, Jim Jeffers, and Valerio Pascucci**

Abstract In situ visualization increasingly involves rendering large numbers of images for post hoc exploration. As both the number of images to be rendered and the data being rendered are large, the scalability of the rendering component is of key concern. Furthermore, the renderer must be able to support a wide range of data distributions, simulation configurations, and HPC systems to provide the flexibility required for a portable, general purpose in situ rendering package. In this chapter, we discuss recent developments in OSPRay’s support for MPI-parallel applications to provide a flexible and scalable rendering API, with a focus on how these developments can be applied to enable scalable, high-quality in situ visualization.

1 Introduction

Rendering images is a common task performed in situ, as the resulting images are small in size compared to the full data sets and can be saved out frequently and viewed post-hoc. The images rendered in situ have been used in a number of ways, e.g., to produce static visualizations [41, 42], movies [14], Cinema databases [3], and other explorable extracts [17, 28, 52, 57], or for interactive visualization during the simulation [23, 40, 43, 45, 55]. Although these applications differ widely in the

W. Usher (✉) · J. Günther · A. Knoll · G. P. Johnson · C. Brownlee · A. Hota · B. Cherniak ·
J. Jeffers

Intel Corporation, Mountain View, USA
e-mail: will.usher@intel.com

J. Amstutz
Intel, Now with Nvidia, Santa Clara, USA

T. Rowley
Intel, Now with Google, Mountain View, USA

V. Pascucci
SCI Institute, University of Utah, Salt Lake City, USA
e-mail: pascucci@sci.utah.edu

number and type of images rendered, they share some common requirements for the underlying renderer used to produce the images.

First, the renderer must support distributed rendering across a cluster, as the simulation data are frequently too large to aggregate to a single node for processing. Moreover, this distributed rendering component must scale with the simulation to prevent the visualization component from becoming a bottleneck. It is also desirable for the renderer to support more advanced effects to provide high-quality images, be extendable with support for new geometric and volumetric representations, and be capable of sharing memory directly with the simulation.

In this chapter, we will discuss how recent developments in OSPRay’s distributed rendering capabilities [44] can be leveraged to enable scalable *in situ* visualization applications. OSPRay [49] is an open-source, scalable, portable ray tracing library. OSPRay provides users powerful scalable distributed rendering capabilities, with support for a range of data configurations, representations, and high-quality local illumination effects. The API also provides the flexibility needed to interoperate with different simulation data layouts and configurations. We first provide a brief overview of OSPRay and the new 2.0 API (Sect. 2), and then discuss the distributed rendering capabilities provided by OSPRay’s distributed device (Sect. 3), evaluate the scalability of the current distributed renderer (Sect. 4), and discuss some example use cases (Sect. 5).

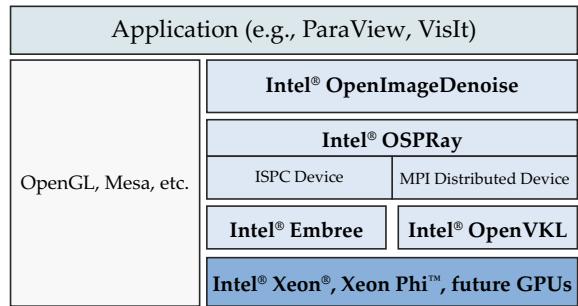
2 OSPRay

Although originally conceived as an API and CPU-based engine for scientific visualization rendering, OSPRay has evolved into a higher level general-purpose API for rendering geometric and volumetric data with multi-node distributed rendering. OSPRay sits at the highest level of Intel’s oneAPI Rendering Toolkit software stack [24]. The libraries in the Rendering Toolkit follow a paradigm of software-defined visualization, in which features are driven by user needs, and implemented in software before potentially being specialized in hardware. The lower level libraries in the Rendering Toolkit are:

- Embree [51], a collection of high-performance ray tracing kernels, providing efficient acceleration structure traversal and geometry intersection methods;
- Open Volume Kernel Library (OpenVKL) [26], which provides high-performance volume traversal and sampling kernels for visualization and film; and
- OpenImageDenoise [25], a recursive autoencoder neural-net-based solution for postprocessing noisy ray-traced and path-traced images.

Figure 1 illustrates this software stack. OSPRay and the other components of the oneAPI Rendering Toolkit are released open-source under the Apache 2.0 license.

Fig. 1 The Intel® oneAPI Rendering Toolkit [24] software stack. OpenImageDenoise, OSPRay, Embree, and OpenVVKL are all released under the Apache 2.0 license



2.1 OSPRay 2.0 Features for In Situ Visualization

OSPRay 2.0, released in January 2020, features several improvements over v1.x. At a high level, these include architectural changes centered around API sustainability, as well as integration of new lower level libraries in the oneAPI Rendering Toolkit. Highlights specifically relevant to in situ visualization include:

- Improved volume rendering performance and quality through the integration of OpenVVKL and more advanced renderers;
- Asynchronous frame rendering, which enables applications to continue processing while OSPRay renders, and to render multiple frames simultaneously;
- The separation of an object’s underlying data from its appearance, through the introduction of volumetric and geometric models; and
- Modules can implement new distributed rendering algorithms to extend OSPRay’s built in renderer.

2.2 OSPRay Actors and Objects

OSPRay provides a hierarchy of OSPObjects for specifying data, geometry, scene, camera, and lighting information. At the lowest level, multidimensional data arrays are specified through OSPData objects, which are set as parameters on OSPGeometry or OSPVolume objects to specify the underlying data for the primitive, e.g., vertex positions, voxels, etc. The OSPData objects can share pointers with the application, providing a zero-copy interface. Geometries and volumes are then associated with the desired appearance information (e.g., materials, transfer functions) through an OSPGeometricModel or OSPVolumetricModel. Groups of such models can then be positioned in the world to setup the scene to be rendered, along with a set of lights.

To render the scene, the application creates an OSPRenderer, selecting the rendering modality desired (e.g., scientific visualization, path tracing), an OSPCamera to place the view point, and an OSPFrameBuffer to output the rendered image to. The

world, renderer, camera, and framebuffer are then passed to `ospRenderFrameAsync` to begin rendering.

2.3 *OSPRay Modules*

OSPRay is designed to be programmable and expandable via custom user modules. Modules serve applications that require additional functionality beyond that provided by the built-in objects in OSPRay, but do not require modifying the API itself. These modules can be used for experimental development and bespoke applications, involving, for example, large particle data structures [50], custom geometric primitives [21, 29, 46], volumes [48], or tailoring OSPRay to the needs of a larger visualization system, as was done for VisIt [54].

2.4 *OSPRay Devices*

The underlying implementation of the OSPRay API is provided through “Devices,” which map OSPRay API calls to an execution backend (e.g., CPUs, GPUs). OSPRay’s current CPU backend, the `ISPCDevice`, uses the Intel SPMD Program Compiler (ISPC) [38] as a data-parallel kernel programming language, which allows for efficiently vectorized code on Intel and other CPU architectures. However, the OSPRay API is designed to be independent from the backend implementation, and is not tied to CPUs exclusively. Efforts are currently underway to implement a GPU backend and generalize the API to foster contributions from other communities and vendors. In the context of distributed and *in situ* visualization, OSPRay’s MPI module provides an MPI-based device, the `MPIDistributedDevice`, that coordinates execution of a local device (currently the `ISPCDevice`) across multiple nodes. The MPI Distributed Device is discussed in detail in the following section.

3 Distributed Rendering in OSPRay

OSPRay’s distributed rendering capabilities are provided through the distributed device in the MPI module. The distributed device allows applications to make OSPRay API calls on each rank to set up a scene containing unique or shared data across the ranks. Rendering is performed collectively across the ranks using the distributed rendering algorithms described by Usher et al. [44]. We provide an overview of these algorithms (Sect. 3.1), but focus primarily on the end application perspective. We present an overview of the distributed API and a distributed application (Sect. 3.2), then discuss in detail key aspects of the API for *in situ* visualization: zero-copy data sharing (Sect. 3.3), asynchronous rendering (Sect. 3.4),

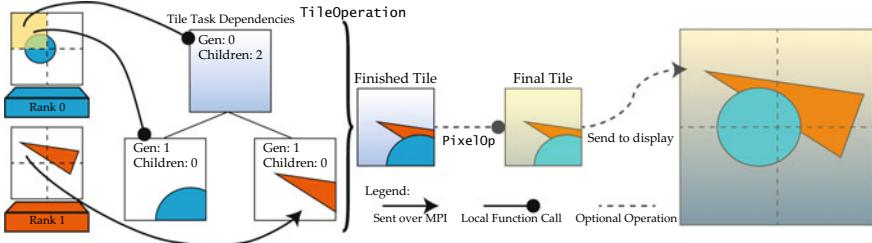


Fig. 2 An example of the Distributed FrameBuffer’s tile processing pipeline in a data-parallel renderer. Dependencies are specified on the fly per-tile and can be further extended by child tiles. To compute the highlighted tile owned by rank 0, the owner renders a background color tile that tells the tile operation to expect two child tiles in generation 1. These tiles correspond to the rendered data from each rank. After these tiles are received, they are composited, optionally processed by a pixel operation, and sent to rank 0 for display. This image is reproduced from our previous work [44]

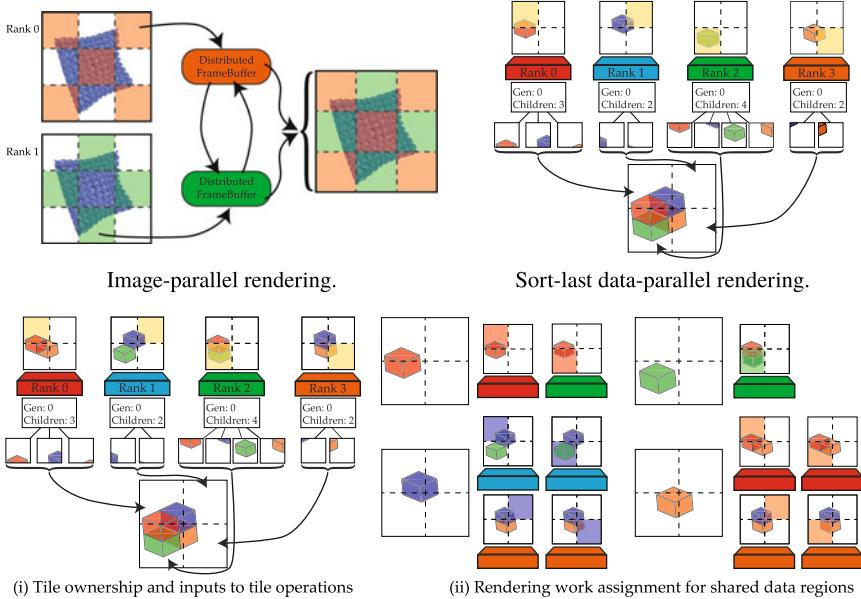
rendering more complex data distributions (Sect. 3.5), and extending the API through modules (Sect. 3.6).

3.1 The Distributed FrameBuffer and Rendering Algorithms

OSPRay’s distributed renderers are built on top of a common distributed framebuffer abstraction, the Distributed FrameBuffer (DFB) [44]. The DFB is an asynchronous image-processing framework for multi-node rendering that works by distributing the ownership of tiles in the image among the ranks (Fig. 2). Each rank is responsible for performing compositing operations for the tiles it is assigned and can render tiles that are owned by other ranks, allowing the rendering work distribution to differ from the compositing work distribution. Tiles rendered on each rank are sent over the network to the rank that owns the image tile for compositing. The DFB does not assume a specific compositing algorithm, and provides the flexibility to implement image-, data-, and hybrid-parallel renderers.

A renderer using the DFB requires two components: a local renderer run on each rank that produces input tiles to the processing pipeline, and a tile operation run by the DFB in the pipeline to combine input tiles into a single finished tile. In contrast to existing special-purpose approaches for image [7, 12, 27, 47] and data [16, 19, 20, 22, 30, 31, 33, 37, 56] parallel rendering, OSPRay’s DFB provides the flexibility to cover the spectrum of image- and data-parallel work distributions, without requiring changes to the application.

The local renderer’s work distribution is not tied to the DFB’s tile distribution, as the DFB will route tiles as needed using MPI. It is also possible for multiple ranks to render input tiles for a single image tile. The tile operation is responsible for defining how the input tiles should be combined to produce an output tile, e.g., by averaging



Sort-last hybrid-parallel rendering. (i) Each rank stores two data regions and (ii) shares the rendering work for each region among those ranks sharing the region. The compositing tasks remain the same as the standard data-parallel case (upper right) but can achieve better load balance.

Fig. 3 The spectrum of image- to data-parallel rendering supported by OSPRay’s distributed renderer. Each configuration uses the same DFB infrastructure and requires no code changes to the end user application. The sort-last data-parallel rendering image is reproduced from our previous work [44]

the inputs together or performing sort-last compositing. Each input tile is specified as a member of some generation of inputs and as having some number of additional children, which can be used to specify a tree or ordering of the tiles as needed for sort-last compositing (Fig. 3).

OSPRay currently provides two distributed renderers built on top of the DFB, an image-parallel renderer and a data-parallel renderer. The image-parallel renderer (Sect. 3.1) is well suited to rendering replicated data or data that can be fetched as needed by each rank through some caching mechanism. Rendering work is distributed among the ranks by assigning each rank a unique subset of tiles to render. The tile operation takes a single input tile for each image tile which is passed through as the finished tile.

The data-parallel renderer (Sect. 3.1) is a sort-last renderer for distributed data, where each rank is assigned a different piece of data to render. Each rank renders its local data to the tiles it projects to, and for those tiles owned by the rank it is responsible for providing a background tile. The background tile is filled with the scene background color, and tells the tile operation how many input tiles to expect and composite together, i.e., how many ranks’ data projects to that tile. After receiving

the set of input tiles, the tile operation sorts them by their sort order and composites them to produce the finished tile. It is possible for a rank to have more than one piece of unique data, in which case an independent partial tile is rendered for each image tile touched by each data region.

The distributed renderer is also capable of rendering hybrid data distributions (Sect. 3.1), where the data is partially replicated among the ranks. In a hybrid configuration, the rendering work is distributed image-parallel among those ranks sharing each piece of data and data-parallel among ranks with different data, allowing applications to mix the standard image- or data-parallel rendering algorithms. Hybrid configurations can be used to achieve better load balance when rendering large data (see [18, 44]), or to combine replicated and distributed data in the same scene. The tile operation is the same sort-last compositing one used in the data-parallel renderer, and the only change is in which rank provides an input tile for a specific pair of data region and image tile.

The data- and hybrid-parallel renderers determine the scene distribution solely by inspecting the bounding boxes of the data on each node, referred to as regions. The region information is used by each rank to set the number of children to expect on the background tiles it renders for sort-last compositing, and to determine if two ranks are sharing the same piece of data and should thus share the rendering work using the hybrid-parallel renderer. The renderer works only at the abstract level of the bounds of each rank’s data and imposes no restrictions on the types of volumes, geometries, etc., which are contained within these bounds. The sole requirement is that the bounds of different regions do not overlap, and for shared regions should be identical, to allow the compositor to order the partial image tiles and find shared regions correctly.

3.2 *The Distributed API*

To use the distributed API, applications can load the MPI module and select the MPI distributed device (Listing 1). Each rank can then make independent API calls to set up its local geometries and volumes, after which the distributed scene can be rendered collectively using the algorithms described above. As OSPRay’s data-parallel renderer needs only the bounding boxes of each rank’s portion of the data, we are able to expose an API for distributed rendering that is nearly identical to local rendering, and that can support existing user geometry and volume modules without changes.

The bulk of the distributed API is similar to local rendering in OSPRay, and should appear familiar to existing users (Listing 2). However, the distributed API makes a distinction between “distributed” and “local” objects. Distributed objects track the global rendering configuration or scene layout (e.g., the framebuffer, renderer, world, and futures), and they must be created and committed collectively by the ranks, to ensure the objects are synchronized properly. With the exception of the list of instances and regions on the world, each rank should set the same parameter values

```

#include <ospray/ospray.h>
#include <ospray/ospray_util.h>

int main(int argc, char **argv) {
    ...
    // If not using asynchronous rendering, MPI_THREAD_SERIALIZED is sufficient
    int thread_support = 0;
    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &thread_support);

    // Load the MPI module and select the distributed device
    ospLoadModule("mpi");
    OSPDevice mpiDevice = ospNewDevice("mpiDistributed");
    ospDeviceCommit(mpiDevice);
    ospSetCurrentDevice(mpiDevice);

```

Listing 1: Initializing the MPI distributed device to use the distributed API in an MPI-parallel application. Thread multiple support is required for asynchronous rendering if the application will make MPI calls during rendering; otherwise, thread serialized support is sufficient.

on distributed objects. Local objects represent local data, geometries, and volumes and are created independently by the ranks. When using the distributed API, only rank 0 will be able to map the framebuffer to access the rendered image.

For rendering data- or hybrid-distributed configurations, applications must use the `mpiRaycast` renderer, which implements the rendering algorithms discussed above. The distributed API also adds an optional “regions” parameter to the `OSPRayWorld`, that is used to specify one or more boxes that bound the local data owned by the rank to configure the compositor. The region list can be used to clip ghost zones, or to partition nonconvex local data distributions into sets of convex regions that can be composited by OSPRay’s sort-last compositor. If no regions are specified, the union of the bounds of the local geometries and volumes is implicitly set as the rank’s region bounds. It is also valid for some ranks to have no data in the scene, and thus no regions or instances in with world.

Finally, it is also possible to specify a subset of ranks that should participate in rendering by specifying the communicator OSPRay should use. By default, OSPRay will use all ranks in the MPI world; however, OSPRay uses an MPI + threads model for parallelism, which, when integrating with an MPI-only simulation, may lead to oversubscribing the nodes. Instead, the application could create a subcommunicator with one rank per-node and set this as OSPRay’s communicator. The data from other ranks on the node can then be gathered to this OSPRay rank using shared memory.

3.2.1 Overview of a Distributed Application

A distributed application begins by setting up the local OSPRay objects representing the geometries and volumes it wishes to render (Listing 2). The creation of local objects is identical to single-node rendering with OSPRay.

```

OSPVolume volume = ospNewVolume("structuredRegular");
float *data = // Local 64$^3 simulation volume data
OSPData volumeData = ospNewSharedData(data, OSP_FLOAT, 64, 64, 64);
ospSetObject(volume, "data", volumeData);
ospSetVec3i(volume, "dimensions", 64, 64, 64);
ospCommit(volume);

OSPTerTransferFunction tfcn = ospNewTransferFunction("piecewiseLinear");
... // Set color and opacity params for transfer function
ospCommit(tfcn);

OSPVolumetricModel volumeModel = ospNewVolumetricModel(volume);
ospSetObject(volumeModel, "transferFunction", volumeModel);
ospCommit(volumeModel);

OSPGGroup group = ospNewGroup();
OSPData volumeList = ospNewSharedData1D(@volumeModel, OSP_VOLUMETRIC_MODEL, 1);
ospSetObject(group, "volume", volumeList);
ospCommit(group);

OSPIInstance instance = ospNewInstance(group);
// Optional 3x4 column major transform to place this brick in the global scene
float tfm[12] = {...};
ospSetParam(instance, "xfm", OSP_AFFINE3F,
tfm); ospCommit(instance);

```

Listing 2: Local object setup for distributed rendering. Each rank has a 64^3 volume brick from the simulation, and shares a pointer to this data with OSPRay to create the volume object. Local object setup is identical to using OSPRay’s existing local rendering APIs.

To set up the distributed objects required to render the scene, the application first collectively creates an **OSPWorld** (Listing 3) and adds its local instances to the world, which provides each rank a shared view of the distributed scene. The application then collectively creates an **mpiRaycast** renderer, which will use the region information from the world to configure the compositor. Finally, the application collectively creates a framebuffer to store the output image. To render a frame the ranks collectively call **ospRenderFrame** as in local rendering, and wait for completion.

3.3 Sharing Data with the Application

When rendering in situ, it is highly desirable for the visualization code to share memory directly with the simulation, to minimize memory pressure on each node and the time spent copying data. With OSPRay, the application can directly share memory with the renderer through the creation of shared **OSPData** objects, which share a pointer directly with the application. OSPRay 2.0 has made it easier for applications to share data with OSPRay, in both local and distributed rendering use

```

OSPIstance instances[] = // List of local instances
OSPData instance_list = ospNewSharedData1D(instances, OSP_INSTANCE, num_instances);
box3f regions[] = // Optional list of regions
OSPData region_list = ospNewSharedData1D(regions, OSP_BOX3F, num_regions);

OSPWorld world = ospNewWorld();
ospSetObject(world, "instance", instance_list);
ospSetObject(world, "regions", region_list);
ospCommit(world);

OSPRenderer renderer = ospNewRenderer("mpiRaycast");
ospCommit(renderer);

OSPFrameBuffer fb = ospNewFrameBuffer(1024, 1024, OSP_FB_SRGBA,
                                         OSP_FB_COLOR | OSP_FB_ACCUM);
OSPFuture future = ospRenderFrame(fb, renderer, camera, world);
ospWait(future, OSP_TASK_FINISHED);

```

Listing 3: Distributed object setup for distributed rendering. The ranks collectively create a world and add their local instances to it. The renderer and framebuffer are also created collectively, after which the scene can be rendered across the ranks.

cases. The new APIs for passing data to OSPRay, `ospNewSharedData*D`, allow applications to specify a byte stride to be used between elements, rows, or planes in 1D, 2D, or 3D arrays. Thus, if the simulation data are not natively ordered or packed to match the layout required by the geometry or volume type, the stride can be used to provide a view into the data that does match the layout. **OSPData** are local objects and support the same memory sharing and parameters as in local rendering. The **OSPData** objects and the referenced data are local to each rank and are not transferred between ranks, ensuring predictable memory and network usage patterns by the renderer.

3.4 Asynchronous Rendering

OSPRay 2.0 has recently switched to asynchronous rendering by default, enabling applications to render multiple images in parallel, or to more easily perform other computation during rendering. The `ospRenderFrame` function returns an **OSPFuture**, which can be tested or waited on to monitor the progress of the frame (Listing 3). Asynchronous rendering is also supported in the distributed API, and is especially useful for *in situ* image database generation for applications such as Cinema [3].

To generate an image database, the application renders a large number of different view points and scene parameter configurations, which can then be explored interactively post hoc using, e.g., Cinema. The cost of rendering each image on each rank depends on the camera position and scene configuration, potentially leading to underutilizing some nodes. With asynchronous rendering, multiple images can

be rendered simultaneously, allowing for better utilization of each node’s compute capabilities. However, care should be taken to avoid oversubscribing the node by limiting the number of frames being rendered in parallel at a time.

To render multiple frames in parallel, the application will need one framebuffer for each active render, along with per-render cameras, groups, instances, and worlds for each active render with different parameters for the respective objects. The underlying geometric or volumetric data are kept distinct from their appearance information, so creating new geometric or volumetric models to change the object’s appearance does not require duplicating the underlying data. OSPRay objects used in an asynchronous render should not be modified until the render has completed to avoid conflicts.

Although OSPRay’s MPI communication is performed by a single thread, if the application will make MPI calls during asynchronous rendering, the MPI runtime must support thread multiple; otherwise, thread serialized is sufficient.

3.5 Configuring the Scene Distribution Using Regions

The optional regions parameter on the `OSPWorld` allows each rank to set one or more bounding boxes that contain the data it owns for rendering. Arbitrary geometric and volumetric data can be placed within the regions, including objects provided by user modules. Each region corresponds to a unique piece of data that is owned by the rank. Each rank can have zero or more such regions, and can share ownership of regions with other ranks. When rendering, only data contained within the region bounds on a rank will be visible to camera rays, though data outside are available for volume interpolation and secondary rays. The regions allow applications to provide ghost zones for interpolation and AO, render nonconvex and nondisjoint data distributions (Sect. 3.5.1), or share data between ranks for hybrid-distributed and data-replicated rendering (Sect. 3.5.2).

3.5.1 Using Regions to Clip Ghost Zones and Ensure Convexity

Ghost zones, or sometimes “halos,” are used in distributed rendering to ensure the image produced by the distributed renderer matches that which would be produced by a renderer running on one rank with the entire data set. For example, an extra layer of voxels is stored around the volume subpiece owned by the rank to avoid interpolation artifacts between neighboring ranks. Similarly, when rendering secondary effects such as ambient occlusion, additional ghost geometries are used to prevent bright spots appearing at rank boundaries where geometry would otherwise be missing. Regions are only used to clip camera rays, allowing secondary rays to still intersect any ghost geometry which may exist outside the region bounds.

The regions can also be leveraged to allow rendering nonconvex or nondisjoint data distributions that would otherwise require redistributing the data when using other sort-last compositors. OSPRay, as with other sort-last compositors, requires

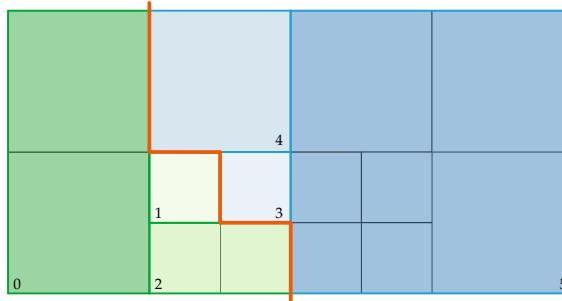


Fig. 4 An example of virtually partitioning an Octree AMR data set using regions. The data are distributed among two ranks along the orange line and virtually partitioned using regions to create a set of disjoint bricks that can be composited by OSPRay’s DFB. The three regions created by each rank only virtually partition the data, leaving the underlying data unchanged

the partial images being composited to correspond to convex, disjoint bricks of data. Without this requirement, sorting and ordering the partial images to produce a correct final image may not be possible. However, OSPRay’s distributed API allows each rank to specify one or more regions that it owns. Each such region is treated as if it were owned by a unique rank, and rendered to produce an independent set of partial image tiles that are given to the compositor. Nonconvex data-distributions, such as those potentially arising in adaptive mesh refinement simulations [2, 4, 5, 8, 10, 32, 35], can be virtually partitioned into a set of disjoint bricks that are suitable for sort-last compositing (Fig. 4). This partitioning takes place only when clipping rays against the region bounds in the renderer, and requires no movement or adjustment of the underlying data.

3.5.2 Rendering of Partially and Fully Replicated Scenes

The regions list can also be used to tell OSPRay that two ranks are sharing one or more regions, by specifying the same region bounds on each rank. If additional memory is available to duplicate some or all of the scene data, ranks can share regions for better load balancing, or even share the entire scene to enable high-quality path-traced rendering.

During the region exchange step when the world is committed, these shared regions are found and merged in the global list of regions. Each unique region in the list corresponds to a unique piece of data that must be rendered and composited independently. When multiple ranks are found to share the same region, they switch to the hybrid-distributed rendering algorithm (Fig. 3). For example, if a region is found to be especially expensive to render, that region can be duplicated onto other ranks for load balancing, leaving the rest of the data as is.

If enough memory is available to fit the entire scene on each rank, the same data and regions can be specified on all ranks, in which case OSPRay will switch to the

Fig. 5 A visualization of the 1024^3 Miranda [11] data set rendered interactively using OSPRay’s volumetric path tracer, with support for scattering and volumetric lighting effects

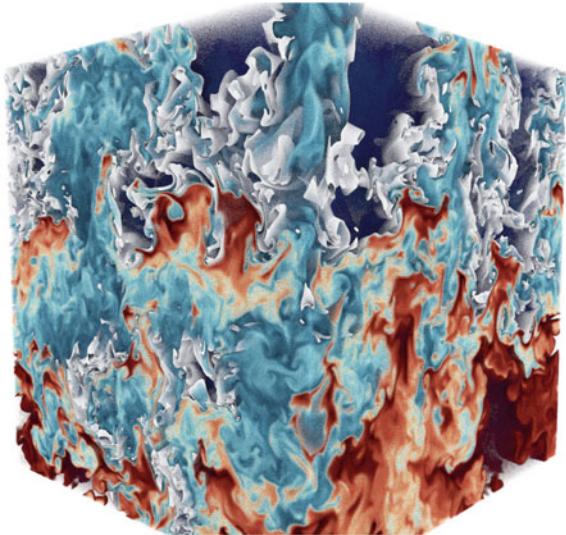


image-parallel rendering algorithm (Fig. 3). The image-parallel renderer can use any of OSPRay’s local renderers, e.g., the path tracer, to provide advanced secondary illumination effects (Fig. 5). Although in a tightly coupled in situ application there is unlikely to be sufficient available memory to replicate the entire scene on each node, it may be possible for a loosely coupled application that is run on a distinct set of nodes.

3.6 Extending OSPRay’s Distributed API with Modules

A key goal in the design of OSPRay’s distributed API was to allow existing OSPRay user modules, which extend OSPRay with new geometries and volumes, written for single-node rendering, to work seamlessly in a distributed setting as well. Creating local OSPRay objects in the distributed API works exactly as in local rendering, and modules extending OSPRay with local objects should work directly in the distributed API without change. For example, a user module that adds support for tensor glyph geometry would be used exactly as in single-node rendering, with the only change being that each rank now specifies only its local data. The application using the module will likely want to duplicate glyphs crossing rank boundaries for continuity at boundaries, and use regions to clip each rank’s glyphs to the region it owns for rendering, to avoid ranks’ overlapping each other.

User modules can also add new distributed objects to extend features specific to OSPRay’s distributed API, e.g., adding new distributed rendering algorithms. New distributed renderer can be implemented in OSPRay by extending the `Distributed Renderer` and overriding the methods to render a specific region for some tile and to

return the tile operation used by the renderer. The “region for tile” rendering method will be used to render local data to the image tiles they project to. The tile operation will be executed in the DFB pipeline to combine the rendered tiles into a finished image tile (Fig. 2).

By default, renderers will use the scheduler included with the MPI module, which supports the spectrum of data-, hybrid-, and image-parallel rendering modes (Sect. 3.1). However, developers wishing to use a custom scheduler can extend this aspect as well, and use the information about the distributed scene provided in the world to inform their scheduler. The existing rendering and communication infrastructure in OSPRay and the flexibility provided by the API should make it possible to implement variants of Galaxy [1], SpRay [36], or other rendering algorithms that send rays or data [6, 34, 39] as modules, to provide support for global illumination effects on distributed data.

4 Scalability

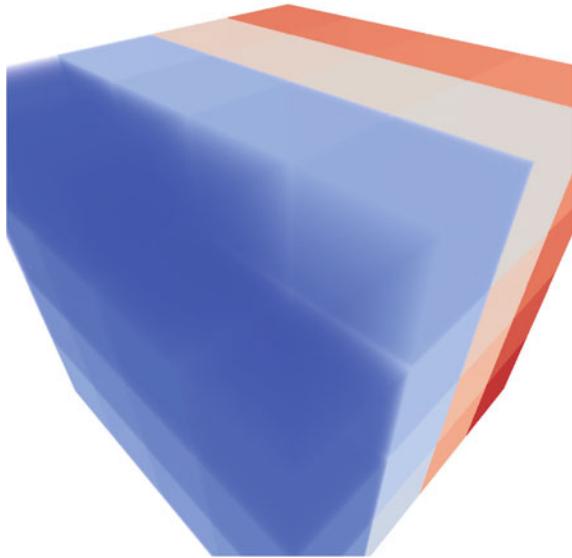
We evaluate the scalability of OSPRay’s Distributed FrameBuffer and rendering algorithms using a similar set of benchmarks as in [44]. However, our evaluation is conducted using the 2.0 version of OSPRay, which includes numerous changes, fixes, and improvements to both the local and distributed renderers, which can affect performance compared to the results in [44]. Moreover, we compare performance against a more typical use case of OSPRay with IceT [33], which uses the scientific visualization renderer locally and IceT for compositing, instead of a modified version of the MPI raycast renderer that locally renders only the tiles the ranks’ data projects to, as done in [44]. Although the latter is better tuned to benchmark the compositing differences between the two approaches, the former is more representative of the performance achievable by end users when using OSPRay for local rendering and IceT for compositing. The former approach is currently used in ParaView and VisIt. See Wu et al. [54] for further details on this approach as used in VisIt.

The benchmark application generates a 128^3 volume brick on each rank and renders a camera orbit around the volume. The code for the benchmark is available on GitHub.¹ An example image rendered by the benchmark is shown in Fig. 6. We record both the total frame time and compositing overhead. Compositing overhead is defined as the additional time spent compositing after the slowest local rendering task has completed [19].

The distributed API version uses OSPRay’s MPI module for distributed rendering, performing compositing with the Distributed FrameBuffer. The IceT version uses OSPRay’s scientific visualization style renderer to render the ranks’ local data, and IceT to composite the partial images. We measure the compositing overhead in OSPRay as the shortest time between local rendering finishing and the frame

¹ <https://github.com/Twinklebear/osp-icet>.

Fig. 6 An image from the scaling benchmark. Each rank generates a 128^3 volume brick colormapped by its rank and collectively render a camera orbit around the data. The bricks are relatively small and mostly opaque to focus the benchmark on the compositing overhead incurred



finishing. For IceT we take the minimum value returned for ICET_COMPOSITE_TIME across all ranks.

We evaluate strong scaling on Stampede2 at the Texas Advanced Computing Center (TACC), on both the Skylake Xeon (SKX) and Xeon Phi Knight's Landing (KNL) nodes (Fig. 7). At lower node counts, the IceT-based renderer outperforms the DFB-based renderer. This difference appears to be attributable to a mix of differences in local rendering performance and compositing overhead. On SKX at low node counts, the total frame time difference is more evenly split between local rendering and compositing, whereas on KNL the difference is primarily attributable to local rendering performance, as the difference in total frame time is much larger than that in compositing overhead.

We observe better scalability when using the distributed API, with performance matching or surpassing the IceT version at higher node counts. This difference is especially stark on the highly parallel KNL architecture, where the IceT version's total time increases beyond 64 nodes on the 2048^2 image benchmark. At 512 SKX nodes, we begin to observe a similar slowdown with IceT on the 2048^2 benchmark. We do observe an odd increase in compositing time with IceT on 128 to 512 KNL nodes in the 2048^2 case, which resolves at 1024 nodes. This increase was reproducible over multiple jobs, and may be attributable to differences in the network topology or job placement when running at these node counts.

Beyond 1024 nodes we still observe increasing total frame time for IceT on the 2048^2 benchmark on KNL, which is attributable to the scivis renderer processing tiles that its data do not project to. In contrast, the distributed API version processes only the tiles that the ranks' data projects to. When using OSPRay's existing local renderers, this optimization is not used, and some threads will be assigned to render

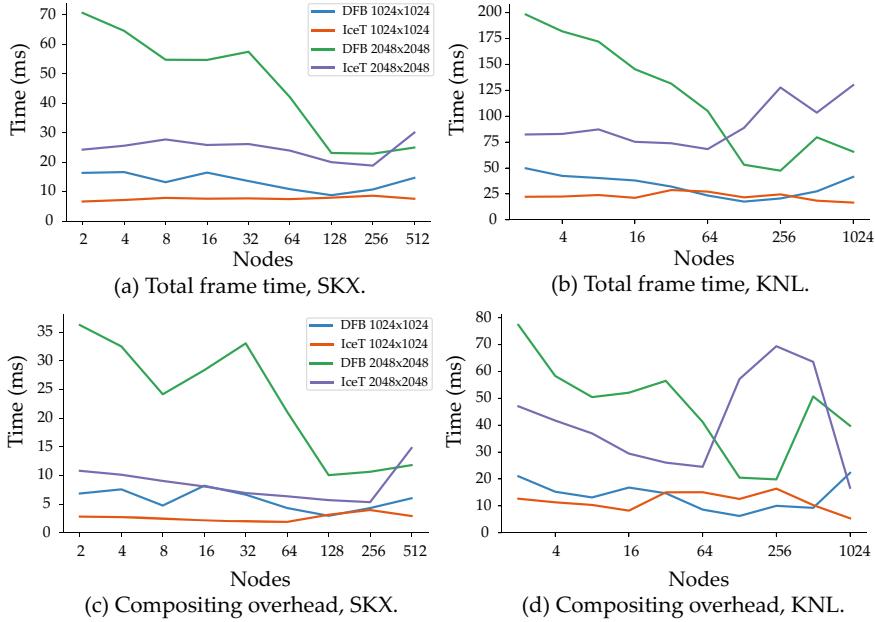


Fig. 7 Strong scaling results on the compositing performance benchmark, comparing a typical IceT + OSPRay renderer versus one using the distributed API. The benchmark renders a 300-frame orbit around the volume. We plot the average frame and compositing overhead time for each run after discarding the first 10 frames as warm-up frames. We find that the distributed API’s local renderer does not perform as well as OSPRay’s scivis renderer; however, the DFB’s compositor scales better at higher node counts, especially on highly parallel architectures

image tiles that the rank’s local data is not visible on, thereby reducing the number of threads that do perform useful rendering work. This effect is less visible on SKX systems where each individual core is relatively powerful; however, as the core count grows or the ratio of pixels touched by local data to those not touched decreases, we would expect to observe a similar effect.

Moreover, the distributed API is able to support more complex data distributions and workloads, providing greater flexibility to in situ visualization systems than traditional compositors such as IceT. For example, in a tightly coupled use case, the regions can be used to create a virtual set of bricks that partition nonconvex or nondisjoint data-distributions into convex, disjoint sets that can be composited, without redistributing the data (Fig. 4). In an in transit use case, the hybrid-data parallel mode can be used to partially replicate data among the visualization ranks to achieve better load balancing, as demonstrated in [44].

5 Example Use Cases

Although OSPRay’s distributed API is relatively new, it has found some early adoption for in situ visualization. Usher et al. [43] used the distributed API to implement an interactive in transit visualization system for LAMMPS simulations, and they reported good weak scaling of the renderer on Theta and Stampede2. The renderer used in their application is available on GitHub.² Demiralp et al. [13] used the distributed API to render streamlines traced by a data-parallel particle advection analysis code for 3D-Polarized Light Imaging data. With the addition of asynchronous rendering and usability improvements made to the distributed API in OSPRay 2.0, we hope to see wider adoption of the distributed API for in situ visualization, and distributed applications in general. Below we present an example application for rendering image databases similar to those used by Cinema, which showcases these improvements.

5.1 *Image Database Generation*

Rendering image databases for applications such as Cinema requires rendering a large number of images, corresponding to a sweep of the visualization parameter space. A large number of images corresponding to multiple viewpoints, different isovalue, transfer functions, etc., will be rendered in situ, producing hundreds to thousands of individual images. The rendering load on each rank will vary based on the visualization parameters, and, for any individual image, a subset of nodes is likely to be underutilized. For example, the rank’s data may not overlap the isovalue or be made fully transparent by the transfer function.

To more fully utilize the nodes, multiple images can be rendered in parallel to each other across the ranks using OSPRay’s asynchronous API. The nodes that are underutilized for one image can compute another image in parallel, without oversubscribing the node. Asynchronous rendering is still collective across the nodes, and thus the number of frames in flight should be chosen to avoid oversubscribing those nodes that are more heavily utilized for ongoing frames. The number of frames in flight is controlled by the application and can be set to a fixed value or dynamically adjusted based on load.

We implement a mini-app to demonstrate the new asynchronous rendering capability’s applicability to image database generation, called mini-cinema³ (Fig. 8). The application supports volume rendering and isosurface rendering, by either explicitly extracting triangles using VTK or using OSPRay’s implicit isosurfaces. The application supports transparent isosurfaces and can compute ambient occlusion on the isosurface by duplicating it across the ranks to provide ghost zones. The mini-app can be configured to render a specific number of frames in flight, to limit memory

² https://github.com/Twinklebear/ospray_senpai.

³ <https://github.com/Twinklebear/mini-cinema>.

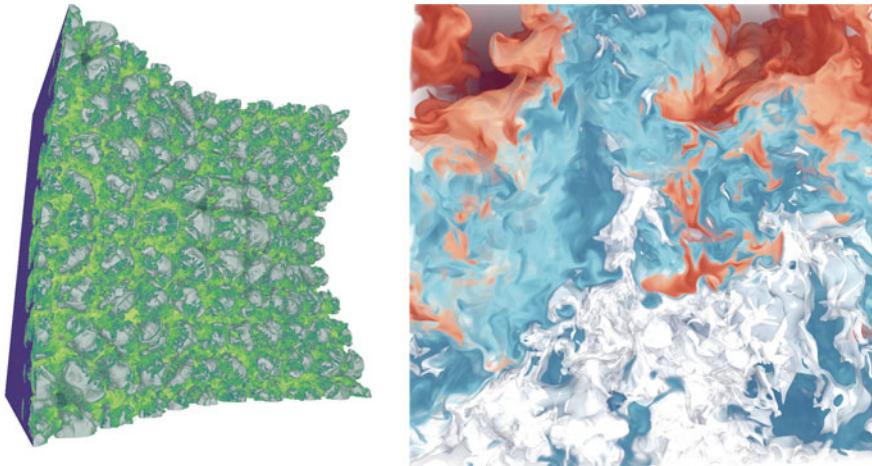


Fig. 8 Example images rendered using the image database generation test app, mini-cinema. The app uses OSPRay’s asynchronous rendering API to render multiple frames in parallel to better utilize the nodes. The Richtmyer-Meshkov [9] (left) uses ghost zones for ambient occlusion on the isosurface, and Miranda [11] (right) combines a semitransparent isosurface with the volume

consumption and avoid oversubscribing those nodes that are more heavily utilized for a frame. After an asynchronous render has completed, the image is written to disk on a background thread managed by TBB.

As the underlying geometry and volume data are kept separate from their appearance information, the data can be shared by multiple ongoing renders. For each transfer function being rendered, only a new `OSPVolumeModel` must be created, which associates a transfer function with existing volume data. Each triangle mesh geometry stores both the triangle data and BVH and is associated with its material by creating an `OSPGeometricModel`. For each unique set of volumetric and geometric models, a new group, instance, and world are created to represent the scene. The world stores the Embree [51] BVH over the instances; however, the bulk of these objects are just references to existing ones, and they incur little memory overhead. Each unique camera position requires a unique `OSPCamera`. The largest amount of additional memory required for each asynchronous render is the framebuffer, which is required to be unique per active frame to avoid write conflicts.

We measure the effect of the number of frames in flight on the total time to task completion in the app using the 1024^3 Miranda data set. We render two isosurfaces, containing 62.47M and 67.65M triangles, combined with the volume from 250 camera positions in an orbit around the volume, for a total of 500 frames. Each image rendered is 1024×1024 . The benchmark is run on 16 Stampede2 SKX nodes, with the number of frames in flight varied from 1 up to 128. We find that moving from 1 to 4 frames in flight yields an improvement of 8.9s in total task time, improving from 23.7s to 14.8s. Increasing the allowed number of frames in flight further does not positively or negatively affect the total task time, likely due to the images completing quickly

enough that not many more than four frames are in flight at a time. We note that the best number of frames in flight is highly dependent on the rendering configuration. More expensive individual frames may result in more in flight at a time, and lead to reduced performance if oversubscribing the nodes.

6 Conclusion

OSPRay’s distributed API provides a compelling option for flexible, scalable, and high-quality in situ visualization. The distributed API is straightforward to learn for those familiar with using OSPRay for single-node rendering, and easily enables the most common data-parallel rendering use cases. Improvements to OSPRay’s API for sharing data enable it to directly share data with a wider range of existing memory layouts, and the new asynchronous rendering API is highly useful for image database generation. To more easily allow moving post hoc rendering applications in situ, existing user geometry and volume modules work seamlessly with the distributed API. To make OSPRay’s distributed rendering more widely accessible to end visualization and simulation users, we are working on integrating it into Catalyst [15]. Integration into LibSim [53] would provide similar benefits to VisIt users.

The flexibility provided by OSPRay’s compositor with regard to data distribution is especially useful to support a wide range of simulations. Applications with more complex data layouts, which standard compositing libraries previously may not have been able to render in place, can be virtually partitioned using OSPRay’s regions to enable rendering them in place. In transit applications can also leverage hybrid-parallel rendering for load balancing to improve performance.

However, OSPRay’s distributed renderer is not without limitations. For a simple data-parallel renderer, we do find poorer rendering performance at lower node counts compared to using OSPRay for local rendering and compositing with IceT. This performance difference is attributable to a combination of local rendering performance and compositing overheads. We find that OSPRay’s distributed renderer performs better than or similar to the IceT version at higher node counts, and is capable of rendering a much wider range of data distributions in place.

In contrast to distributed ray tracers [1, 34, 36], which send rays or move data between ranks to compute global illumination lighting effects, OSPRay’s distributed renderer is restricted to local illumination effects with ghost zones. It should now be possible to integrate such renderers into the distributed API as a module to provide high-fidelity imagery in situ, although moving data in the presence of user modules poses significant interesting challenges.

Acknowledgements The Miranda data set is courtesy Andrew W. Cook, William Cabot, and Paul L. Miller, the Richtmyer-Meshkov is courtesy Ronald H. Cohen, William P. Dannevik, Andris M. Dimitis, Donald E. Eliason, Arthur A. Mirin, and Ye Zhou. Both data sets were made available through the Open Scientific Visualization Datasets repository. This work is supported in part by the

Intel Graphics and Visualization Institute of eXcellence at the Scientific Computing and Imaging Institute, University of Utah. This work is supported in part by NSF: CGV Award: 1314896, NSF:IIP Award: 1602127, NSF:ACI Award: 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC Award: 1842042. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported in this paper.

References

1. Abram, G., Navrátil, P., Grossset, A.V.P., Rogers, D., Ahrens, J.: Galaxy: asynchronous ray tracing for large high-fidelity visualization. In: 2018 IEEE Symposium on Large Data Analysis and Visualization (2018)
2. Aftosmis, M., Berger, M., Adomavicius, G.: A parallel multilevel method for adaptively refined cartesian grids with embedded boundaries. Technical Report AIAA-00-0808, American Institute of Aeronautics and Astronautics (2000). 38th Aerospace Sciences Meeting and Exhibit
3. Ahrens, J., Jourdain, S., O'Leary, P., Patchett, J., Rogers, D.H., Petersen, M.: An image-based approach to extreme scale in situ visualization and analysis. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2014)
4. Berger, M.J., Colella, P.: Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.* (1989)
5. Berger, M.J., Oliger, J.: Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.* (1984)
6. Biedert, T., Werner, K., Hentschel, B., Garth, C.: A Task-Based Parallel Rendering Component For Large-Scale Visualization Applications. In: Eurographics Symposium on Parallel Graphics and Visualization (2017)
7. Bigler, J., Stephens, A., Parker, S.G.: Design for parallel interactive ray tracing systems. In: 2006 IEEE Symposium on Interactive Ray Tracing (2006)
8. Burstedde, C., Wilcox, L.C., Ghattas, O.: P4est: scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.* (2011)
9. Cohen, R.H., Dannevik, W.P., Dimits, A.M., Eliason, D.E., Mirin, A.A., Zhou, Y., Porter, D.H., Woodward, P.R.: Three-dimensional simulation of a Richtmyer-Meshkov instability with a two-scale initial perturbation. *Phys. Fluids* (2002)
10. Colella, P., Graves, D., Ligocki, T., Martin, D., Modiano, D., Serafini, D., Van Straalen, B.: Chombo software package for amr applications design document (2000)
11. Cook, A.W., Cabot, W., Miller, P.L.: The mixing transition in Rayleigh–Taylor instability. *J. Fluid Mech.* (2004)
12. DeMarle, D.E., Gribble, C.P., Boulos, S., Parker, S.G.: Memory sharing for interactive ray tracing on clusters. *Parallel Comput.* (2005)
13. Demiralp, A.C., Zielasko, D., Axer, M., Vierjahn, T., Kuhlen, T.W.: Parallel particle advection and lagrangian analysis for 3D-PLI fiber orientation maps. In: 2019 IEEE 9th Symposium on Large Data Analysis and Visualization (LDAV), Posters (2019)
14. Ellsworth, D., Green, B., Henze, C., Moran, P., Sandstrom, T.: Concurrent visualization in a production supercomputing environment. *IEEE Trans. Vis. Comput. Graph.* (2006)
15. Fabian, N., Moreland, K., Thompson, D., Bauer, A., Marion, P., Geveci, B., Rasquin, M., Jansen, K.E.: The paraview coprocessing library: a scalable, general purpose in situ visualization library. In: 2011 IEEE Symposium on Large Data Analysis and Visualization (2011)
16. Favre, J.M., dos Santos, L.P., Reiners, D.: Direct send compositing for parallel sort-last rendering. In: Eurographics Symposium on Parallel Graphics and Visualization (2007)
17. Fernandes, O., Frey, S., Sadlo, F., Ertl, T.: Space-time volumetric depth images for in-situ visualization. In: 2014 IEEE 4th Symposium On Large Data Analysis and Visualization (LDAV) (2014)

18. Frey, S., Ertl, T.: Load balancing utilizing data redundancy in distributed volume rendering. In: Eurographics Symposium on Parallel Graphics and Visualization (2011)
19. Grossset, A.P., Knoll, A., Hansen, C.: Dynamically scheduled region-based image compositing. In: Eurographics Symposium on Parallel Graphics and Visualization (2016)
20. Grossset, A.V.P., Prasad, M., Christensen, C., Knoll, A., Hansen, C.: TOD-tree: task-overlapped direct send tree image compositing for hybrid MPI parallelism and GPUs. *IEEE Trans. Vis. Comput. Graph.* (2017)
21. Han, M., Wald, I., Usher, W., Wu, Q., Wang, F., Pascucci, V., Hansen, C.D., Johnson, C.R.: Ray tracing generalized tube primitives: method and applications. *Comput. Graph. Forum* (2019). <https://doi.org/10.1111/cgf.13703>
22. Hsu, W.M.: Segmented ray casting for data parallel volume rendering. In: Proceedings of the 1993 Symposium on Parallel Rendering (1993)
23. Ibrahim, S., Stitt, T., Larsen, M., Harrison, C.: 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. Denver Colorado (2019)
24. Intel: OneAPI Rendering Toolkit. <https://software.intel.com/en-us/rendering-framework>
25. Intel: Open Image Denoise. <https://www.openimagedenoise.org>
26. Intel: Open Volume Kernel Library. <https://www.openvkl.org>
27. Ize, T., Brownlee, C., Hansen, C.D.: Real-time ray tracer for visualizing massive models on a cluster. In: Eurographics Symposium on Parallel Graphics and Visualization (2011)
28. Kageyama, A., Yamada, T.: An approach to exascale visualization: interactive viewing of in-situ visualization. *Comput. Phys. Commun.* (2014)
29. Karlsson, J., Abdellah, M., Speierer, S., Foni, A., Lapere, S., Schürmann, F.: High fidelity visualization of large scale digitally reconstructed brain circuitry with signed distance functions. In: 2019 IEEE Visualization Conference (VIS) (2019)
30. Kendall, W., Peterka, T., Huang, J., Shen, H.W., Ross, R.B.: Accelerating and benchmarking radix-k image compositing at large scale. In: Eurographics Symposium on Parallel Graphics and Visualization (2010)
31. Ma, K.L., Painter, J.S., Hansen, C.D., Krogh, M.F.: Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl.* (1994)
32. MacNeice, P., Olson, K.M., Mobarry, C., de Fainchtein, R., Packer, C.: PARAMESH: a parallel adaptive mesh refinement community toolkit. *Comput. Phys. Commun.* (2000)
33. Moreland, K., Kendall, W., Peterka, T., Huang, J.: An image compositing solution at scale. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (2011)
34. Navrátil, P.A., Fussell, D., Lin, C., Childs, H.: Dynamic scheduling for large-scale distributed-memory ray tracing. In: Eurographics Symposium on Parallel Graphics and Visualization (2012)
35. O'shea, B.W., Bryan, G., Bordner, J., Norman, M.L., Abel, T., Harkness, R., Krtsuk, A.: Introducing Enzo, an AMR cosmology application. In: Adaptive Mesh Refinement-Theory and Applications, Lecture Notes in Computational Science and Engineering. Springer (2005)
36. Park, H., Fussell, D., Navrátil, P.: SpRay: speculative ray scheduling for large data visualization. In: 2018 IEEE Symposium on Large Data Analysis and Visualization (2018)
37. Peterka, T., Goodell, D., Ross, R., Shen, H.W., Thakur, R.: A configurable algorithm for parallel image-compositing applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (2009)
38. Pharr, M., Mark, W.R.: ispc: A SPMD compiler for high-performance CPU programming. In: Innovative Parallel Computing (InPar) (2012)
39. Reinhard, E., Chalmers, A., Jansen, F.W.: Hybrid scheduling for parallel rendering using coherent ray tasks. In: Proceedings of the 1999 IEEE Symposium on Parallel Visualization and Graphics (1999)
40. Rizzi, S., Hereld, M., Insley, J., Papka, M.E., Uram, T., Vishwanath, V.: Large-scale co-visualization for lammps using Vi3. In: 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV) (2015)

41. Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K.L., O'hallaron, D.R.: From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (2006)
42. Turuncoglu, U.U., Önal, B., Ilicak, M.: A new approach for in situ analysis in fully coupled earth system models. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV '19. Denver, Colorado (2019)
43. Usher, W., Rizzi, S., Wald, I., Amstutz, J., Insley, J., Vishwanath, V., Ferrier, N., Papka, M.E., Pascucci, V.: libIS: A lightweight library for flexible in transit visualization. In: ISAV: In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (2018)
44. Usher, W., Wald, I., Amstutz, J., Günther, J., Brownlee, C., Pascucci, V.: Scalable ray tracing using the distributed framebuffer. Comput. Graph. Forum (2019)
45. Usher, W., Wald, I., Knoll, A., Papka, M.E., Pascucci, V.: In situ exploration of particle simulations with CPU ray tracing. Supercomput. Front. Innov. (2016)
46. Vierjahn, T., Schnorr, A., Weyers, B., Denker, D., Wald, I., Garth, C., Kuhlen, T.W., Hentschel, B.: Interactive exploration of dissipation element geometry. In: Eurographics Symposium on Parallel Graphics and Visualization (2017)
47. Wald, I., Benthin, C., Slusallek, P.: A flexible and scalable rendering engine for interactive 3D graphics. Saarland University, Technical report (2002)
48. Wald, I., Brownlee, C., Usher, W., Knoll, A.: CPU volume rendering of adaptive mesh refinement data. In: SIGGRAPH Asia 2017 Symposium on Visualization (2017)
49. Wald, I., Johnson, G.P., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navrátil, P.: OSPRay—a CPU ray tracing framework for scientific visualization. IEEE Trans. Vis. Comput. Graph. (2017)
50. Wald, I., Knoll, A., Johnson, G.P., Usher, W., Pascucci, V., Papka, M.E.: CPU ray tracing large particle data with balanced P-k-d trees. In: 2015 IEEE Scientific Visualization Conference (SciVis), pp. 57–64 (2015)
51. Wald, I., Woop, S., Benthin, C., Johnson, G.S., Ernst, M.: Embree: A kernel framework for efficient CPU ray tracing. ACM Trans. Graph. (2014)
52. Wang, K.C., Shareef, N., Shen, H.W.: Image and distribution based volume rendering for large data sets. In: 2018 IEEE Pacific Visualization Symposium (PacificVis) (2018)
53. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Eurographics Symposium on Parallel Graphics and Visualization (2011)
54. Wu, Q., Usher, W., Petruzza, S., Kumar, S., Wang, F., Wald, I., Pascucci, V., Hansen, C.D.: VisIt-OSPRay: toward an exascale volume visualization system. In: Eurographics Symposium on Parallel Graphics and Visualization (2018)
55. Yu, H., Wang, C., Grout, R.W., Chen, J.H., Ma, K.L.: In situ visualization for large-scale combustion simulations. IEEE Comput. Graph. Appl. (2010)
56. Yu, H., Wang, C., Ma, K.L.: Massively parallel volume rendering using 2–3 swap image compositing. In: SC-International Conference for High Performance Computing, Networking, Storage and Analysis (2008)
57. Yucong, Y., Miller, R., Ma, K.L.: In situ pathtube visualization with explorable images. In: Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization (2013)

Multivariate Functional Approximation of Scientific Data



Tom Peterka, Youssef Nashed, Iulian Grindeanu, Vijay Mahadevan, Raine Yeh, and David Lenz

Abstract While our computational capability to generate raw data grows, the ability to store, transform, and draw conclusions from scientific data is lagging. Beginning with the introductory chapter and continuing throughout much of this book, we see numerous examples of how *in situ* processing can help close the gap between data generation and data analysis. This chapter expands the discussion of *in situ* methods beyond when and where data are processed, to how data are represented. Rethinking the way that scientific data are represented can empower subsequent visualization and analysis, especially when such data transformations are performed *in situ*. Scientific data may be transformed by recasting to a data model fundamentally different from the discrete pointwise or element-wise datasets produced by computational models. In **Multivariate Functional Approximation**, or **MFA**, scientific datasets are redefined in a hypervolume of piecewise-continuous basis functions. Compared with existing discrete models, the continuous functional model can save space while affording many of the same spatiotemporal analyses without reverting back to the discrete form. In this chapter, modeling the MFA, *in situ*, is presented. The data model and modeling approach are parallelized for high-performance computing. A lightweight and efficient method of enforcing high-degree continuity across subdomains in the parallel decomposition is also included. The MFA can subsequently be used post hoc to evaluate points and derivatives anywhere in the domain, facilitating numerous analysis and visualization applications.

1 Introduction

Three observations motivate the approach described in this chapter. The first two reasons are common to other *in situ* methods. The growing disparity between computational and I/O rates with each new generation of supercomputer requires more

T. Peterka (✉) · Y. Nashed · I. Grindeanu · V. Mahadevan · D. Lenz

Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
e-mail: tpeterka@mcs.anl.gov

R. Yeh
Purdue University, West Lafayette, IN, USA

data analysis and visualization to be conducted at the source of the data. For example, the rate of data that can be computed on the Summit supercomputer at the Oak Ridge Leadership Computing Facility (assuming 1 byte generated per clock cycle) is five orders of magnitude greater than the bandwidth of its parallel file system [40]. Beyond performance, *in situ* analyses can also improve the quality of science. While the fidelity of post hoc analysis is hamstrung by the low temporal frequency of writing data to persistent storage, *in situ* analysis can have much higher fidelity because analysis tasks have access to all the data directly, not only the tiny fraction saved for postprocessing.

The third motivation, specifically in this chapter, addresses the type of processing to conduct *in situ*. In addition to reducing data size, the objective is to transform the data model into a more useful form for analysis. Both *in situ* and post hoc, many operations involving managing, analyzing, and visualizing scientific data can be streamlined when data are in a continuous functional form, rather than a discrete set of points. Converting discrete data into piecewise-continuous basis functions multiplied by a sparse set of control points results in a hypervolume of NURBS or B-splines. This transformation is called **Multivariate Functional Approximation**, or **MFA**. The objective of this chapter is to represent scientific data *in situ*, in an MFA model, and to use that model instead of the original discrete dataset for post hoc analysis and visualization.

The MFA model can represent numerous types of data because it is agnostic to the mesh, field, or discretization of the input dataset. Compared with existing discrete data models, the MFA model can enable many spatiotemporal analyses, without converting the entire dataset back to the original discrete form. The MFA often occupies less storage space than the original discrete data do, providing some data reduction, depending on data complexity and intended usage. For example, noise may be intentionally smoothed by using a small number of control points and high-degree basis functions; alternatively, high-frequency data features may be preserved with more control points and lower degree. Post hoc, the MFA enables analytical closed-form evaluation of points and derivatives, to high order, anywhere inside the domain, without being limited to the locations of the input data points.

A key decision in functional data analysis is the choice of basis function family. Fourier [6], wavelet [22, 25], and geometric (i.e., spline) [13] bases are commonly used; recently Austin et al. [1] proposed the Tucker decomposition as a low-rank alternative. Majdisova and Skala proposed radial basis functions for particle data [31]. The MFA uses geometric (specifically, nonuniform rational B-spline) bases because they mirror the space-time properties in the original data and retain these geometric properties in analytics and visualization. Computing geometric bases requires minimal memory overhead, and the parallel decomposition of the original space-time domain is identical in the geometric functional domain, an important consideration for minimizing data movement at scale.

NURBS (nonuniform rational B-splines) are piecewise continuous, differentiable, have local support, and are invariant to affine transformations. In order to compute the MFA in d dimensions, a d -dimensional tensor product of NURBS bases is computed. Both field geometry (space-time vertex positions) and science variables (pressure,

Fig. 1 One 2-d slice of an S3D dataset partitioned into 20 blocks and modeled in parallel, blocks colored by process ID



density, temperature, etc.) are modeled. The model is efficiently represented by a set of control points and knots. The control points are reference points that “push and pull” the representation through their linear combination with basis functions. The knots map partitions of the data to the control points and associated basis functions.

The MFA is designed to model scientific data *in situ*, in parallel, on high-performance computing (HPC) platforms. The model is adaptively refined until a user-set error threshold is achieved, and it includes a lightweight and efficient method of enforcing high-degree continuity across subdomains in the parallel decomposition, to the same degree of continuity as the blocks’ interior MFA models (Fig. 1).

What makes the MFA unique is that the transformed data retain geometric properties: spatial and temporal contiguity, derivatives, and statistical distributions are all preserved. The goal of analyzing scientific data—which are inherently spatiotemporal—is usually to understand the relationship of science variables to their position in space and time. Thus, once data are modeled in the MFA, many data analyses and visualizations are possible directly from it.

Two fundamental operations are required in an MFA: building a model from an input discrete dataset, and evaluating points and derivatives from the model. The former is usually performed *in situ*; the latter is often done post hoc. Both stages are covered in this chapter. Following a discussion of related work in the next section, *in situ* modeling is presented in Sect. 3, and post hoc uses of the MFA are explained in Sect. 4. Parallel implementations of both modeling and evaluation are covered in Sect. 5. The chapter concludes with a look at some scientific use cases and a glimpse at the future of functional approximation for scientific data.

2 Related Work

Multivariate functional approximation borrows ideas from several fields: compression, statistics, modeling, visualization, and analysis. Piecewise functional approx-

imations replace discrete data points with linear combinations of basis functions and a small number of reference points called *control points*. Statisticians call this method *functional data analysis* [17, 46], with low-dimensional serial implementations available in popular statistics packages [45].

NURBS are used extensively within computer-aided design (CAD) software tools [48]. Fitting NURBS to existing data is not straightforward, however, because the inclusion of rational weights results in a nonlinear problem. Hence, most NURBS implementations resort to using uniform weights (all set to 1.0), which are then manually tweaked by users for additional model shape control. This is the approach taken in the MFA as well. Rational weights are implemented and supported, but for performance reasons the weights are currently set to 1.0 (i.e., reverting to B-splines).

Previous research demonstrated that B-splines can faithfully represent scientific data in up to three dimensions: 2-d triangular surfaces and 3-d tetrahedral meshes can be converted into bivariate and trivariate models. Martin and Cohen first developed the data model and framework in 2001 [34], and Martin et al. described how to parameterize triangular and tetrahedral data as tensor products in 2008 [33] for modeling exterior surfaces and interior volumes of medical datasets.

Geometric functional representations can replace data models used today in visualization algorithms. Martin and Cohen derived isosurfacing, slicing, and ray tracing algorithms directly from B-spline models [34]. Raviv and Elber [47] showed direct rendering from trivariate B-splines for isosurface extraction, planar slicing, and volume rendering. Park and Lee [39] visualized trivariate NURBS for flow data. Hua et al. [23] modeled 3-d solids using simplex splines and showed that visualization algorithms such as isosurfacing and volume rendering can operate on the same data model.

Functional data models are used in mechanical and fluid engineering simulations. One example is isogeometric analysis (IGA) [24], which uses NURBS models for mechanical simulations. Recently, a parallel IGA toolkit [11] built on PETSc [4] was developed for solving high-order partial differential equations over NURBS basis domains. Spectral methods are another example of a discretization that uses basis functions to evaluate data: the Nek5000 Navier–Stokes solver [14] is based on a weighted sum of basis functions defined over a coarse set of control points.

Such engineering simulations usually take as input a mesh that was created in a CAD system, and NURBS modeling is a rich and mature topic in CAD literature. Lin et al. [28] summarized numerous approaches to reverse engineering NURBS surfaces given a set of 3-d input points. Partition of unity methods [3, 7] allow local approximation to extend to the entire domain by blending basis functions of local approximations into a global solution with the desired continuity across boundaries of local regions.

The local regions of a global model can be organized in various ways. Sederberg et al. [5, 49] introduced T-splines to conserve space and to ensure continuity along B-spline patches. The idea is to localize the addition of knots and control points in refined regions, rather than extending the refinement in all dimensions over the entire domain, as in a traditional tensor product representation. T-splines, however,

increase the bookkeeping effort to keep track of the irregular T-shaped connectivity of refined regions.

3 In Situ Modeling of the MFA

We begin by introducing the minimum background, consisting of common notation and a few equations, needed to understand the rest of the chapter. Afterwards, we describe the method to build an MFA model approximating an input dataset to a desired tolerance; first with a fixed number of control points and then with an adaptive scheme where the resolution of the model varies with complexity of the dataset.

3.1 Mathematical Background

A curve (Fig. 2 left) can be parameterized as a vector-valued function of a single parameter u such that

$$\mathbf{C}(u) = \sum_{i=0}^{n-1} N_{i,p}(u) \mathbf{P}_i. \quad (1)$$

$N_{i,p}$ are the p th degree basis functions, and \mathbf{P} is the set of n control points; n , the number of output control points, is usually less than m , the number of input data points. The definition extends to higher manifold dimensions (surfaces, volumes, hypervolumes) as a tensor product of multiple parameters u_1, u_2, \dots, u_d . The right panel of Fig. 2 shows a surface, and in general a d -dimensional hypervolume is parameterized as follows:

$$\mathbf{V}(u_1, \dots, u_d) = \sum_{i_1} \cdots \sum_{i_d} N_{i_1,p}(u_1) \times \cdots \times N_{i_d,p}(u_d) \mathbf{P}_{i_1, \dots, i_d}. \quad (2)$$

The basis functions are computed by using the recurrence formula of Cox [10] and De Boor [12]. The recursive computation of $N_{i,p}(u)$ requires computing $O(p^2)$ nonzero coefficients. The degree p is a small number, usually between 1 and 10.

The n control points \mathbf{P} are found by solving a linear least-squares optimization problem $(N^T N) \mathbf{P} = \mathbf{R}$, where \mathbf{R} is computed from the m input data points and basis functions in $O(m + n)$ time [27]. The matrix of basis functions $N^T N$ (in normal form) is n^2 in size, positive definite, and sparse with $2p + 1$ nonzero entries along the diagonal [13]. The vector of control points can be solved without pivoting in $O(n^3)$ time.

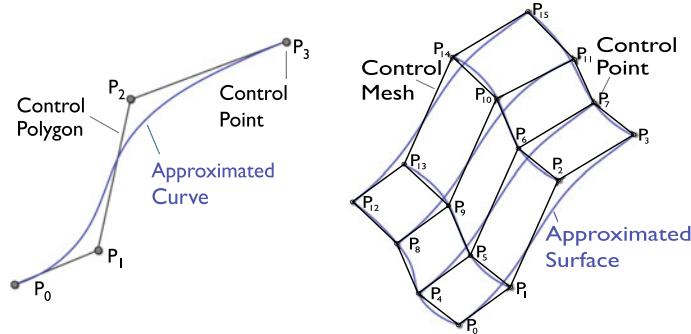


Fig. 2 Left: a B-spline curve. Right: a tensor product of B-spline curves in a surface. Tensor products extend to higher-dimensional hypervolumes

3.2 Modeling with Fixed Size and Separable Dimensions

In this section, we describe the basic steps to model the MFA: specifically, how to model the dimensions separately in a multidimensional domain, thereby reducing the computational complexity and memory footprint. In the separable-dimension approach, the memory to compute the matrix of basis functions is allocated only once for each new dimension and is reused; hence, memory size does not grow with the number of dimensions.

Understanding the steps needed to build an MFA model that approximates a set of m input points is easiest if one begins with a fixed number of control points, n . In addition to n , the user specifies the polynomial degree, p , for the model. The first step is to compute parameter values in the range [0.0, 1.0] for all the input data points. In the current MFA implementation, parameters are assigned to input points according to the spacing of input points in each of the principal domain dimensions, specifically by translating and scaling each input point by the extents of the domain in each dimension. Other parameterizations based on chord length between points, and other metrics, are possible. In general, however, the parameterization problem is ill-posed for arbitrary input points. Piegl and Tiller [44] provided a comprehensive overview of parameterization approaches.

Knots are the breakpoints in the parameter space between different basis functions. The number of knots per dimension is $n + p + 1$ by definition. Absent any local adaptive refinement (the subject of the next section), the default knot placement for the internal knots is to follow the distribution of the parameters of the input points. Both the parameters and the knots are represented by one vector per dimension, with the total number being the sum over the dimensions rather than the product.

Instead of fitting all the dimensions simultaneously, it is less expensive to decompose the dimensions and fit each dimension separately. In the work of Peterka et al. [41], the approach of Piegl and Tiller [43] was extended to any number of dimensions, as illustrated in Fig. 3 for 3-d input points. Assume that the number of input

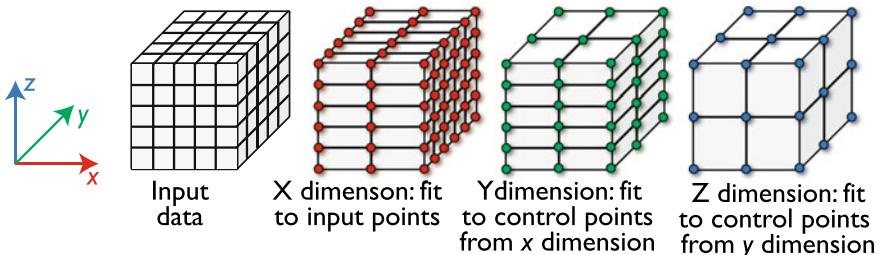


Fig. 3 Iterative fitting over separable dimensions reduces the complexity with each dimension because the control points from the previous dimension are the input points to the next dimension

points is (m_x, m_y, m_z) in the x , y , z dimensions and that the number of control points is (n_x, n_y, n_z) . The original data are approximated as a set of 1-d curves in the first dimension, for example, the x direction. The number of curves is the product of the number of points in the remaining dimensions, $m_y \times m_z$. The linear least-squares system described in the preceding section is solved for each curve in $O(n_x^3)$ time.

Before solving in the second dimension, the n_x control points for each solved curve in the first dimension replace the input data points. The input points now consist of fewer points (assuming fewer control points than input points), specifically $n_x \times m_y \times m_z$. We now take curves in the y dimension, and to each curve we fit a set of control points by solving the same linear system, of n_y control points in $O(n_y^3)$ time, for each curve. There are $n_x \times m_z$ number of those curves.

Once again, the resulting control points replace the input data points, becoming the new input for the third dimension. The new input data points now consist of $n_x \times n_y \times m_z$ points. Taking curves in the z direction and fitting n_z control points for each of the $n_x \times n_y$ curves, the resulting control points are the final solution.

The method extends to any number of dimensions, and the result is the same as if a full-dimensional system were solved in one step. However, the cost of performing separate steps, solving for 1-d curves in each dimension, is lower. Assuming that the number of domain dimensions is d and that the number of control points is n in each dimension, solving the full-dimensional least-squares problem in a single step would take $O(n^{3d})$ time complexity and $O(n^{2d})$ space complexity. By comparison, the separable method takes $O(n^{d+2})$ time complexity and $O(n^2)$ space complexity, producing identical results in shorter time and less space. The savings of the separable method over the full-dimensional method improve as the dimensionality grows.

3.3 Local Adaptive Refinement

A model built on a fixed number of control points and predetermined knot spacing, without taking features in the data into account, cannot represent complex input signals accurately or efficiently. An alternative is to begin with some initial set of

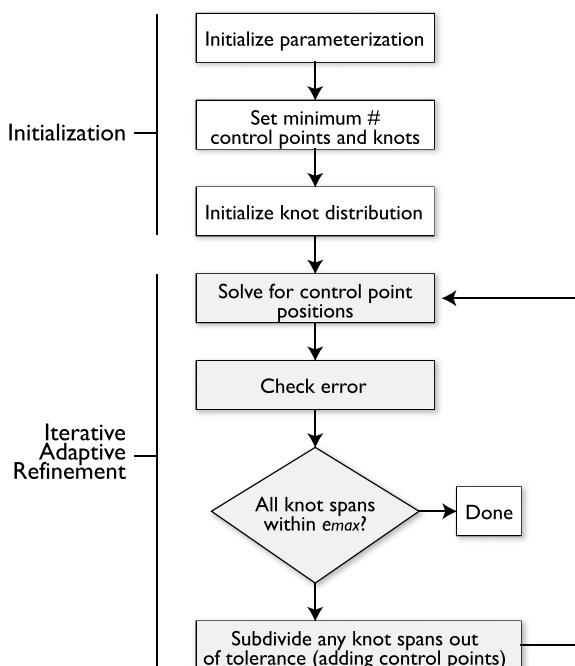
control points and knots, as described above, and then adaptively add more detail as needed. The user provides the allowable error, e_{max} , for example, the maximum relative error normalized by the extent of the data values. Other error metrics such as the L2-norm or root mean squared (RMS) error are possible with no loss of generality.

Figure 4 illustrates the steps to adaptively fit the MFA, refining regions of high error until the error everywhere is below a user-prescribed threshold e_{max} . Beginning with an initial knot distribution for the minimum number of control points, control points and knots are adaptively added until all the evaluated points in each span of knots are within e_{max} of the original points. Knot spans with error greater than e_{max} are subdivided, and the MFA is recomputed.

Recomputing all the control points anew during every iteration is inefficient because a global solution and evaluation of the error at every input point are required during every adaptive iteration. Because the global solution procedure ignores the results of the previous iteration, the global procedure solves the entire domain of input points from scratch each time. To address these shortcomings, Nashed et al. [37] developed a local adaptive algorithm that is able to incrementally refine an existing model without recomputing the MFA over the entire input domain or compromising the continuity of the model.

This local incremental method takes advantage of B-spline local support to refine regions of the approximated model, acting locally on both input and model subdomains, without affecting other regions of the global approximation. The method is

Fig. 4 Overall adaptive refinement algorithm



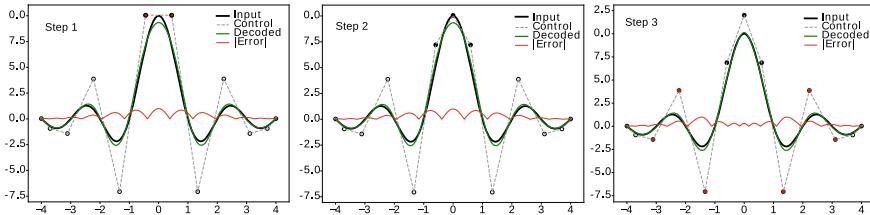


Fig. 5 Steps to locally refine a region while maintaining high-order continuity at the junction between the original region (at the edges of each diagram) and the new region (in the center of each diagram). This example is for $p = 3$ in 1-d. The net effect of adding one new control point is accomplished by (left image) removing $p - 1$ control points, (center image) adding p control points, and (right image) solving for the new p control points constrained by existing p control points on either side

diagrammed in Fig. 5 for $p = 3$ and proceeds as follows. Insert a knot where the error is highest, by removing $p - 1$ control points (red circles, left panel of Fig. 5) and adding p new control points (black circles, center panel), without changing the decoded curve/surface. Perform a local fitting (right panel of Fig. 5) for the new control points (black circles) with equality constraints for p boundary control points (red circles). Repeat for the next highest error location.

This approach reduces the computational burden by restricting the iterative optimization locally in subdomains of both the approximation and the input domains, and it naturally lends the algorithm to a parallel implementation. In contrast, the global refinement method performs global modeling and evaluation operations in each iteration, which eventually become more expensive than those in the local refinement method. Figure 6 shows that the local refinement algorithm results in higher compression (left panel) and lower time to solution (right panel) once the input size is large enough, compared with the global algorithm.

The local adaptive refinement method is directly generalizable to high dimensions and is able to model scientific data, as shown in Fig. 7. The adaptive spacing of control points that results is clearly visible, as well as the relationship between the control point spacing in the upper portion of the figure and the varying complexity of the data in the lower portion.

The adaptive method and the resulting accuracy of the model are driven by a user-supplied error bound. At present, we apply this error bound only to values, not to derivatives, when fitting the model. However, one could modify the adaptive algorithm described above to control the error of derivatives or some combination of values, derivatives, and other application-specific constraints, if desired.

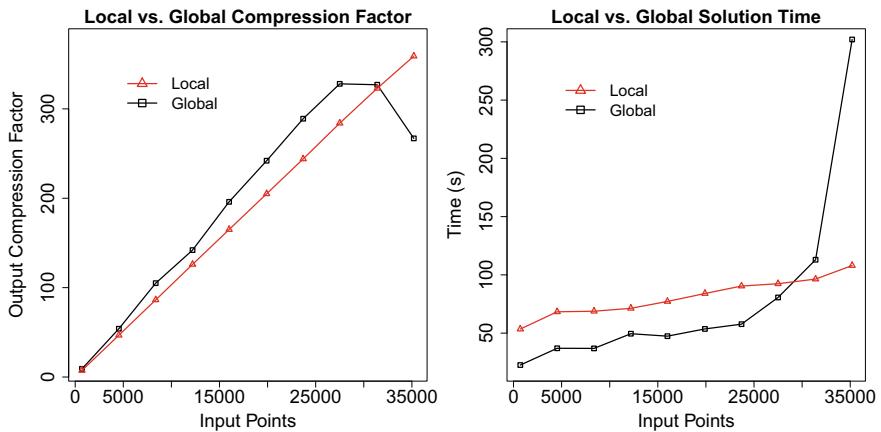


Fig. 6 Comparison of local refinement with global refinement in terms of resulting compression factor on the left and solution time on the right

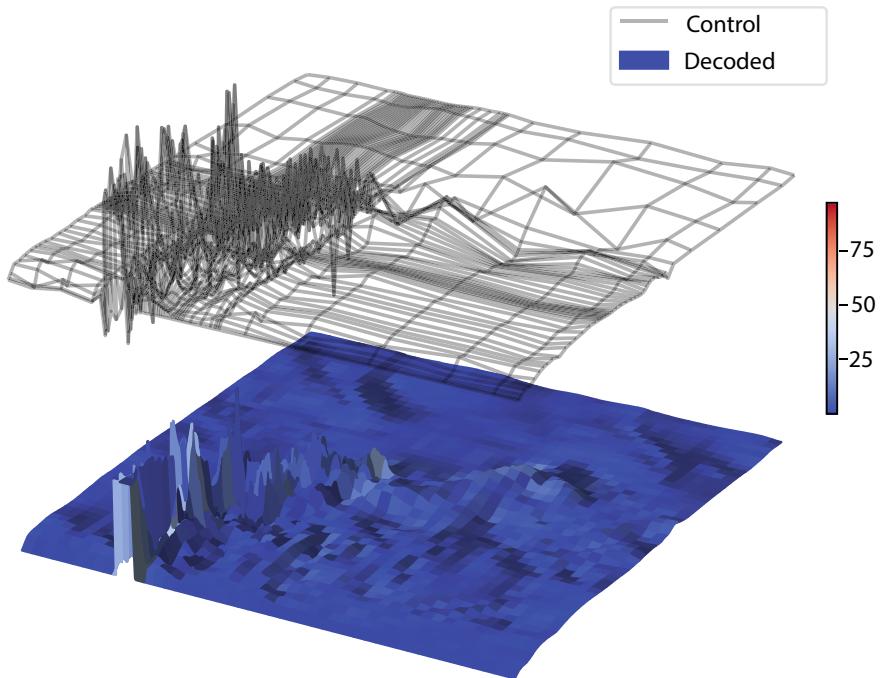


Fig. 7 A 2-d slice of S3D [8] turbulent combustion data [20] modeled by the MFA. Below, the surface evaluated from the model. Above, the mesh of control points

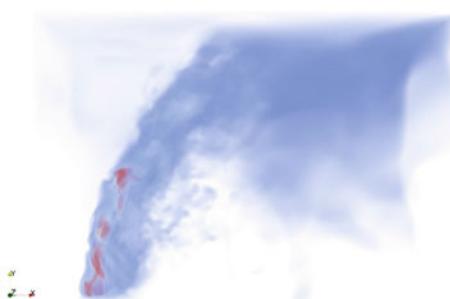
3.4 Modeling Scientific Data

An assortment of scientific use cases demonstrates how the MFA can be used to fit actual scientific datasets consisting of turbulence, shocks, and high-frequency variation. Exemplar applications include combustion, computational fluid dynamics, turbulent mixing, and global climate. Data relating the size of the MFA and error of the approximation, along with an image of reconstructed points evaluated from the MFA model, are presented for each dataset. The relative RMS error reported is the root mean squared error normalized by the extent of the data range.

Turbulent combustion. The turbulent combustion dataset is generated by an S3D simulation [8] of fuel jet combustion in the presence of an external crossflow [18, 20]. The domain is 3-d (x, y, z) ($704 \times 540 \times 550$), and the range variable $f(x, y, z)$ is the magnitude of the 3-d velocity. Figure 8 shows a volume rendering of the resulting evaluated points from an MFA modeled with $p = 3$.

Thermal hydraulics. A 3-d vector field representing the numerical results of a large-eddy simulation of Navier–Stokes equations for the MAX experiment [35] is representative of turbulent mixing and thermal striping that occur in the upper plenum of liquid sodium fast reactors. The data, generated by the Nek5000 solver, have been resampled from their original topology onto a $200 \times 200 \times 200$ regular grid, and the magnitude of the velocity vector is associated with each 3-d domain point. Figure 9 shows a volume rendering of reconstructed data when the MFA is modeled with $p = 3$.

Rayleigh-Taylor instability. Rayleigh-Taylor instability [30] occurs at the interface between a heavy fluid overlying a light fluid, under a constant acceleration, and is of fundamental importance in a multitude of applications ranging from astrophysics to ocean and atmosphere dynamics. Small perturbations at the interface between the two fluids interact nonlinearly and eventually become turbulent. The dataset was generated by the CFDNS [29] Navier–Stokes solver. One time step of $288 \times 512 \times 512$ velocity vector magnitude data was modeled with $p = 3$, and Fig. 10 shows a volume rendering of the reconstructed data from the MFA.



Input Data Points	Output Control Points	Relative RMS Error
2.1×10^8	1.4×10^7	4.6×10^{-4}

Fig. 8 S3D dataset showing velocity magnitude evaluated from the MFA model

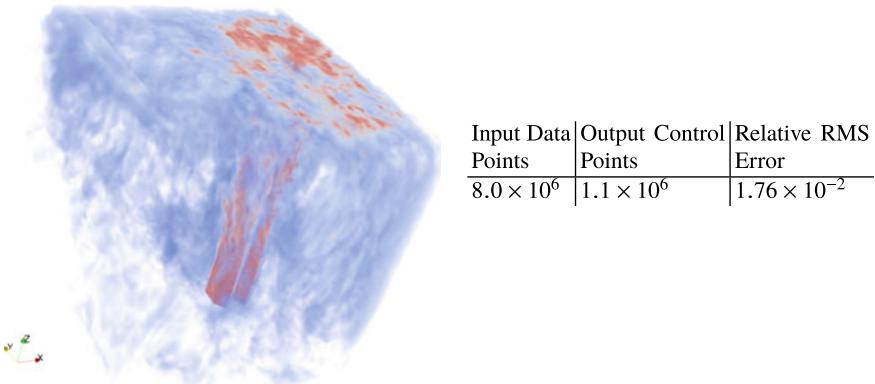


Fig. 9 Volume rendering of data evaluated from the MFA of the Nek5000 dataset

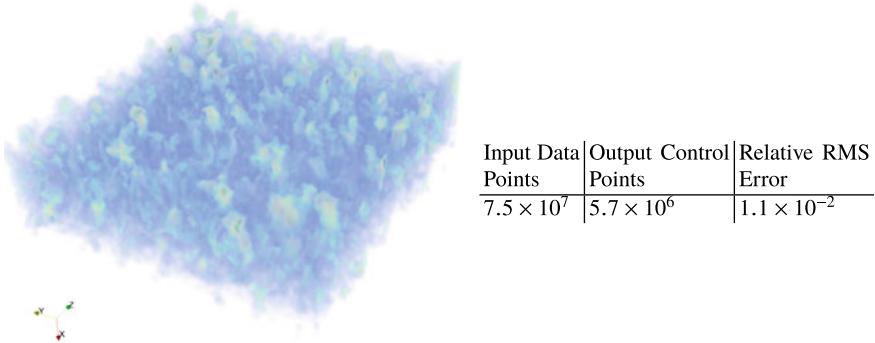


Fig. 10 Volume rendering of data evaluated from the MFA of the CFDNS dataset

Atmospheric climate. The Community Earth System Model (CESM) is a global climate model providing data of the Earth's oceans, atmosphere, land, and sea ice. The dataset in this example is the FLDSC (clear-sky downwelling longwave flux at surface) variable of the Community Atmosphere Model developed at the National Center for Atmospheric Research [38], and the data lie on an 1800×3600 2-d domain with one value of FLDSC at each grid point. Figure 11 shows the evaluated points when modeled with $p = 3$.

The MFA, in contrast to floating-point lossy compression methods, is a transformed data model that retains geometric properties and enables analytical operations directly. Rather than reducing the number of bits used to encode the input data points, the adaptive MFA fitting algorithm automatically selects the number and location of control points. The resulting accuracy and size of the model depend on the complexity of the data, as the previous examples show. The MFA and lossy compression can be seen as complementary methods with different objectives. The two methods can

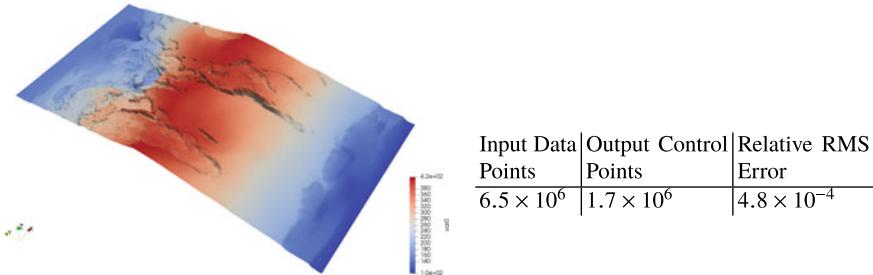


Fig. 11 Surface rendering of CESM data evaluated from the MFA model

be combined; for example, compression can further reduce the storage size of MFA control points and knots. In other instances, one or the other method may be more appropriate depending on the purpose of the in situ operation and its post hoc uses.

4 Post Hoc Use of the MFA

The MFA is a surrogate data model that is solved once (typically in situ) and then used many times for data analysis and visualization post hoc, facilitating applications beyond simply reproducing the input data points. The MFA is designed to be used directly for subsequent data analysis and visualization, distinguishing it from other basis representations such as wavelets, cosine transformations, and compression algorithms that require the inverse transform to be applied first. Several operations are possible directly from the MFA without reverting to the original discrete data model. Moreover, they are possible in the full order and accuracy of the model, without linear interpolation or finite difference estimation. While the model itself is an approximation to the original data, subsequent applications of the MFA are exact, analytical, high order, and closed form. No additional error is introduced when evaluating or differentiating points from a model that has previously been approximated.

Once the model is computed in situ, one can evaluate the model at any point in the domain, not just at the original input points, and differentiate the model up to $p - 1$ times in any combination of partial derivatives in the domain dimensions. Because the model is intrinsically high-order, all point evaluations are done at the same high order or degree to which the model is fitted. Affine transformations can be applied directly to the control points. Statistical and machine learning algorithms using the MFA, such as clustering, topological analysis, and principal components analysis, are areas of further research. The following sections explain point evaluation and differentiation—basic building blocks for post hoc usage of the MFA—followed by applications of these building blocks.

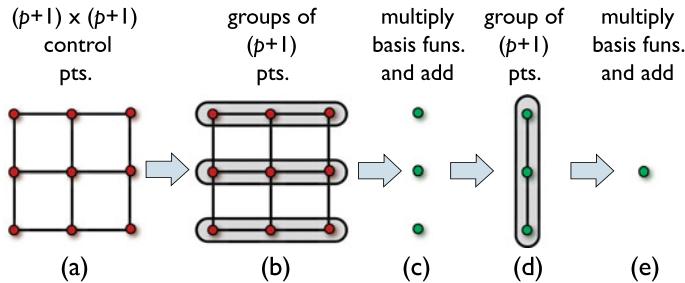


Fig. 12 This example shows the steps to evaluate a point from the MFA with degree $p = 2$ in 2-d, but the same idea extends to any dimension and degree

4.1 Multidimensional Point Evaluation

Points in any dimensionality are evaluated by computing Eq. 2. Figure 12 shows an example of evaluating a 2-d point with $p = 2$ in both dimensions. The number of basis functions and control points multiplied in each dimension is $p + 1$; hence, 9 control points are shown in image (a). First, each curve in the first (horizontal) dimension is collapsed into a single point by the matrix multiplication of basis functions and control points. This step is shown in images (b) and (c). Next, the three resulting points form a curve in the vertical direction, which is collapsed in (d) through multiplication of basis functions to become the resulting evaluated point in (e).

This algorithm, which extends naturally to any number of dimensions, has two main steps: computing the basis functions and multiplying basis functions by control points. The basis functions are computed in $O(p^2)$ time, and the control points are multiplied by basis functions in $O((p + 1)^d)$ time, in d dimensions.

4.2 High-Order Differentiation

To compute derivatives from a previously modeled MFA, one needs only to differentiate the basis functions, replacing $N_{i,p}$ with $N'_{i,p}$ in Eq. 2. Because the basis functions are p -degree polynomials, they are differentiable p times except at knot locations, where they are differentiable $p - 1$ times. The derivatives of a p -degree basis function are computed with a similar recurrence formula as the basis function values, in $O(p^2)$ time. In fact, the original basis functions for the values can be thought of as the zero-degree derivatives. For example, the basis functions for $p = 3$ polynomials and their first two derivatives are shown in Fig. 13.

Derivatives of the MFA use the same knots and control points as the original MFA does, allowing the MFA to be computed once in situ and stored in a file. Basis functions and their derivatives are not stored; instead, they are recomputed as needed. A single model, consisting of knots and control points, allows all the values and all

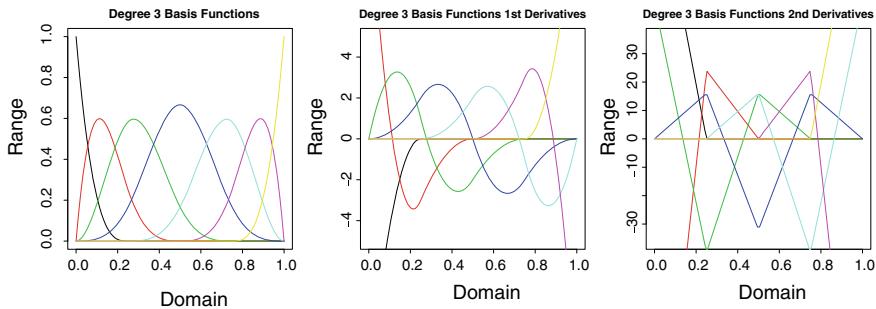


Fig. 13 Basis functions for $p = 3$ values and derivatives

derivatives (total, partial, and mixed) to be available post hoc. The only requirement is to model the original MFA with high enough degree p that any derivatives needed later are less than or equal to $p - 1$ in degree.

4.3 Applications

High-order point evaluation and differentiation enable a multitude of visualization and analysis operations. Some applications are possible directly from the control points and knots of the MFA, while others take advantage of being able to evaluate and differentiate points (sample the data) at arbitrary positions, in closed form. The MFA can be used to visualize high-order data, to compute Jacobian and Hessian fields, as a surrogate representation in machine learning algorithms, to compare and remap data points from one discretization to another, and to smooth data and filter noise.

High-order visualization. High-order data such as those resulting from finite element and spectral methods have traditionally been problematic for scientific visualization algorithms such as isosurfacing, slicing, and volume rendering. Solutions were limited to either linearly approximating high-order elements—still the most common approach—or calling custom functions provided by the simulation. Linear approximations introduce error, while custom callbacks require complex software interconnections between simulations and visualization, which are difficult post hoc when the simulation no longer is running. The MFA solves this problem because high-order evaluation of data points in any resolution is a closed-form operation with no additional error. In fact, all of the visualizations in this chapter were generated by evaluating a dense set of points from high-order MFA models of at least degree three and then applying existing visualization tools. In the future, the algorithms in those tools, such as volume rendering, can be written to ingest the MFA model directly, essentially moving the MFA point evaluation to be inside the visualization algorithm.

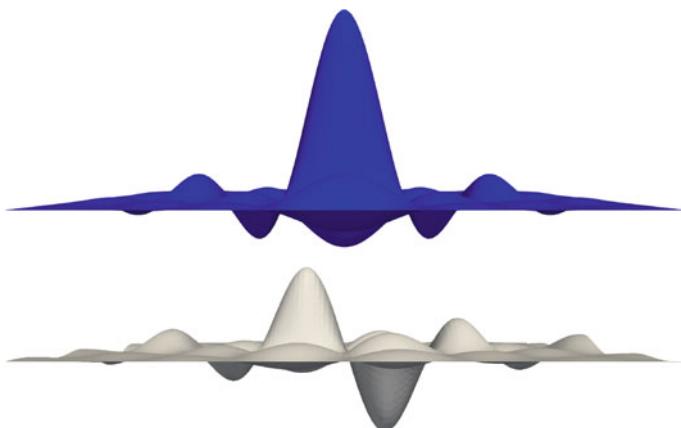


Fig. 14 Demonstration of post hoc evaluation and differentiation of an MFA model. Top: points evaluated from the MFA model of a 2-d sinc function. Bottom: first partial derivative with respect to x from the same model

Derivative fields. Differentiation, including high-order derivatives, is a staple of visualization and data analysis. Gradient fields, velocities, Jacobian matrices, edge detection, topological segmentation, and uncertainty quantification all require first derivatives. Second derivatives are used for computing curvature, acceleration, and Hessian matrices. For example, ridge and valley features are defined in terms of gradients and eigenvalues of the Hessian [15]. Applying lighting and shading to ridge features requires their normal, or the third derivative of the original model. Smooth derivatives are also required for optimization algorithms, used in machine learning (segmentation, classification, and regression, e.g.) that search for minima along directions of steepest descent. Figure 14 demonstrates computing a gradient field, showing evaluated points in the upper image and showing in the lower image the gradient field where the z coordinate is the first partial derivative of the MFA with respect to x . Both the upper and lower images were computed post hoc from the same previously computed and saved MFA model. It is easy to visually confirm that the slope in the top image corresponds to the value in the lower image. Other combinations of derivative degree (second derivative, e.g.) and partial and total derivatives are possible to compute from the same model post hoc.

Comparison of datasets with different discretizations. The ability to evaluate data points at arbitrary locations, not limited to locations of the input dataset, facilitates several analysis operations. One such operation is to compare two datasets whose resolutions or, more generally, discretizations differ. Discretization here includes element type, order, and resolution. Each of the input datasets can be modeled in a separate MFA, and then the two MFA models can be evaluated at the same parameter locations. Because the MFA is meshless, the field data of two heterogeneous datasets can be compared at uniform locations, independent of the mesh representation of each dataset.

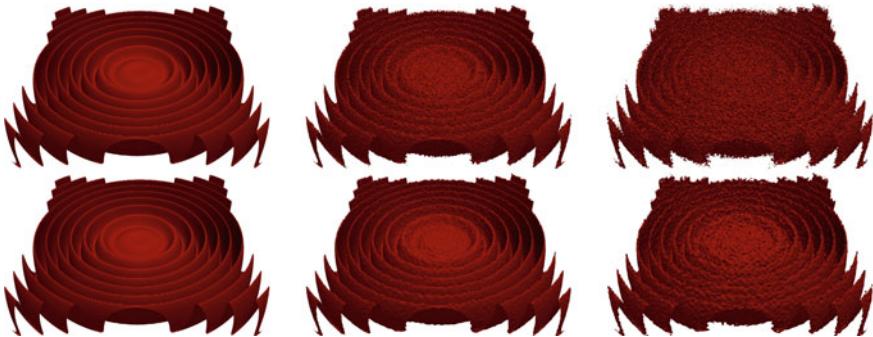


Fig. 15 Filtering noise with the MFA. Top row from left to right: Input dataset with 0% noise, 2% noise, and 5% noise. Bottom row: output of the MFA for the same input dataset as above shows that noise can be intentionally smoothed

Remapping between coupled multiphysics simulations. A related problem making use of high-order arbitrary evaluation is the transferring of a field from one discretization to another, as occurs in coupled multiphysics simulations. This process is usually called remapping. A further challenge in remapping is to satisfy physical constraints such as conservation of integrals and derivatives, for example, total energy of a system or flux across an interface. The linear least-squares method of solving the MFA can be replaced with nonlinear constrained optimization—as was done in Sect. 3.3—with penalty terms added for the physics constraints. Once modeled, points are evaluated from the MFA as described above.

Smoothing, simplification, filtering. Smoothing, simplification, and filtering are related operations that extract the underlying signal from a noisy dataset. Data arising from sensors and experiments usually contain noise that can mask the underlying behavior, which is often more smooth and has lower frequency than the raw data have. The MFA provides smoothing capability, being built from piecewise-continuous functions of high order. Compared with mesh simplification and topological simplification methods, the MFA is agnostic to mesh discretization and topology. Simplification or smoothing is accomplished by decreasing the number of control points while optionally increasing the degree of the MFA. These are common geometric editing operations of NURBS and B-splines [43]. Figure 15 demonstrates smoothing of a synthetic dataset [32] with different degrees of input noise, by using a high degree of $p = 10$ and half as many control points in each dimension as input points. The MFA in the lower row retains more of the low-frequency structure of the data than the input points in the upper row, where the structure is masked as the amount of noise increases. Alternatively, instead of being smoothed, high-frequency features could be retained by reducing the degree and increasing the number of control points (not shown).

5 Parallel Approximation and Evaluation

Computation of the MFA is parallelized on three levels. First, block parallelism (i.e., data parallelism) is used to decompose the domain into blocks and execute each block in distributed-memory compute nodes of a supercomputer or computing cluster. The DIY library [36] provides block parallelism. Within a block, task parallelism is utilized to approximate curves and evaluate points by using thread-level tasks. Currently, TBB [42] is the threading model used for CPU threading, but other task-parallel programming models can be used. For example, a SYCL [26] kernel is under development to parallelize the evaluation of multiple point locations, which can run on a GPU as well as on a CPU. Linear algebra operations, for example, to invert matrices, are SIMD (single instruction, multiple data) vectorized by using the Eigen [21] linear algebra library.

Each block of the parallel data model is a tensor product defined by n control points and $n + p + 1$ knots in each dimension. Control points are in the same coordinate system as original data points are, and therefore the same spatiotemporal domain decomposition used for input data can be reused for the MFA, minimizing data movement. The basis functions are not stored and are recomputed as needed.

The output MFA is stored in a binary file in DIY format, which is read and written in parallel by using MPI-I/O. There is also a serial utility to convert the file to VTK format so that the results are compatible with visualization and analysis tools derived from VTK such as ParaView [2] and VisIt [9]. In future work, it is anticipated that those tools will read the DIY file in parallel, and VTK filters will be written to operate directly on the MFA. One of the reasons for choosing a NURBS geometric basis for the MFA is its compatibility with such downstream tasks.

When computing an MFA approximation in parallel over multiple blocks in a spatial domain decomposition, the interior of each block will be C^p -continuous, p being the polynomial degree, but discontinuities exist across neighboring block boundaries. Grindeanu et al. [19] developed an efficient and scalable solution to this problem that involves blending neighboring approximations to ensure C^p continuity across block boundaries. They showed that after decomposing the domain in structured, overlapping blocks and approximating blocks independently to the desired accuracy, the local solution can be extended post hoc to the global domain by using compact, multidimensional smoothstep functions. This approach, which can be viewed as an extended partition of unity approximation method, is scalable on HPC architectures.

The global domain is first decomposed into rectangular blocks, which are extended by a fraction (ghost zone) to overlap the neighboring blocks. The MFA is then computed locally, in situ, independently for each extended block. The only difference between this step and the modeling algorithms presented earlier is that the blocks of input data are slightly enlarged by the ghost zone; otherwise, the modeling is identical.

Post hoc, additional communication is involved when evaluating points from the MFA. Blocks send requests for evaluated points in ghost zones to neighboring blocks and receive the evaluated points from their neighbors. The evaluated points from mul-

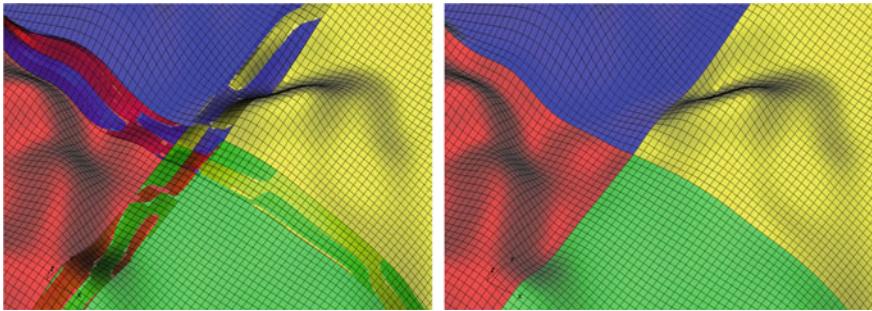


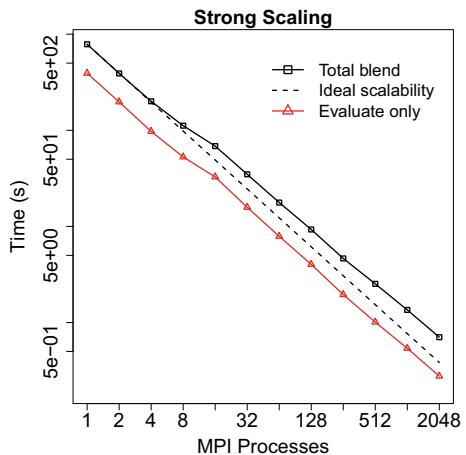
Fig. 16 Left: blocks modeled independently are discontinuous at the block boundaries. Right: blending using high-degree smoothstep functions restores continuity of the desired degree across block boundaries

tiple blocks that intersect in a ghost zone are blended together by using a smoothstep function, selected from a family of functions of varying degrees p , depending on the desired continuity C^p . These functions [16], used in computer graphics and visualization, are simple to evaluate, since they involve only polynomials of degree $2p + 1$. The value of the smoothstep function, α , is used in 1-d to blend two points P_1 and P_2 as $\alpha P_1 + (1 - \alpha)P_2$. By induction, Grindeanu et al. [19] extended the blending to higher dimensions and also proved that the error, with respect to the input data, of the blended point in the ghost zone is guaranteed to be within the same bounds as the user-specified error that was used to model the block interiors.

Blending occurs independently per dimension, with different blending functions and different ghost zone sizes possible in each dimension. The number of points being blended together depends on the dimensionality of the domain and on the number of neighboring blocks meeting at a face or corner. The result is a multidimensional partition of unity of the corresponding point evaluations that satisfies the same degree of continuity and same error bounds as the block interiors. Figure 16 shows a close-up of the junction between 4 independent blocks, before and after blending. Point evaluations are local operations per block, and blending is also a local operation after the values are received from neighboring blocks. For structured data, the communication pattern is predictable and localized to exchanging information only between neighboring blocks.

In order to evaluate the cost of the blending, parallel scaling studies were conducted on the Bebop cluster at Argonne's Laboratory Computing Resource Center. Bebop has 1,024 compute nodes, with Intel Broadwell processors having 36 cores and 128 GB of memory per node, connected by an Omni-Path fabric. Figure 17 shows a strong-scaling study on an S3D scientific dataset. The turbulent combustion dataset generated by an S3D simulation [8] of fuel jet combustion in the presence of an external crossflow [20] is 3-d ($704 \times 540 \times 550$); the field variable is the magnitude of the 3-d velocity, and $p = 3$ was used to model the MFA. The experiment measures how much time the post hoc point evaluation takes with and without blending, in parallel,

Fig. 17 Strong-scaling efficiency for the S3D dataset is 54%. The additional blending step generates high-degree continuity without changing the strong-scaling efficiency



using a previously computed and saved MFA model. Results show that blending added a factor of approximately 2X to the cost of evaluating, and the strong-scaling efficiency is 54% at 2,048 MPI processes.

6 Ongoing and Future Work

The techniques presented in this chapter are freely available for others to test and use in their own research,¹ and community contributions are welcome.

Research efforts continue along several fronts. Currently, the MFA for each block is computed *in situ* independently, without continuity constraints, and it is only during the evaluation of points from the MFA post hoc that continuity is achieved by blending evaluated points from neighboring blocks. An alternative approach is being investigated that enforces continuity constraints while neighboring MFA blocks are being solved *in situ*.

Research also is ongoing to determine an optimal set of knots—both number and location—based on properties of the discrete data. The choice of knot vector influences the resulting accuracy of the approximation. Methods to automatically determine a knot vector that achieves high approximation quality are being developed. At the core of the approach is a feature function that characterizes the amount and spatial distribution of geometric details in the input data by accumulating derivatives. Knots are then selected to evenly distribute the feature contents across their intervals. A solution to this problem in 1-d has recently been published [50], with future extensions to higher-dimensional and unstructured input data being pursued.

¹ Freely available at <https://github.com/t peterka/mfa>.

Also being developed is an alternative data model representation for the MFA based on T-splines. This is an adaptive model, analogous to adaptive mesh refinement, consisting of regions with various levels of adaptivity (numbers of knots and control points). In the current tensor product representation, every time a new control point is added, it is duplicated in every dimension of the hypervolume, essentially adding a hyperplane of control points in each dimension. The T-spline formulation sidesteps this requirement, at the expense of added complexity of the data model and potentially increased cost of computing the MFA and evaluating points from it. It is anticipated that the storage savings will be a favorable trade-off given the added complexity.

Acknowledgements This work is supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Margaret Lentz. This research used resources of the Argonne Leadership Computing Facility (ALCF), which is a DOE Office of Science User Facility supported under Contract No. DE-AC02-06CH11357, the Argonne Laboratory Computing Resource Center (LCRC), and the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

References

1. Austin, W., Ballard, G., Kolda, T.G.: Parallel tensor compression for large-scale scientific data (2015). [arXiv:1510.06689](https://arxiv.org/abs/1510.06689)
2. Ayachit, U.: The ParaView Guide: A Parallel Visualization Application. Kitware Inc, New York (2015)
3. Babuska, I., Melenk, J.M.: The partition of unity method. *Int. J. Numer. Methods Eng.* **40**(4), 727–758 (1997). [https://doi.org/10.1002/\(SICI\)1097-0207\(19970228\)40:4<727::AID-NME86>3e3.0.CO;2-N](https://doi.org/10.1002/(SICI)1097-0207(19970228)40:4<727::AID-NME86>3e3.0.CO;2-N)
4. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) *Modern Software Tools in Scientific Computing*, pp. 163–202. Birkhäuser Press, Basel (1997)
5. Bazilevs, Y., Calo, V.M., Cottrell, J.A., Evans, J.A., Hughes, T.J.R., Lipton, S., Scott, M.A., Sederberg, T.W.: Isogeometric analysis using T-splines. *Comput. Methods Appl. Mech. Eng.* **199**(5–8), 229–263 (2010)
6. Boashash, B.: *Time-frequency Signal Analysis and Processing: A Comprehensive Reference*. Academic, New York (2015)
7. Cavoretto, R., Rossi, A.D., Perracchione, E.: Partition of unity interpolation on multivariate convex domains. *Int. J. Model. Simul. Sci. Comput.* **06**(04) (2015). <https://doi.org/10.1142/S1793962315500348>
8. Chen, J.H., Choudhary, A., de Supinski, B., DeVries, M., Hawkes, E.R., Klasky, S., Liao, W.K., Ma, K.L., Mellor-Crummey, J., Podhorszki, N., Sankaran, R., Shende, S., Yoo, C.S.: Terascale direct numerical simulations of turbulent combustion using S3D. *Comput. Sci. Disc.* **2**, 015001 (2009)
9. Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagis, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P.: VisIt: an end-user tool for visualizing and analyzing very large data. In: *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pp. 357–372 (2012)
10. Cox, M.G.: The numerical evaluation of B-splines. *IMA J. Appl. Math.* **10**(2), 134–149 (1972)

11. Dalcin, L., Collier, N., Vignal, P., Cortes, A., Calo, V.: PetIGA: a framework for high-performance isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **308**, 151–181 (2016). <https://doi.org/10.1016/j.cma.2016.05.011>
12. De Boor, C.: On calculating with B-splines. *J. Approx. Theory* **6**(1), 50–62 (1972)
13. De Boor, C.: *A Practical Guide to Splines* (revised ed.). New York (2001)
14. Deville, M.O., Fischer, P.F., Mund, E.H.: High-order methods for incompressible fluid flow (2002)
15. Eberly, D.: *Ridges in Image and Data Analysis*, vol. 7. Springer Science & Business Media, Berlin (2012)
16. Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, Burlington (2003)
17. Ferraty, F., Vieu, P.: *Nonparametric Functional Data Analysis: Theory and Practice*. Springer Science & Business Media, Berlin (2006)
18. Fric, T.F., Roshko, A.: Vortical structure in the wake of a transverse jet. *J. Fluid Mech.* **279**, 1–47 (1994)
19. Grindeanu, I., Peterka, T., Mahadevan, V.S., Nashed, Y.S.: Scalable, high-order continuity across block boundaries of functional approximations computed in parallel. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–9. IEEE (2019)
20. Grout, R.W., Gruber, A., Yoo, C., Chen, J.: Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. *Proc. Combust. Inst.* **33**, 1629–1637 (2011)
21. Guennebaud, G., Jacob, B., et al.: Eigen Version 3 (2010). <http://eigen.tuxfamily.org>
22. Heil, C.E., Walnut, D.F.: Continuous and discrete wavelet transforms. *SIAM Rev.* **31**(4), 628–666 (1989)
23. Hua, J., He, Y., Qin, H.: Multiresolution heterogeneous solid modeling and visualization using trivariate simplex splines. In: Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications, pp. 47–58. Eurographics Association (2004)
24. Hughes, T.J., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Eng.* **194**(39), 4135–4195 (2005)
25. Jansen, M.H., Oonincx, P.J.: *Second Generation Wavelets and Applications*. Springer Science & Business Media, Berlin (2005)
26. Khronos Group: SYCL Overview (2020). <https://www.khronos.org/sycl>
27. Lancaster, P., Salkauskas, K.: *Curve and Surface Fitting*. Academic, New York (1986)
28. Lin, H., Maekawa, T., Deng, C.: Survey on geometric iterative methods and their applications. *Comput.-Aided Des.* **95**, 40–51 (2018)
29. Livescu, D., Khang, Y., Mohd-Yusof, J., Petersen, M., Grove, J.: CFDNS: a computer code for direct numerical simulation of turbulent flows. Technical Report (2009). Technical Report LA-CC-09-100, Los Alamos National Laboratory
30. Livescu, D., Ristorcelli, J., Petersen, M., Gore, R.: New phenomena in variable-density rayleigh-taylor turbulence. *Physica Scripta* (2010). In press
31. Majdisova, Z., Skala, V.: Radial basis function approximations: comparison and applications. *Appl. Math. Model.* **51**, 728–743 (2017)
32. Marschner, S.R., Lobb, R.J.: An evaluation of reconstruction filters for volume rendering. In: *Proceedings Visualization'94*, pp. 100–107. IEEE (1994)
33. Martin, T., Cohen, E., Kirby, M.: Volumetric parameterization and trivariate B-spline fitting using harmonic functions. In: *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, pp. 269–280. ACM (2008)
34. Martin, W., Cohen, E.: Representation and extraction of volumetric attributes using trivariate splines: a mathematical framework. In: *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pp. 234–240. ACM (2001)
35. Merzari, E., Pointer, W., Obabko, A., Fischer, P.: On the numerical simulation of thermal striping in the upper plenum of a fast reactor. In: *Proceedings of ICAPP 2010*. San Diego, CA (2010)

36. Morozov, D., Peterka, T.: DIY2: Data-parallel out-of-core library. In: Proceedings of the 2016 IEEE Large Data Analysis and Visualization Symposium LDAV'16. Baltimore, MD (2016)
37. Nashed, Y.S., Peterka, T., Mahadevan, V., Grindeanu, I.: Rational approximation of scientific data. In: International Conference on Computational Science, pp. 18–31. Springer (2019)
38. Neale, R.B., Chen, C.C., Gettelman, A., Lauritzen, P.H., Park, S., Williamson, D.L., Conley, A.J., Garcia, R., Kinnison, D., Lamarque, J.F., et al.: Description of the NCAR community atmosphere model (CAM 5.0). Technical Report (2010)
39. Park, S., Lee, K.: High-dimensional trivariate NURBS representation for analyzing and visualizing fluid flow data. *Comput. & Graph.* **21**(4), 473–482 (1997)
40. Peterka, T., Bard, D., Bennett, J.C., Bethel, E.W., Oldfield, R.A., Pouchard, L., Sweeney, C., Wolf, M.: Priority research directions for in situ data management: enabling scientific discovery from diverse data sources. *Int. J. High Perf. Comput. Appl.* **34**(4), 409–427 (2020)
41. Peterka, T., Youssef, S., Grindeanu, I., Mahadevan, V.S., Yeh, R., Tricoche, X., et al.: Foundations of multivariate functional approximation for scientific data. In: 2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV), pp. 61–71 (2018)
42. Pheatt, C.: Intel® threading building blocks. *J. Comput. Sci. Coll.* **23**(4), 298–298 (2008)
43. Piegl, L., Tiller, W.: The NURBS Book (1997)
44. Piegl, L.A., Tiller, W.: Parametrization for surface fitting in reverse engineering. *Comput.-Aided Des.* **33**(8), 593–603 (2001)
45. Ramsay, J.O., Hooker, G., Graves, S.: Functional Data Analysis with R and MATLAB. Springer Science & Business Media, Berlin (2009)
46. Ramsay, J.O., Silverman, B.W.: Applied Functional Data Analysis: Methods and Case Studies, vol. 77. Princeton, Citeseer (2002)
47. Raviv, A., Elber, G.: Interactive direct rendering of trivariate B-spline scalar functions. *IEEE Trans. Vis. Comput. Graph.* **7**(2), 109–119 (2001)
48. Rogers, D.F.: An Introduction to NURBS: With Historical Perspective. Elsevier, Amsterdam (2000)
49. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-splines and T-NURCCs. *ACM Trans. Graph. (TOG)* **22**(3), 477–484 (2003)
50. Yeh, R., Nashed, Y.S., Peterka, T., Tricoche, X.: Fast Automatic knot placement method for accurate B-spline curve fitting. *Comput.-Aided Des.* 102905 (2020)

A Simulation-Oblivious Data Transport Model for Flexible In Transit Visualization



Will Usher, Hyungman Park, Myoungkyu Lee, Paul Navrátil, Donald Fussell, and Valerio Pascucci

Abstract In transit visualization offers a desirable approach to performing in situ visualization by decoupling the simulation and visualization components. This decoupling requires that the data be transferred from the simulation to the visualization, which is typically done using some form of aggregation and redistribution. As the data distribution is adjusted to match the visualization's parallelism during redistribution, the data transport layer must have knowledge of the input data structures to partition or merge them. In this chapter, we will discuss an alternative approach suitable for quickly integrating in transit visualization into simulations without incurring significant overhead or aggregation cost. Our approach adopts an abstract view of the input simulation data and works only on regions of space owned by the simulation ranks, which are sent to visualization clients on demand.

W. Usher (✉)

Intel Corporation, Mountain View, USA

e-mail: will.usher@intel.com

H. Park

Electrical and Computer Engineering, Texas Advanced Computing Center, The University of Texas at Austin, Austin, USA

e-mail: hyungman@utexas.edu

M. Lee

Sandia National Laboratories, Albuquerque, USA

e-mail: mklee@ua.edu

The Department of Aerospace Engineering and Mechanics, The University of Alabama, Tuscaloosa, AL, USA

P. Navrátil

Texas Advanced Computing Center, The University of Texas at Austin, Austin, USA

e-mail: pnav@tacc.utexas.edu

D. Fussell

Computer Science, The University of Texas at Austin, Austin, USA

e-mail: fussell@cs.utexas.edu

V. Pascucci

SCI Institute, University of Utah, Salt Lake City, USA

e-mail: pascucci@sci.utah.edu

1 Introduction

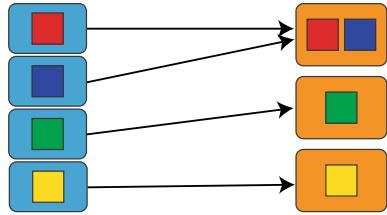
As discussed in the introduction chapter, two of the axes along which an in situ system can be classified are *proximity* and *access*. In situ methods can be roughly categorized on these axes as “tightly coupled” (same process, direct access) or “loosely coupled” (different process, indirect access). Loosely coupled approaches have also been referred to as *in transit*, and we adopt this terminology throughout the chapter. A tightly coupled approach provides clear benefits by eliminating data copies or the need to synchronize between multiple processes to transfer data; however, loosely coupled approaches can provide a desirable alternative at scale [11].

A loosely coupled approach can run the simulation and visualization on distinct nodes, reducing the impact of the visualization on the simulation [8, 30]. This separation is highly desirable when scalability or resource contention is of concern, as is often the case in large-scale simulations. Furthermore, the visualization component is free to run at a different level of parallelism than the simulation, or can run in parallel to the simulation to perform additional computation [3, 8, 18, 29, 30] or enable interactive visualization [23, 25]. The visualization can also be run on demand, starting and stopping as desired in a separate process or job, based on, e.g., simulation triggers [12]. However, the application must now deal with the challenge of coordinating the two processes to transfer data from the simulation to the visualization.

When considering applying in transit visualization in practice, several challenges remain. Most off-the-shelf libraries for in situ visualization target tightly coupled use cases [9, 28]. The few libraries that do support in transit typically do so by repurposing an existing I/O API [16, 26], and perform data aggregation and redistribution using a general distributed data access strategy (e.g., DataSpaces [5], FlexIO [31], Flexpath [4]). Although such approaches fit well into simulations already using the repurposed I/O API, migrating to a new I/O API for the express purpose of enabling in transit visualization may not be desirable, especially for simulations leveraging a custom optimized I/O library, such as HACC [10].

This chapter discusses the simulation-oblivious data transport model employed by libIS [23] for in transit visualization, which lowers the bar to using in transit visualization in practice. By adopting an abstract view of the simulation data, this approach can be integrated into a range of simulations employing arbitrary mesh types. This approach is also well suited to $M : N$ configurations, where M simulation ranks communicate with N visualization ranks, allowing each task to be run in its ideal configuration. This strategy natively supports asynchronous and on-demand in transit visualization, is portable across a range of HPC or ad hoc cluster systems, and does not require significant changes to the simulation to integrate. Moreover, we discuss developments extending libIS which allow greater portability across HPC systems and simulation codes.

Fig. 1 A 4 : 3 mapping of simulation data (left) to visualization clients (right). With $M > N$ each rank is assigned $\frac{M}{N}$ simulation states, with any remainder distributed among the clients



2 A Simulation-Oblivious Data Transport Model

Our simulation-oblivious data model treats each simulation rank as an opaque block of bytes, the interpretation of which is left up to the simulation and client. Specifically, the model does not inspect, adjust, or redistribute the input data distribution provided by the simulation, but instead simply forward each rank's data to the assigned client. Each client receives one simulation state from each simulation rank it is assigned to process data from, allowing for $M : N$ configurations (see Fig. 1).

If the simulation and visualization differ in their parallelism model (i.e., MPI-only vs. MPI + threads) or scalability, an $M : N$ configuration can be used to run each in its ideal configuration. In a 1 : 1 configuration, each client is assigned a single simulation rank's data. When $M > N$ each client is assigned data from $\frac{M}{N}$ simulation ranks, with any remainder distributed among the clients. The client application is then free to merge its assigned data together, or keep it distinct. Depending on the simulation and visualization configuration, it may not be possible for each client to combine the received set of simulation states into a single convex region. We note that this oblivious model supports only $M \geq N$, since it does not support splitting a rank's data to redistribute it. However, it is typically the case that the simulation is run at the same or higher level of parallelism than the visualization.

Although a somewhat primitive data model, this simplified view provides some desirable benefits over redistribution-based approaches. Removing the restructuring process reduces computational cost and library complexity, and eases integration into simulations with more exotic mesh types or primitives that may not be supported by the chosen redistribution strategy. Moreover, this approach is able to achieve high network utilization when querying data from the simulation, as little computation is performed during the process to assign data to clients, in turn reducing the simulation time spent performing data transfers to the clients. From a practical standpoint, this approach does not rely on more complex distributed data models or libraries, and can be easily deployed on a variety of systems, making it useful for lightweight in transit integrations, or as the underlying data transport layer for in full-featured systems (e.g., SENSEI [2]).

2.1 Implementation in libIS

The libIS library¹ [23] implements our oblivious data transport model to provide a lightweight library for asynchronous in transit visualization. A simulation using libIS acts as a data server that clients can connect to and query data from. Data are transferred to clients each timestep using the model described above. The library is split into a simulation and client library: the simulation library provides a C API to support simulations written in most languages, and the client library provides a C++ API to fit well with C++ based visualization software (e.g., VTK [22]). Visualization clients using libIS can connect and disconnect as desired from the simulation, to support on-demand execution of the visualization. Furthermore, libIS does not impose requirements on where the clients are run. The clients can be run on the same nodes as the simulation, distinct nodes on the same HPC resource, or an entirely separate HPC resource. The clients can be started manually by the user or automatically using, e.g., *in situ* triggers [12].

While our previously published work [23] established some of the benefits of libIS, this chapter describes recent efforts to improve portability across MPI runtimes and simulations, and to improve usability and performance. The library now includes a Fortran wrapper over the C API to ease integration into Fortran-based simulations and a fallback socket-based intercommunicator for portability across different HPC systems and MPI configurations. Moreover, this chapter presents an extensive evaluation of the performance of libIS and the impacts of libIS and visualization clients on the simulation in the supported configurations and communication modes.

2.1.1 Portable Communication Between the Simulation and Client

LibIS allows clients to connect and disconnect from the simulation as desired to enable on-demand execution. Rank 0 of the simulation spawns a background thread that listens for incoming connections on a socket. To establish an intercommunicator (Fig. 2), rank 0 of the client connects to this socket and sends the simulation its hostname and port to connect back to. The client is then able to request data from the simulation and can disconnect when its analysis is complete.

The initial version of libIS used an MPI intercommunicator, and thus required the `MPI_Open_port`, `MPI_Comm_accept` and `MPI_Comm_connect` APIs to establish it. In practice these APIs are not always available. For example, Theta at Argonne National Laboratory does not provide these APIs, which motivated the implementation of an MPI multilaunch mode in our initial work on libIS. However, the MPI multilaunch mode does not support on-demand execution of the visualization, as the visualization processes must be started together with the simulation in the same `mpirun` command, using MPI's MPMD launch mode.

To portably support on-demand connection in libIS, we have implemented a socket-based fallback intercommunicator. When attempting to connect, both the

¹ <https://github.com/Twinklebear/libIS>.

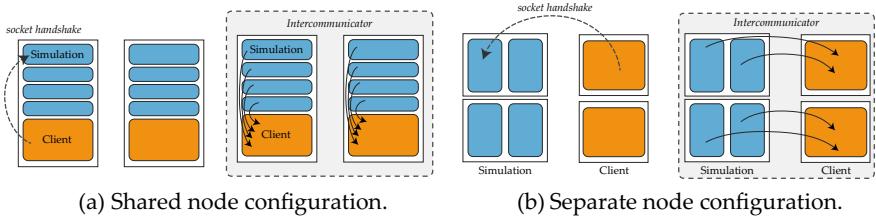


Fig. 2 Visualization clients using libIS can be run on the same nodes as the simulation (**a**), and thus take advantage of shared memory data transfers at the cost of oversubscription, or separate nodes (**b**), where the applications will not impact each other at the cost of communicating over the network. **a** shows a 4 : 1 configuration run on two nodes, **b** shows a 2 : 1 configuration on four nodes. The figures are reproduced from our previous work [23]

simulation and client test if the `MPI_Open_port` API is available. If the API is not available, they fall back to a socket-based intercommunicator. In this case, each client opens a socket and listens for connections from the simulation. Rank 0 of the simulation broadcasts rank 0 of the client's hostname and port number to the other ranks, which each connect to client rank 0 and receive the hostname and port numbers for the other clients. The simulation ranks then connect to the remaining clients to establish an all-to-all socket intercommunicator. When setting up a connection the simulation rank sends its rank number to the client, which sends back its rank number. Each socket on a process is indexed by the rank of the process on the other end to provide a send and receive API identical to MPI. Finally, to avoid flooding each client with incoming connections from a large simulation run, we rate limit the simulation ranks' connection requests. Without rate limiting the OS would see the large number of incoming connections as a network flooding attack.

The socket intercommunicator also enables running the clients on entirely different hardware and software stacks or HPC resources. Performing the in transit visualization on an entirely distinct cluster can be useful in sensitive or time critical applications, as discussed by Ellsworth et al. [8].

2.1.2 Querying Data

After establishing the intercommunicator, the client can request to receive data from the simulation. The simulation will probe for incoming messages from rank 0 of the client after each timestep when calling `libISPProcess`. The client can request a new timestep or disconnect from the simulation. If a new message is received, it is broadcast to the other simulation ranks and processed collectively.

Data are transferred from the simulation ranks to the clients using the data model and $M : N$ mapping discussed above. We compute N groups of $\frac{M}{N} : 1$ simulation to client groupings to assign data to clients. Each such group transfers data to the client rank independently and in parallel to the others. When using the MPI intercommunicator, data are transferred using point-to-point communication. When

```

// Set the world bounds (per-rank local/ghost bounds identical)
void libISSetWorldBounds(libISSimState *state, const libISBox3f box);

// Convenience method to set a 3D regular grid field
void libISSetField(libISSimState *state, const char *fieldName,
    const uint64_t dimensions[3], const libISDType type, const void *data);

// Convenience method to set an array of local and optional ghost particles
void libISSetParticles(libISSimState *state, const uint64_t numParticles,
    const uint64_t numGhostParticles, const uint64_t particleStride,
    const void *data);

// Set an arbitrary 1D buffer of data
void libISSetBuffer(libISSimState *state, const char *bufferName,
    const uint64_t size, const void *data);

// Call after each timestep to send data to any clients
void libISProcess(const libISSimState *state);

```

Listing 1: The libIS simulation API is used to configure a simulation state object, which stores pointers to the rank’s local data. Convenience methods are provided for regular 3D fields and particles; arbitrary 1D buffers of data can be sent via the buffer API.

using the socket intercommunicator, data are transferred using the socket connection established on each simulation rank to the assigned client. The data transfer avoids global all-to-all communication and scales well with the number of simulation and client ranks.

LibIS does not buffer the previous timestep to send to clients that requested data during computation. Clients requesting data will wait until the current timestep finishes and the data can be sent directly from the simulation’s memory. By not buffering the previous timestep, libIS does not need to keep an additional copy of the simulation state, reducing memory pressure.

2.1.3 The Simulation API

The simulation library provides a C API and a Fortran wrapper. Simulations begin listening for clients by calling `libISInit` and passing the port number to listen on. To make data available to clients, each rank configures a `libISSimState`, which stores pointers to the simulation data and metadata describing it (Listing 1).

The current API provides convenience wrappers for setting regular 3D fields and particles, along with a generic API to pass arbitrary 1D buffers of data. Internally, 3D fields and particles are treated the same as 1D buffers, since the data are not inspected or redistributed when sent to clients. Simulations using more complex data representations, e.g., unstructured meshes, octree AMR, block-structured AMR, etc.,

```

struct SimState {
    libISBox3f world, local, ghost;
    // The rank we received this data from
    int simRank;
    // The 3D fields and 1D buffers sent by the simulation
    std::unordered_map<std::string, Buffer> buffers;
    // The particles sent by the simulation, if any
    Particles particles;
};

// Connect to the simulation listening on rank 0 at host:port
void connect(const std::string &host, const int port,
    MPI_Comm ownComm, bool *sim_quit = nullptr);

// Query the next timestep, blocking until it is ready
std::vector<SimState> query(bool *sim_quit = nullptr);

// Asynchronously query the next timestep.
// The future can be monitored for completion
std::future<std::vector<SimState>> query_async(bool *sim_quit = nullptr);

// Disconnect from the simulation
void disconnect();

// Check if the simulation has terminated
bool sim_connected();

```

Listing 2: The libIS client API is used to connect to a running simulation and query data from it. Each client will receive $\frac{M}{N}$ simulation states, containing data from the simulation ranks it is assigned to receive data from.

can use the buffer API to send the local mesh data to clients. The clients must be tailored to the simulation and know how to interpret and process the incoming data.

The `libISProcess` function is called collectively by the simulation ranks to send data to any clients that have requested to receive the latest timestep. Data are sent directly from the simulation pointers shared with libIS when configuring the simulation state to avoid data copies.

2.1.4 The Client API

The libIS client library provides a C++ API (Listing 2) to integrate well into existing visualization software, which primarily uses C++. Clients first connect to the simulation by calling `connect`, after which they can query data using the blocking or asynchronous API. Once the desired analysis has completed, the client can disconnect and exit. The client can also check if the simulation has quit by passing an optional parameter to the API calls, or explicitly checking `sim_connected`.

Each client receives a vector of simulation states, containing the $\frac{M}{N}$ simulation ranks the client is assigned to receive data from. Each state contains the 3D and 1D buffers sent by the simulation, associated with their name, and any particles.

3 Example Use Cases

We evaluated the scalability and portability of libIS on two simulations, LAMMPS (Sect. 3.1) and Poongback (Sect. 3.2), using two HPC systems. We ran our benchmarks on Stampede2 at the Texas Advanced Computing Center (TACC) and Theta at Argonne National Laboratory. Both systems contain roughly similar Intel Xeon Phi Knight’s Landing (KNL) nodes: KNL 7250 on Stampede2 and KNL 7230 on Theta. Stampede2 contains an additional partition of Intel Skylake Xeon (SKX) nodes.

Although the KNL nodes are similar on the two systems, the network architecture differs significantly. Stampede2 employs a fat-tree topology Omnipath network, whereas Theta uses a 3-level Dragonfly topology Cray Aries network. On Stampede2 MPI uses the Omnipath network, which can provide up to 100 Gbps of bandwidth; however, sockets use the 1 Gbps ethernet network. Theta does not provide an ethernet network. Instead, sockets use the Aries network and can achieve a peak bandwidth of 14 Gbps.

The following sections compare the MPI and socket intercommunicators on Stampede2 and Theta using LAMMPS, to evaluate network performance portability and scalability with a test client application that queries data repeatedly (Sect. 3.1.1). We also present example use cases of in transit image database generation with LAMMPS (Sect. 3.1.2) and Poongback (Sect. 3.2.3) on Stampede2, and measure rendering performance of the client and impact on simulation performance. Finally, we provide a brief comparison against the existing redistribution-based data transfer method used in ADIOS (Sect. 3.3).

The in transit image database rendering benchmark is an example of using libIS in combination with an OSPRay-based [27] rendering application to render image databases, similar to those used in Cinema [1]. We modify the OSPRay mini-cinema example² to query data using libIS. After receiving the data mini-cinema renders a camera orbit around it using OSPRay’s data-distributed API [24] for data-parallel rendering. When the image set is finished, the application queries the next timestep and repeats for a specified number of timesteps.

3.1 LAMMPS

LAMMPS [19, 20] is a large-scale molecular dynamics simulation code, which we run MPI-parallel. To integrate libIS into LAMMPS, we leverage existing mechanisms

² <https://github.com/Twinklebear/mini-cinema>.

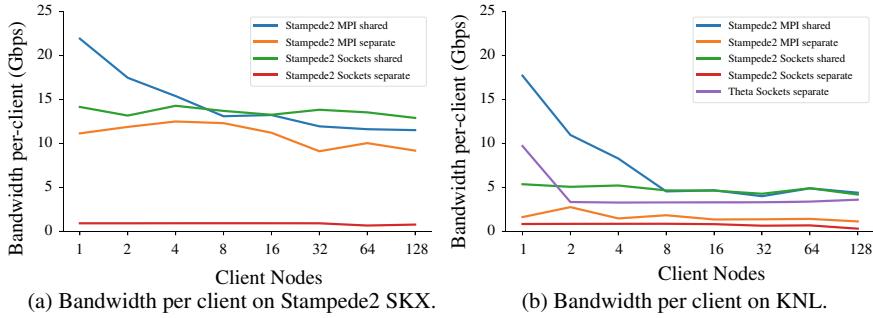


Fig. 3 Bandwidth per client rank achieved in the weak scaling benchmark. We find that the independent communication strategy employed by libIS achieves good weak scaling and high network utilization. Shared node runs are able to leverage shared memory for improved bandwidth

for coupling LAMMPS with other codes [21]. We build a wrapper application that behaves as the regular LAMMPS executable, with the difference that before the simulation starts, our wrapper initializes libIS and installs a fix callback to call `libISProcess` each timestep. This application is available as an example on the libIS GitHub.³ After each timestep, the simulation will send its local and ghost particles to clients querying data.

3.1.1 Performance Portability and Scalability

The weak scaling benchmark uses the Lennard Jones benchmark problem included with LAMMPS, which we replicate to store 131k particles per simulation rank. We measure the bandwidth achieved when querying data using the example client included with libIS, which simply prints out the received metadata. The benchmarks are run in a 16 : 1 configuration, where each client receives approximately 2.1M particles, which amounts to roughly 45MB of data per client after including ghost zones. Each group of 16 simulation ranks is placed on one node, and the client is run with one process per-node.

The benchmark is run on 1–128 client nodes, corresponding to 16 : 1 to 2048 : 128 configurations, using the SKX nodes on Stampede2 (Fig. 3a) and the KNL nodes on Stampede2 and Theta (Fig. 3b). In the shared node configuration, the client and simulation are run across the entire set of nodes, using the same resources. In the separate node configuration, we split the allocation of nodes in half between the simulation and client. The benchmark records bandwidth per client, and we compare the shared and separate configurations and the MPI and socket intercommunicators.

We find that the independent communication mode employed by libIS weak scales well, and can nearly saturate the 1Gbps ethernet network when using sockets on Stampede2. When using MPI over Omnipath on Stampede2, libIS is not network

³ <https://github.com/Twinklebear/libIS>.

bound, and averages 11% network utilization on SKX and 2% on KNL. When using sockets on Stampede2, libIS averages 88% network utilization on SKX and 74% on KNL. When using sockets on Theta, libIS averages 30% network utilization. The relatively low network utilization achieved with MPI could potentially be resolved by parallelizing the data transfer from the simulation ranks to their clients. Although each group of simulation ranks and clients communicates in parallel, the simulation ranks within a group send their data to the client in serial.

When run in the shared node configuration, shared memory can be used for higher bandwidth by both the MPI and sockets intercommunicators, and we find the sockets intercommunicator achieves performance similar to MPI. The shared node configuration avoids the practical challenges of queuing and running a second job on the HPC system and is likely to be a common use case for in transit visualization. The ability of both MPI and sockets to use shared memory for higher bandwidth is especially promising for performance portability.

3.1.2 In Transit Image Database Rendering

For the image database rendering example, we use the Rhodopsin benchmark input and run LAMMPS in the same 16 : 1 configuration as before. OSPRay uses an MPI + Threads model and is run with a single rank per node, whereas LAMMPS is run MPI-parallel with 16 ranks per node. The Rhodopsin benchmark stores 32 k particles per simulation rank, resulting in each client rank receiving 512 k particles in the 16 : 1 configuration. After accounting for ghost zones, this corresponds to roughly 20 MB of data per-client. The additional ghost particles are used to compute ambient occlusion to provide better depth cues (see Fig. 4). The benchmark renders an orbit of 80 camera positions around the data, and does so for 50 timesteps. OSPRay’s asynchronous rendering support is used to reduce the total time to render the set of by rendering eight images in parallel. We measure the total time to render all 80 frames for each timestep at 1024^2 pixels, and compare the shared and separate configurations using the MPI and sockets intercommunicators (Fig. 5).

We find that at low node counts, the difference in the total render time between the MPI and socket intercommunicators is unexpectedly large. The different communicators affect only the data query with libIS, which is not timed in the benchmark, and we would expect to observe similar performance. At higher node counts, the MPI and socket modes converge to similar render times, with the separate node configuration achieving slightly better performance than the shared node one, as expected. The shared node configuration runs both the simulation and renderer on the same nodes, and some performance degradation of both is to be expected. On KNL each configuration achieves roughly similar performance, potentially due to the larger number of available cores for the renderer, and the relatively weaker per-core performance compared to SKX. We note that the overall render time decreases in the benchmark, which is a result of each client’s local data projecting to a smaller region of the image as the data set is scaled up.

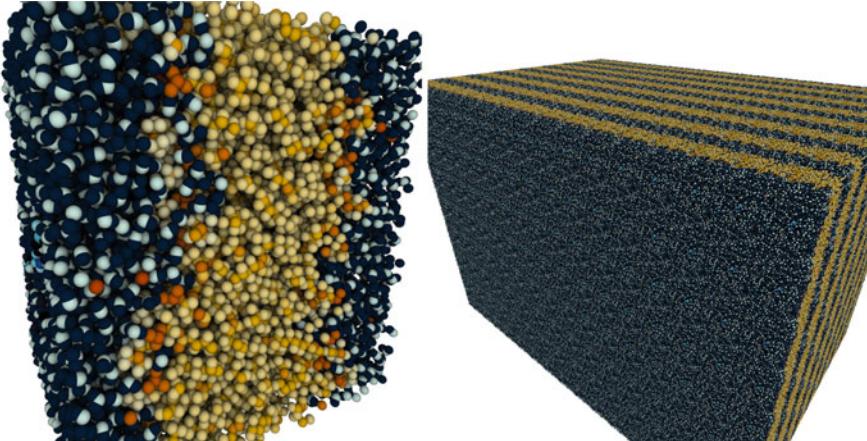


Fig. 4 The LAMMPS Rhodopsin benchmark rendered with ambient occlusion. Left: the simulation data on a single rank. Right: replicated in our weak scaling benchmark for 1024 ranks. By using the ghost particles already computed by LAMMPS, our in transit renderer is able to compute ambient occlusion effects using only the local data available on each rank

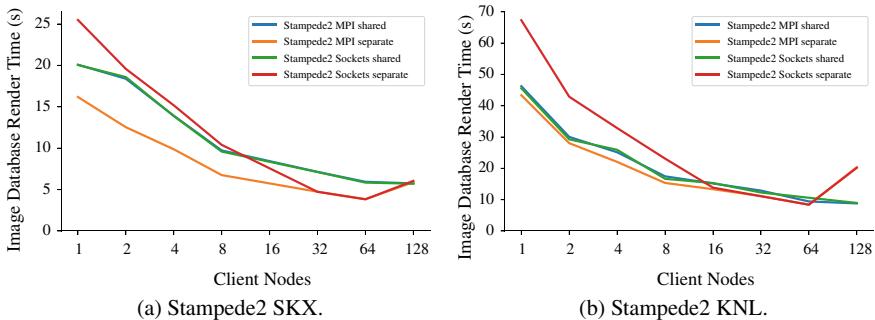


Fig. 5 Camera orbit render times on SKX and KNL for each timestep. We find that the image database render task scales well, with the shared node configuration slightly impacted by the simulation. We find a performance decrease at 128 clients in the separate configuration, which may be due to a different network placement for the 256 node job impacting compositing performance

3.1.3 Impact on Simulation Performance

Finally, we measure the impact on simulation performance for each configuration by comparing the time taken to compute each timestep with and without the mini-cinema client running (Fig. 6). In the separate node configuration, the simulation and client are run on distinct nodes, and do not contend for resources. Thus, the simulation is impacted only by the time spent sending data to the clients. On the Stampede2 SKX nodes, libIS has a negligible impact when using MPI or sockets, though we do observe an outlier at 128 nodes where a larger impact when using sockets is observed.

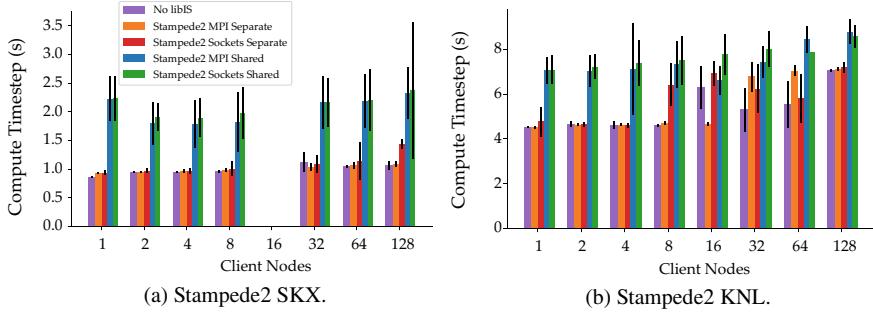


Fig. 6 The impact of the image database rendering client on LAMMPS performance. Bars show the average time to compute each timestep in the different modes, with standard deviation shown as error bars. As expected, the shared node configuration significantly impacts performance, whereas the separate node configuration has relatively little effect. On 16 SKX nodes the LAMMPS simulation became unstable, and we were unable to run benchmarks in that configuration. The plots share the same legend

On the KNLs the simulation is 4% slower on average when using MPI, with a similar impact observed for sockets, though we do observe an outlier at 16 nodes where a greater impact when using sockets is measured. The greater impact on KNL can be attributed to the lower single-core performance compared to SKX, which increases the time spent transferring data to the clients.

The shared node configuration runs the simulation and client on the same nodes, potentially leading to significant resource contention and impacting simulation performance. On SKX simulation takes 82% longer when using MPI and 104% longer when using sockets. On KNL the simulation takes 44% longer when using MPI and 44% longer when using sockets. The reduced impact on KNL is likely attributable to the larger number of cores available, which may reduce resource contention for processors, and the availability of MCDRAM, which in this configuration is large enough to hold the data of both LAMMPS and mini-cinema.

3.2 Direct Numerical Simulation of Turbulent Channel Flow with Poongback

We simulate a large-scale turbulent channel fluid flow using Poongback [13–15, 17], and use libIS to transfer data to the mini-cinema clients as before. Poongback is a computational fluid dynamics (CFD) solver for direct numerical simulation (DNS) of incompressible turbulent channel flows written in Fortran. The simulation generates a 3D volume data set and runs MPI-parallel. To integrate libIS into Poongback, we use libIS’s Fortran wrapper. Minimal modifications to the simulation code are required to integrate in transit visualization through libIS. First, before the Poongback simulation begins, we initialize libIS and configure the libIS simulation state on each rank with

Table 1 Weak scaling configurations for the Poongback benchmark, targeting roughly 1.4 GB of volume data per client rank

Client ranks	Total voxels	Total volume size (GB)
1	$384 \times 768 \times 576$	1
2	$768 \times 768 \times 576$	3
4	$768 \times 768 \times 1152$	5
8	$1536 \times 768 \times 1152$	11
16	$1536 \times 768 \times 2304$	22
32	$3072 \times 768 \times 2304$	43
64	$3072 \times 768 \times 4608$	87
128	$6144 \times 768 \times 4608$	174

the bounds of its local volume data. After each timestep we call the `libISProcess` wrapper to send the data on to any clients requesting the current timestep. Poongback stores its data row-major, and does not require a transpose to be done after receiving data on the client.

3.2.1 Evaluation Setup

We evaluate libIS with Poongback using mini-cinema for in transit image database rendering, and record network utilization, rendering performance, and the impact on simulation performance. The Poongback benchmarks are run on Stampede2 SKX and KNL nodes, using a weak scaling benchmark for Poongback. To create the weak scaling benchmark, we proportionally increase the simulation dimensions based on the number of clients, to maintain roughly 1.4 GB of volume data per client (Table 1). The simulation and clients are run in a 32 : 1 configuration. Poongback is run MPI-parallel with 32 ranks per node, whereas the mini-cinema clients are run using MPI + Threads with one rank per node. In our benchmark we compare both the shared and separate configurations and the MPI and socket intercommunicators. When using the shared node configuration, the simulation and application are run on the same nodes, whereas in the separate configuration half the nodes are assigned to the simulation and half to the client. The benchmarks are run up to an aggregate of 128 SKX nodes and 256 KNL nodes. For shared node runs, we scale from a 32 : 1 configuration up to a 4096 : 128 one on SKX and KNL. For separate node runs, we scale from a 32 : 1 configuration up to a 2048 : 64 one on SKX and a 4096 : 128 one on KNL.

3.2.2 Performance Portability and Scalability

We measure the average data transfer bandwidth achieved on each client over 50 timesteps (Fig. 7). When using MPI in the separate node configuration, data are

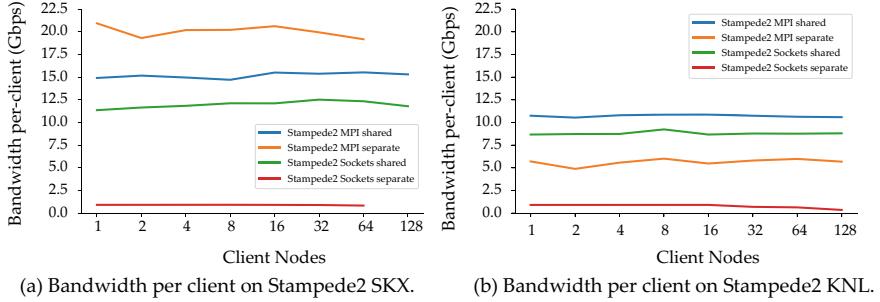


Fig. 7 Bandwidth per client rank achieved in the Poongback weak scaling benchmark. We find that the independent communication strategy employed by libIS achieves good weak scaling, and performs well when transferring the larger portion of data sent by each Poongback rank

transferred over the 100 Gbps Omnipath network, and libIS achieves an average of 20% network utilization on SKX and 6% on KNL. When using sockets in the separate node configuration, data are transferred over the 1 Gbps ethernet network, and libIS achieves 90% network utilization on SKX and 78% on KNL. As discussed previously, parallelizing the data transfer within each set of simulation ranks and client could improve network utilization when using MPI, especially for the larger aggregate data transfer performed for the Poongback simulation.

In the shared node configuration, each set of 32 : 1 ranks is run on the same node, ensuring communication is local to each node. As a result, data can be transferred using shared memory instead of over the network to achieve higher bandwidth. A notable exception is on SKX (Fig. 7a), where higher bandwidth is achieved in the separate configuration than in the shared one when using MPI. This result is counter to our results on KNL with Poongback and on KNL and SKX with LAMMPS, and warrants further investigation.

Overall, we achieve higher bandwidth on SKX than KNL, likely due to the higher per-core performance of SKX. Similar to our findings with LAMMPS in Sect. 3.1.1, MPI achieved higher bandwidth than sockets, as MPI is able to leverage the faster Omnipath network. However, in the shared configuration both can leverage shared memory and the gap between the two narrows. In terms of overall weak scalability, the independent data transfer strategy employed by libIS scales well with the number of clients, with per client bandwidth remaining nearly constant across each scaling run.

3.2.3 In Transit Image Database Rendering

For each of the 50 timesteps queried during the benchmark, the mini-cinema client renders an 80-position camera orbit around the data set. Each image is rendered at a 1024^2 resolution, with one sample per pixel and a volume sampling rate of one sample per voxel. In the largest run with 128 client ranks, the application queries

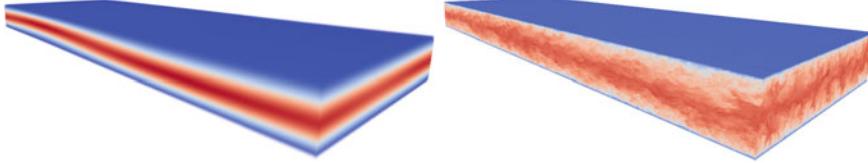


Fig. 8 Images of the Poongback turbulent channel-flow simulation rendered using our mini-cinema libIS client. Both volumes are $6144 \times 768 \times 4608$ voxels in double-precision floating-point values (174 GB). Left: The simulation state at timestep 0, the initial condition used in our weak scaling benchmarks. Right: A checkpoint of the same simulation (fully developed turbulence)

and renders a total of 8.7 TB of volume data over the course of the 50 timesteps. We disable the asynchronous rendering functionality of mini-cinema to reduce the impact of the client in the shared node configuration, and disable it in the separate configuration for consistency (Fig. 8).

The time taken to render each camera orbit is shown in Fig. 9. In contrast to the results observed with LAMMPS, the overall rendering time increases as additional client ranks are added. We believe this to be due to the differing data distributions of the two simulations. Compared to LAMMPS, where each simulation rank has a cube of data, Poongback partitions its data using a pencil decomposition [13]. After multiple pencil-pencil data transposes, the set of regions assigned to each client is a group of these x-pencils along the axis of flow, so that each pencil spans the entire x-axis of the data set. The set of regions assigned to each client is a group of these x-pencils, where each pencil spans the entire x axis of the data set. The set of regions project to a large number of pixels for most of the viewpoints in the orbit. The number of pixels covered does not reduce significantly as more ranks are added, leading to a significant amount of rendering and compositing work. One measure that could be taken to alleviate the compositing work is to merge the 32 regions assigned to each client rank into a single OSPRay rendering region, allowing them to be rendered and composited as a single brick, instead of 32 separate ones. This optimization would reduce the compositing work by a factor of 32, allow for faster local rendering, and provide a meaningful performance improvement.

We find similar results as those observed with LAMMPS when comparing the relative rendering performance of the different configurations and communicators. Although we find little impact of the communication method used (MPI versus sockets) on render time, we find a significant impact on performance when using the shared configuration. The shared configuration can lead to contention between the simulation and client, and impacts the performance of both.

3.2.4 Impact on Simulation Performance

Finally, we measure the impact of connecting the mini-cinema client in the different configurations on simulation performance (Fig. 10). We find that the overhead is

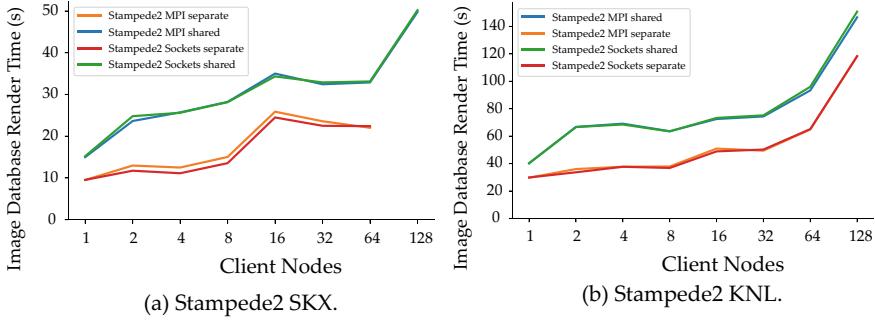


Fig. 9 Camera orbit render times on SKX and KNL for each timestep in our weak scaling benchmark. We achieve better total rendering performance on SKX, and in the separate node configurations, as the shared node configuration oversubscribes the nodes. In contrast to the LAMMPS results, we find poorer scaling at higher node counts, potentially due to the differences in data distribution and compositing workload

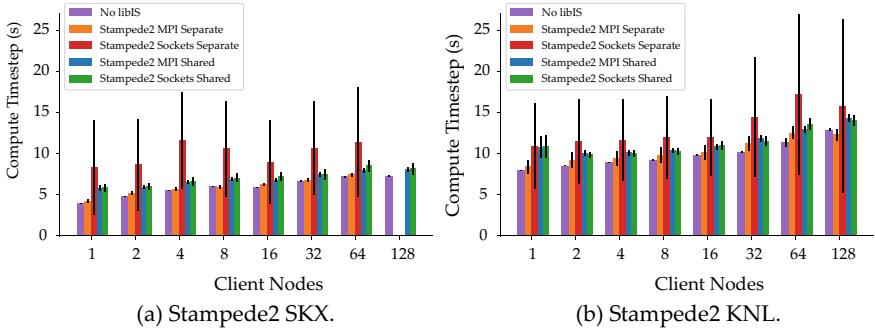


Fig. 10 The impact of the image database rendering client on Poongback performance. Bars show the average time to compute each timestep in the different modes, with standard deviation shown as error bars. Overall, the time-consuming socket-based communication in the separate configuration incurs the most overhead. By not rendering multiple frames in parallel, the renderer's impact on the simulation is reduced in the shared configurations

primarily determined by the ratio of compute time and data transfer time for each timestep. Depending on how large this ratio is, the simulation becomes either compute bound or communication bound. Thus, the impact on simulation time is related to the network performance measured in Sect. 3.2.2. We also observe a much higher standard deviation in the overhead, which is likely tied to varying network performance when using sockets, while MPI and local sockets are able to provide more consistent performance. Moreover, when not rendering multiple frames in parallel, the rendering client has little impact on the simulation in the shared configuration.

When using the socket communicator in the separate mode, the simulation becomes communication bound, and must wait for the data transfer to complete

before advancing, thereby impacting performance. The sockets separate configuration has the greatest impact on simulation performance of the different modes, increasing compute time by 78% on SKX and 34% on KNL. The impact is greater on SKX as faster nodes lead to a $1.7 \times$ faster simulation, exacerbating the impact of the simulation becoming bound by the data transfer. In the other configurations, the impact is less severe. The MPI separate mode increases simulation time by 4 and 6%, on SKX and KNL, respectively; the MPI shared mode by 18 and 14%; and the sockets shared mode by 22 and 17%.

3.3 Comparison to Existing Libraries

To compare data transfer performance against existing restructuring-based techniques we provide a brief comparison against the widely used ADIOS [16] I/O library (version 2.5.0). ADIOS repurposes its existing I/O API to support in transit visualization, allowing users to simply change the I/O “engine” being used by the simulation and client. ADIOS adopts a lightweight data model, processing one-, two-, or three-dimensional arrays of primitive types which are passed to the I/O engine. However, in contrast to libIS’s oblivious data model, ADIOS supports redistributing the data to the clients. Each client specifies a starting offset within an array and number of elements to be read, as if reading from a file. ADIOS will then make the requested data available on that rank. Though this provides a transparent transition from a file-based pipeline, in an in transit scenario this requires some form of data redistribution, potentially adding overhead to the data transfer.

We write a test application which generates 131k particles per rank and run the simulation and client in a 16 : 1 configuration, matching the Lennard Jones benchmark configuration in Sect. 3.1.1. Each particle attribute (x , y , z , type) is passed to ADIOS as a global array. Clients request the subregion of this array corresponding to their assigned set of simulation ranks. To transfer data we use the “SST” engine, which supports $M : N$ data transfer and will make use of RDMA enabled networks where available.

In both the shared and separate configurations ADIOS achieves similar data transfer speeds, averaging 0.5 Gbps per client on SKX and KNL when run with 16 clients on Stampede2. In contrast, libIS averages 13 Gbps and 11 Gbps respectively on SKX, and 5 Gbps and 1.5 Gbps respectively on KNL (see Fig. 3). The potential need for more global communication among ranks to perform data redistribution comes at a cost compared to our independent model, though is convenient for applications.

SENSEI [2], which provides adapters to bridge between the multitude of in situ frameworks to ease portability, could select between ADIOS or libIS for data transport, depending on the application’s needs. For example, if the visualization task work on subblocks of data instead of requiring redistribution, libIS can be used for increased bandwidth. If this is not the case, ADIOS could be used to redistribute the data to meet the client’s needs.

4 Conclusion

We have presented a simulation-oblivious data transport model for in transit visualization. Although a relatively primitive model, this approach is desirable in a number of scenarios. When time spent by the simulation in data aggregation and redistribution is of concern, our oblivious approach eliminates this step to enable high bandwidth communication, in turn reducing impact on simulation performance. This approach also imposes no restrictions on the simulation runtime configuration or data structures, supporting $M : N$ configurations and arbitrary buffers of data.

From a practical standpoint, our implementation in libIS provides an easy-to-use C API that can be integrated into simulations with minimal changes to the simulation code. The data model does not require advanced networking functionality, and can use MPI or sockets to transfer data between the simulation and client. Although the sockets intercommunicator may not be able to take advantage of some high-speed interconnects, it ensures portability across different HPC or ad hoc clusters. The ease of use, flexibility, and portability of libIS also make it a useful building block for more full-featured libraries (e.g., SENSEI [2]), where libIS can serve as a base data transport layer.

In our evaluation we compared the separate and shared node configurations, and the MPI and socket intercommunicators on two HPC systems using two simulation codes. We found that the independent communication model employed by libIS weak scales well across the systems and network architectures. The fast data transfer to clients achieved by libIS means the simulation performance is not severely impacted when running the visualization on separate nodes. The library also allows clients to connect and disconnect as desired, which is suitable for spurious and on-demand visualization tasks.

Through our evaluation we have found further areas for improvement in libIS's networking model. When sending data to a client rank, each of the assigned $\frac{M}{N}$ simulations ranks sends its data serially to the client. This serialization of the data transfer leads to underutilizing the higher bandwidth network architectures, and makes it more likely for the simulation compute time to become bound by the libIS data transfer time, as observed with Poongback. Parallelizing the data transfer within each set of simulation ranks and client would further improve network utilization, and alleviate these issues.

In practice, we recommend using the separate node configuration for long-lived visualization tasks, and the shared node configuration for short-lived on-demand tasks. Short-lived tasks do not impact the simulation performance for a long period, and from a practical standpoint it may not be possible to start an additional job quickly enough to run the task in time. Oversubscribing the nodes for a short period to run the task is a better option than blocking the simulation until the task starts, or missing the desired timestep entirely. Potential future changes to HPC job schedulers to enable co-scheduling [6, 7] could make it easier to run on-demand tasks on separate nodes to reduce the impact of the visualization.

Acknowledgements This work is supported in part by the Intel Graphics and Visualization Institute of eXcellence at the Scientific Computing and Imaging Institute, University of Utah and at the Texas Advanced Computing Center at the University of Texas at Austin. This work is supported in part by NSF: CGV Award: 1314896, NSF:IIP Award: 1602127, NSF:ACI Award: 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC Award: 1842042. The software development of PoongBack is supported by NSF PetaApps grants: OCI-0749223 and NSF PRAC Grant 0832634. This work used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility supported under Contract DE-AC02-06CH11357. The authors acknowledge TACC for providing HPC resources that have contributed to the research results reported in this paper. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

1. Ahrens, J., Jourdain, S., O'Leary, P., Patchett, J., Rogers, D.H., Petersen, M.: An Image-based approach to extreme scale in situ visualization and analysis. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (2014)
2. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: The SENSEI generic in situ interface. In: Proceedings of the 2nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization, ISAV '16 (2016)
3. Bennett, J.C., Abbasi, H., Bremer, P.T., Grout, R., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (2012)
4. Dayal, J., Bratcher, D., Eisenhauer, G., Schwan, K., Wolf, M., Zhang, X., Abbasi, H., Klasky, S., Podhorszki, N.: Flexpath: type-based publish/subscribe system for large-scale science analytics. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2014-05)
5. Docan, C., Parashar, M., Klasky, S.: DataSpaces: an interaction and coordination framework for coupled simulation workflows. Cluster Comput. (2012)
6. Dorier, M., Dreher, M., Peterka, T., Wozniak, J.M., Antoniu, G., Raffin, B.: Lessons learned from building in situ coupling frameworks. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (2015)
7. Dorier, M., Yildiz, O., Peterka, T., Ross, R.: The challenges of elastic in situ analysis and visualization. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV '19. Denver, Colorado (2019)
8. Ellsworth, D., Green, B., Henze, C., Moran, P., Sandstrom, T.: Concurrent visualization in a production supercomputing environment. IEEE Trans. Vis. Comput. Graph. (2006)
9. Fabian, N., Moreland, K., Thompson, D., Bauer, A., Marion, P., Geveci, B., Rasquin, M., Jansen, K.: The paraview coprocessing library: a scalable, general purpose in situ visualization library. In: Symposium on Large Data Analysis and Visualization (LDAV) (2011)
10. Habib, S., Pope, A., Finkel, H., Frontiere, N., Heitmann, K., Daniel, D., Fasel, P., Morozov, V., Zagaris, G., Peterka, T., et al.: HACC: simulating sky surveys on state-of-the-art supercomputing architectures. New Astron. (2016)
11. Kress, J., Klasky, S., Podhorszki, N., Choi, J., Childs, H., Pugmire, D.: Loosely coupled in situ visualization: a perspective on why it's here to stay. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV (2015)

12. Larsen, M., Woods, A., Marsaglia, N., Biswas, A., Dutta, S., Harrison, C., Childs, H.: A flexible system for in situ triggers. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization—ISAV ’18. Dallas, Texas (2018)
13. Lee, M., Malaya, N., Moser, R.D.: Petascale direct numerical simulation of turbulent channel flow on up to 786K Cores. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC’13 (2013)
14. Lee, M., Moser, R.: Direct numerical simulation of turbulent channel flow up to $\text{Re}_\tau \approx 5200$. *J. Fluid Mech.* (2015)
15. Lee, M., Ulerich, R., Malaya, N., Moser, R.D.: Experiences from leadership computing in simulations of turbulent fluid flows. *Comput. Sci. Eng.* (2014)
16. Lofstead, J., Zheng, F., Klasky, S., Schwan, K.: Adaptable, metadata rich IO methods for portable high performance IO. In: IEEE International Symposium on Parallel & Distributed Processing, 2009. IPDPS (2009)
17. Malaya, N., McDougall, D., Michoski, C., Lee, M., Simmons, C.S.: Experiences porting scientific applications to the intel (KNL) Xeon Phi platform. In: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact (2017)
18. Moreland, K., Oldfield, R., Marion, P., Jourdain, S., Podhorszki, N., Vishwanath, V., Fabian, N., Docan, C., Parashar, M., Hereld, M.: Examples of in transit visualization. In: Proceedings of the 2nd International Workshop on Petascal Data Analytics: Challenges and Opportunities (2011)
19. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* (1995)
20. Sandia National Laboratories: LAMMPS Molecular Dynamics Simulator. <https://lammps.sandia.gov>
21. Sandia National Laboratories: Coupling LAMMPS to Other Codes. Accessed Jan 2020. https://lammps.sandia.gov/doc/Howto_couple.html
22. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit, 4th edn. Kitware, New York (2006)
23. Usher, W., Rizzi, S., Wald, I., Amstutz, J., Insley, J., Vishwanath, V., Ferrier, N., Papka, M.E., Pascucci, V.: libIS: a lightweight library for flexible in transit visualization. In: ISAV: In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV’18 (2018)
24. Usher, W., Wald, I., Amstutz, J., Günther, J., Brownlee, C., Pascucci, V.: Scalable ray tracing using the distributed framebuffer. *Comput. Graph. Forum* (2019)
25. Usher, W., Wald, I., Knoll, A., Papka, M., Pascucci, V.: In situ exploration of particle simulations with CPU Ray tracing. *Supercomput. Front. Innov.* (2016)
26. Vishwanath, V., Hereld, M., Morozov, V., Papka, M.E.: Topology-aware data movement and staging for I/O acceleration on blue gene/P supercomputing systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM (2011)
27. Wald, I., Johnson, G.P., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navrátil, P.: OSPRay—a CPU ray tracing framework for scientific visualization. *IEEE Trans. Vis. Comput. Graph.* (2017)
28. Whitlock, B., Favre, J.M., Meredith, J.S.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, EGPGV ’11 (2011)
29. Zanúz, H.C., Raffin, B., Mures, O.A., Padrón, E.J.: In-transit molecular dynamics analysis with apache flink. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization—ISAV ’18. Dallas, Texas (2018)
30. Zhang, F., Laslusa, S., Jin, T., Rodero, I., Bui, H., Parashar, M.: In-situ feature-based objects tracking for large-scale scientific simulations. In: High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion (2012)

31. Zheng, F., Zou, H., Eisenhauer, G., Schwan, K., Wolf, M., Dayal, J., Nguyen, T.A., Cao, J., Abbasi, H., Klasky, S., Podhorszki, N., Yu, H.: FlexIO: I/O middleware for location-flexible scientific data analytics. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (2013)

Distributed Multi-tenant In Situ Analysis Using Galaxy



Greg Abram, Paul Navrátil, David Rogers, and James Ahrens

Abstract Galaxy is a multi-tenant server platform for visualization and analysis of data distributed across many nodes of a supercomputer. It presents an interface through which multiple concurrent clients can connect to perform visualization and analysis tasks on shared data. Galaxy minimizes the memory footprint of visualization and analysis tasks to reduce the impact of analysis on the simulation. It does so by utilizing a visualization renderer that incorporates common visualization techniques directly in the rendering process. The client GUI provides a data-flow programming paradigm that enables users to connect to the Galaxy Multiserver, devise complex multi-step analytics workflows, and visualize results on their desktop.

1 Introduction

As discussed in the opening chapter, modern supercomputing-level computation calls for in situ analysis. By analyzing the data *as it is being generated*, in situ analysis avoids the necessity of saving time-step data snapshots for post-processing, a costly process that generally limits the temporal frequency at which data can be saved. However, for current systems (e.g. ParaView/Catalyst [3], VisIt/libsim [5], SENSEI [4]) these benefits bear significant costs. For example, the analysis to be performed must be known prior to the start of the computation so that the necessary changes can be made to the general-purpose simulation. Within these systems, the

G. Abram · P. Navrátil (✉)

Texas Advanced Computing Center, The University of Texas, Austin, TX, USA
e-mail: pnav@tacc.utexas.edu

G. Abram

e-mail: gda@tacc.utexas.edu

D. Rogers · J. Ahrens

Los Alamos National Laboratory, Los Alamos, NM, USA
e-mail: dhr@lanl.gov

J. Ahrens

e-mail: ahrens@lanl.gov

distributed simulation process is instrumented with an adapter that converts the simulation’s run-time data to a form palatable for analysis. This results in a monolithic simulation/analysis code that, when run, performs a specific simulation/analysis task, and requires two different customizations: to the simulation, so that the simulation provides the required internal data to the adapter, and to the analysis, to cause it to perform the specific analysis required by the science case.

Typically, such systems co-locate simulation and analysis work in the same actual processes. Consequently, simulation and analysis share memory, computation and communication resources and therefore affect the performance of one another. The available memory available to each process must be sufficient to serve the worst cases load of each phase. Unfortunately, while the footprint of many simulations can be determined, the footprint and computational cost of typical geometry-pipeline visualization algorithms is hard to predict and, in the worst case, is tied to the size of the underlying computational grid. Further, as distributed supercomputers become ever larger, system-wide synchronization becomes costly. When the workload of a task is not evenly distributed, synchronizing may cause much of the computational resources assigned the task to wait for a few overburdened participants to complete.

Given these challenges, Galaxy aims to provide the following differentiating capabilities: (1) separation of simulation and analysis activities so that multiple, concurrent and asynchronous analyses of simulation data products can attach, operate and disconnect at run-time; and (2) performance of data visualization within bounded, well-understood memory limits. Galaxy delivers these capabilities via a multiserver framework, which provides each connected tenant access to Galaxy’s services for distributed data, communication, and computation; and Galaxy’s filtering, analysis, and visualization methods, which are designed to limit tenant resource requirements rather than requiring full use or duplication of the input data. Galaxy replaces traditional monolithic *in situ* architecture with a shared distributed data- and computation-space that is accessible to concurrent data-processing activities. These activities can be organized adaptively, in real-time, into distinct workflows that can access shared data at run-time to perform different science tasks concurrently. Using Galaxy, a simulation workflow consists of one or more stages “connected” by data exposed in the shared data-space. Workflow produce data products that are stored within Galaxy’s shared data-space. As activities attach to the Galaxy environment, they can access data products available within the data-space to generate additional data products and analysis results.

In the remainder of this chapter, we briefly discuss relevant related work not covered elsewhere in this book (Sect. 2). We then provide a high-level overview of Galaxy’s architecture, focusing on the novel multi-tenant and *in situ* capabilities (Sect. 3; a description of additional Galaxy functionality is presented in [1]). We then provide a deep-dive into the algorithms and structure of Galaxy’s ray tracer and direct volume renderer, a complex Galaxy analysis that can be used *in situ* with simulations (Sect. 4). Lastly, we provide examples of Galaxy’s performance on current-generation hardware architectures (Sect. 5).

2 Related Work

A substantial body of literature exists on in situ systems, as outlined in the introductory chapter of this book. We discuss four in situ frameworks below for their relevance to Galaxy, for their large user bases, their ready availability, and their deployment at many supercomputing centers around the world. We consider these four to be divided into two subsets—*closed systems*, that are specific to a analysis/visualization tool, and *middlewares*, which connect instrumented simulations with analysis/visualization components that have appropriate interfaces available.

ParaView Catalyst [3] and VisIt’s libsim [5] are closed systems that enable the use of pre-existing, well-developed visualization systems in situ. In each, the simulation is instrumented with an application-specific API that provides an interface to the visualization system. In general, both operate by incorporating portions of the visualization system’s run-time library into the simulation process to perform the visualization. Both ParaView and VisIt’s visualization engines utilize VTK [9], a class library of filters that leverage high-speed rasterization hardware (and software) by extracting renderable geometry from the input data. The resulting geometry is rendered with OpenGL and both systems use depth-compositing to combine the rendered results from many parallel processes into a single image for display/storage. In their most recent versions, both have incorporated ray tracing backends in addition to OpenGL; however, each still utilizes depth-compositing to create final images, which limits ray tracing effects to only process-local data.

Libsim is designed to interface a simulation with VisIt’s GUI. By connecting the two, libsim provides an interactive view into the running simulation using the full capabilities of the VisIt runtime in a directly-integrated, synchronous manner. The user can change the visualization as it is running via the VisIt GUI. Catalyst supports the development of visualization pipelines in a pre-process step, using the ParaView GUI and representative data, generating Python scripts that capture the pipeline to be deployed in situ. When the simulation is started, the Catalyst run-time loads the Python script and, when passed data from the simulation, performs the visualization in a directly interfaced, fully synchronous manner. We note that the Python script can use VTK’s serialized I/O capabilities to interface with an external visualization process. ParaView itself can then connect to the simulation using this export interface to provide an indirect, intermittent ability to interact with the simulation data as it is running.

ADIOS [6] and SENSEI [4] are middleware for in situ analysis that facilitate communication of data between source and destination activities. They are based on an externally reconfigurable I/O layer that connects the output of a source application with the inputs of the destination application. Both include an API used to instrument the simulation code to support a variety of I/O and analysis. Once the implementation costs have been paid to integrate the API within the simulation, each middleware simplifies the addition and substitution of analysis components. As middleware,

ADIOS and SENSEI do not provide analysis/visualization components. Instead, they provide an interface to other well developed systems, including ParaView and VisIt.

3 Galaxy Overview

Galaxy [1] is a parallel computation environment in which distributed parallel computation is performed by applications that intercommunicate (source code is freely available at <https://github.com/TACC/Galaxy/>). Like ADIOS and SENSEI, Galaxy serves as middleware that enables communication between activities (e.g., simulations and analysis/visualization components). However, unlike those systems, Galaxy’s distributed computation model is intended for use as an algorithmic framework. Two components are central to the Galaxy architecture: a *global name-space* into which data objects are published, and an asynchronous point-to-point and broadcast messaging framework for *work messages*. These components enable any worker to cause one or all of its fellow workers to perform a parameterized operation on data by sending a work packet containing an *action*, parameters, and the global names of sources and destinations. When a work packet is processed by its recipient, the *local* data objects associated with the global names of sources and destinations are identified, and the action performed. Actions can, in turn, create and transmit additional work packets requesting work be done elsewhere in the system.

As an example, a simple data parallel algorithm can be implemented in Galaxy by subclassing the `Work` class and defining (1) its action to be the per-local-data operation to be performed, (2) its members as the source and destination object names and, (3) whatever parameters are necessary for the algorithm at hand. This work code would be invoked by a driver procedure that is parameterized with the name of a pre-existing source object, the name of a (possibly preexisting) destination object, and the other parameters. If the destination object does not already exist, the driver would use the Galaxy API to create local objects on the same workers that contain partitions of the source object. The driver would then create an instance of the `Work` object subclass and, using the Galaxy asynchronous broadcast capability, request that each worker perform the requested action on its local source data and produce results in its local destination object.

Galaxy also supports global communication. For some algorithms, the driver may need to know that the data-parallel processes have all completed (recall that the work messages are fully asynchronous). Furthermore, the distributed destination object may need to acquire some global data (e.g., the data range). These examples require the workers to communicate status/results to the driver upon completion of their data-parallel work. For the data range example, the driver could create a data structure for a response object, initializing a count variable with the total number of data-parallel workers. It would then pass the response object’s address (in the driver’s memory space) to the workers. A second application-specific `Work` subclass would be defined with an action method that accesses the driver’s data structure, stores the associated

worker's data range, and decrements the driver's count of worker results accumulated thus far. After broadcasting the original work message, the driver would wait for the worker count to decrement zero, at which time it knows (a) that all data-parallel work is complete and, (b) it has the accumulated global data range information.

3.1 Multi-tenancy

Galaxy is intended for use as a multi-tenant, persistent co-processing environment. In the simple examples described thus far, a single driver causes single parallel algorithm to run. However, Galaxy supports multiple algorithms running concurrently, sharing the messaging system, the global object name-space, and computational resources. Consequently, many work messages can exist simultaneously, either in work queues waiting for the necessary resources or in a running state. As detailed later in the chapter, each worker supports multiple *worker threads* which process messages concurrently. This enables Galaxy to support direct, co-located, shared-memory co-processing.

The Galaxy `Multiserver` implements a socket-based client/server interface enabling multiple external user interfaces to asynchronously attach to a running Galaxy instance, install activity-specific capabilities (e.g., libraries containing subclasses of Galaxy's `Work` class etc.) on all the workers, and run concurrently as part of the Galaxy world. `Multiserver` clients communicate with the server via a simple string-based protocol. When run as a `Multiserver`, one process of the Galaxy spawns a thread that opens a socket and listens on it. When a `Multiserver` client attaches, a client-specific thread is spawned on the recipient Galaxy process to handle communications, and a message handler is installed for a few general purpose actions.

One *pre-installed* action enables the client to send the name of a shared library to Galaxy. Galaxy's server-side creates and broadcasts a `Work` message that causes every worker process in the Galaxy world to load the library (and its dependencies). A string-based *message handler* is installed, which the client-specific server-side thread retains to process subsequent application-specific exchanges. When subsequent messages arrive, the server side thread passes the message to each installed handler in turn until one recognizes the message, enabling multiple independent Galaxy activities to be installed by any client.

As a trivial example of multi-tenancy, Galaxy includes `msdata`, a `Multiserver` client that presents a command-line interface that enables the user to interact with the Galaxy data space. For example, the code below connects to the Galaxy instance and discovers that there are no visible data objects available.

An Empty Dataspace Listing in Galaxy

```
% msdata
? list;
datasets:
?
```

We can then start a second, possibly remote, msdata instance that imports a dataset into the Galaxy Dataspace:

Importing Data into a Galaxy Dataspace

```
\% msdata
? import { "Datasets": [
{
  "name": "eightBalls",
  "filename": "eightBalls.vol",
  "type": "Volume" } ]};
?
```

This import statement causes the Galaxy world to load a dataset from the file "eightballs.vol", of type "Volume", and name it "eightballs". The loaded dataset is now visible to both msdata instances:

A Populated Dataspace Listing in Galaxy

```
? list;
datasets:
eightballs
?
```

Two user interfaces have now connected to Galaxy and installed a shared-data capability in which they can see and access the same data space.

3.1.1 Interactive Rendering Using the Galaxy Multiserver

Galaxy includes a Multiserver client specifically for rendering datasets resident in the Galaxy data space. The *gxyviewer* application accepts a description of a rendering that includes a camera and a set of rendering operators. Rendering operators specify which datasets to render and how to render them using render-time visualization operations such as colormapping, isosurfacing and slicing. These operators are more fully discussed below.

As do all Multiserver clients, *gxyviewer* connects to the server via a socket and installs server-side code to implement a command-protocol between the client and server. In this case, *gxyviewer* implements two functions: one to transfer the initial visualization operators, and one to send camera updates to the server for rendering and to receive buffers of pixels in return. The client then creates an interactive display window, interprets user interaction events as camera transformations, and calls the server to render pixels that it then stores in its display window.

3.1.2 Non-Render-Time Visualization Algorithms

While *gxyviewer* enables the use of the Galaxy renderer directly to perform rendering-time operations (such as slicing, volume rendering, and isosurfacing), real world visualization problems may involve visualization algorithms that cannot be performed strictly at render-time. Such algorithms can themselves be implemented in the Galaxy framework; operating on Galaxy-resident data to create derived data objects.

One such algorithm is particle advection. To visualize flow fields, we implement a Runge-Kutta method to trace particles through a dataset describing a vector field. This implementation traces multiple particles (as many as there are seed points) in parallel through the partitions of the dataset, leaving partial traces in the Galaxy workers responsible for the partitions in which they pass, and using Galaxy asynchronous point-to-point transfers to move the head of particles that encounter partition boundaries to the process responsible for the neighboring partition.

3.1.3 The Galaxy GUI

Galaxy includes a user interface enabling users to build complex visualizations using a visual programming paradigm. This interface is implemented as a Multiserver client which installs a set of visualization algorithms, as described above, to create derived data using non-render-time algorithms and to orchestrate complex multi-faceted visualizations.

Figure 1 shows the Galaxy GUI in use. Datasets from an asteroid-strike simulation have been placed in the Galaxy data space by another client. The visualization shown accesses three of these datasets, describing pressure, density and the gradient of density. It uses a sampling algorithm to select seed points at a density level of interest, creating a temporary particles dataset. The particles and the original gradient vector

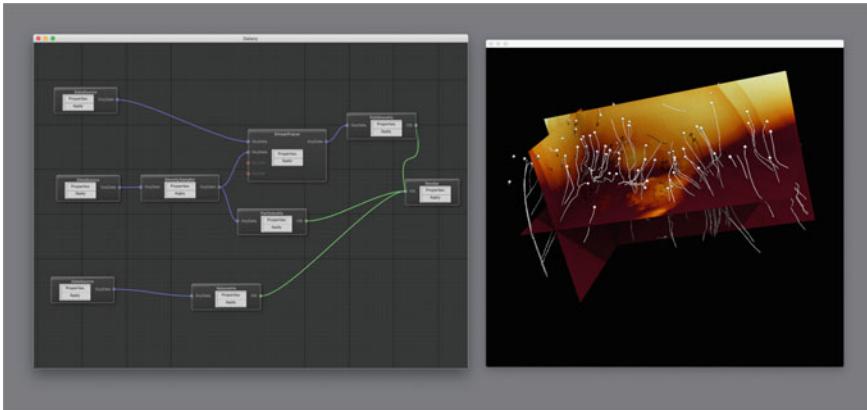


Fig. 1 Galaxy User Interface on the author's laptop using the Stampede2 supercomputer to visualize data

field are received by the stream tracer, which traces streamlines that represent the density gradient at the sample points. The particles and pathlines and the original volumetric pressure field are passed through filters that enable the user to control how they will be rendered - notably the radius of the spheres and stream tubes and, in the case of the pressure field, the location of slicing planes and color maps to apply. These results are then received by a Renderer filter, which owns an interactive rendering window and, as above with gxyviewer, calls Galaxy to render images.

3.2 Using Galaxy In Situ

Galaxy can perform *direct, co-located* *in situ* analysis of simulation data. A simple approach to enable this entails implementing a simulation as a Galaxy client application. In this manner, a simulation would run independently of other clients (such as the analysis client). The simulation client would publish one or more objects intended for access by other clients to a global dataset directory, and analyses can browse the global dataset directory to discover data for analysis. Galaxy incorporates an event architecture (based on the observer pattern) so that upstream publishing clients can signal that data objects (or other global capabilities) have been updated. Downstream clients, having registered interest in data objects, will then be notified when the objects have changed.

In order to work in this manner, preexisting simulations would be need to be instrumented to work in the Galaxy environment. While this code transformation may be non-trivial, we note that a client can receive its own private MPI communicator, enabling it to run largely unchanged, interacting with the Galaxy environment only when it explicitly needs to export published data.

Alternatively, Galaxy and a simulation can cooperate in an indirect manner by incorporating a communications client that presents an external parallel interface for separate distributed-parallel applications to attach to and to transmit data. In this approach the application is instrumented with an I/O API that negotiates and sends data to the communications client, and need not run inside the Galaxy computational framework.

We have implemented a simple example of this approach as the SocketConnector Galaxy client using a simple sockets-based protocol. This enables an external application (e.g., simulation) to connect to a Galaxy Multiserver instance, to initialize and publish data objects, and to periodically transfer time-step data to update the Galaxy-side data using parallel process-to-process communication. We note that sockets are simply a convenient communications mechanism that enable cooperation between distributed-parallel applications on the same or different servers; alternatives exist for special cases such as co-located processes using shared memory, or using a single MPI world with separate communicators to enable MPI point-to-point communications.

Galaxy enables different clients to communicate through a common Multiserver instance using events on data objects. Specifically, the server-side components of the DataSource filters on the left side of Fig. 1 attach *observers* on the data objects selected by the GUI user. When these data objects are updated (by a client interfacing with an external simulation), the observers are triggered, causing a message to be passed to the remote GUI client process. This then triggers the data-flow execution of the visualization and updates derived data objects and, ultimately, the rendered result in the GUI display window.

This is in fact what is happening in Fig. 1. A demonstration client (*simsim*, included in the Galaxy distribution) is given two timesteps as VTK .vti files and, according to command-line arguments, periodically interpolates time steps in between the two given input datasets and transfers the data to Galaxy using the Socket-Connector in situ interface.

4 The Galaxy Ray Tracing Engine

The original motivating problem behind the development of Galaxy was to investigate the use of ray tracing and ray tracing-based direct volume rendering (DVR) techniques for large scientific data applications. We present a deep discussion of this application here to highlight performance and implementation considerations when using Galaxy components for in situ analysis or other computation.

4.1 Performance in Space and Time

In visualization contexts, ray tracing and DVR offer significant advantages over geometry rasterization systems. Ray tracing and DVR techniques invert the basic loop of many visualization algorithms: where geometry-based systems iterate through the input data elements to determine how they contribute to the output image (at cost $O(N)$ time), ray-based systems iterate through the sample space, asking what input elements contribute to each ray, at cost $O(XYK)$ time, where XY reflects the resolution of the sample space, and K the cost of performing the algorithm along the ray.

In the important case of structured grids (and composite grids consisting of structured subgrids) interpolating the dataset at an arbitrary point is $O(1)$. When the grid is unstructured, a ray-based system will first create an acceleration structure (at cost $O(N \log N)$ time), which is amortized across all uses of the grid; once this is created, interpolation is of cost $O(\log N)$. In the following we demonstrate this advantage in several cases.

- Rather than extract a triangulated surface first, slicing planes can be rendered by a ray tracer by intersecting the ray with the equation of the plane, then interpolating the result.
- Volume rendering is implemented in a DVR system by iterating along the ray (at cost roughly $O(N^{\frac{1}{3}})$). Thus the cost of evaluating each ray is $K = O(N^{\frac{1}{3}} K')$ where $K' = O(1)$, for structured grids, and $O(\log N)$ for unstructured grids for a total cost of $O(XYN^{\frac{1}{3}})$ or $O(XYN^{\frac{1}{3}} \log N)$.
- Similarly, isosurfaces can be rendered by iterating along the ray looking for intervals that contain the isovalue. The performance analysis is the same as above, for volume rendering.
- Other algorithms that cannot be implemented as local operations along a ray can be rendered in a two-pass manner as in geometry-pipeline systems, substituting standard ray tracing in place of geometry-based rasterization. While this will necessitate the creation of an acceleration structure, we note that the cost of this will be $O(K \log K)$ where K is the size of the derived dataset, and K is likely much less than N . Once this acceleration structure is available, the rendering cost reverts to $O(XY \log K)$.

We note that current versions of both ParaView and VisIt incorporate ray tracing back-ends. However, they simply substitute ray tracing to perform geometry-based rasterization; each still relies fully on geometry pipelines and intermediate derived datasets.

4.2 Adapting Galaxy to Be a Sampling Engine

The Galaxy ray tracing engine can be thought of as an abstract engine for processing rays as they pass through data on a massively parallel system. In most cases, we want

to use this engine to render the data, as described above. However, we can use this engine for more abstract operations, including sampling. The simplest way to think about this is that the sampling engine sends rays through a dataset, and produces a new dataset of samples (particles).

A sampling operation is an abstraction of the following operations, which are normally done in the context of rendering the data:

- Intersection of ray with data. Because the engine processes intersections with data, we can write abstract operations evaluated instead of standard intersection operations. For example, we can use rays to find places in the data that have certain data values, gradients, or multi-variate properties.
- Action taken when an intersection occurs. In the case of sampling, we typically would like to take a sample of a data when we detect an intersection or a "hit". Thus, we can produce a dataset of samples, with each sample being taken when a ray intersects with something in the data.
- Ray propagation. When a ray creates a sample, we can decide to stop the ray, change its path based on data, or propagate other rays from that point in space. This gives the engine control over how to continue sampling the space.
This sampling capability is a powerful option for exploring properties of the data in a new way.

4.3 Ray Tracing Distributed Simulation Data

In the general case, ray tracing distributed data is difficult. Any arbitrary ray will encounter surfaces that may be distributed anywhere across the distributed system; therefore the *closest* intersection requires gathering information from each process that contains data that *might* intersect the data. An arbitrary region of space may contain surfaces that might be anywhere on the distributed system.

Ray tracing systems generally handle this issue by using a global acceleration structure, such as a bounding volume hierarchy (BVH). If the full dataset is not present on each node, the BVH will necessarily contain only bounding volumes of surfaces (rather than the surfaces themselves) and these bounding volumes will often overlap.

Tracing an arbitrary ray on a given process is done by testing the ray against the global BVH to identify data that *might* contain the next hit point. The ray is then tested against that data (either by transmitting the data to the process containing the data or by requesting the data from that process so that the test be performed locally). The results are then compared to find the actual closest hit points. Much work is being done to efficiently perform this task, notably including *speculative* ray tracing [7], in which each ray is traced against the data in each process.

Fortunately, simulation-based data offers us a simplifying fact: simulations are generally based on *spatially partitioned* data; each cooperating process is assigned a compact region of the computation space and, to the greatest extent possible, handles

its data independently of other partitions using minimal regions (ghost zones) of overlap with neighboring processes.

This means that anything that may affect a ray in a partition subspace is resident within the process responsible for the subspace. Importantly, this is also true of any data *derived* from a partition of data in a typical data-parallel process will also be resident on the node responsible for the partition, modulo an exchange of results near boundaries with the neighbors across the boundaries.

Given this property, the tracing of a ray with the computational space can be performed by breaking it into an *ordered set of non-overlapping intervals* based on its intersection with the partition boundaries, and that the intersection of the ray within each interval can be processed on the node responsible for the corresponding partition completely independently of data resident elsewhere on the system.

4.4 Ray Tracing with Asynchronous Work Messages

Based on the observations above, we have designed an algorithm for ray tracing/DVR in-situ simulation data with global illumination based on the asynchronous passing of rays from process to process as the ray traverses the partitioned computational space. This process is begun by an initial one-to-all CameraMessage that carries camera information to every node. Each recipient compares the camera information against its local partition to find exactly those rays that enter the computation space through their local partition. These are bundled onto RayPacketMessage objects and queued for processing locally.

When a process processes a RayPacket, it knows that the rays in that packet all begin in the local partition. The rays are processed against local data looking for *events* that cause (potentially temporary) termination of the ray (e.g., surface hit points or, importantly, the far, exiting surface of the partition) and each ray is ‘tagged’ to denote its terminating event type. When all the rays in the packet have terminated (as far as the local partition is concerned) the tracing process concludes. Once the local ray tracing of a packet of rays completes, the rays in the packet are examined to determine what occurred in the tracing process.

Rays that struck exiting boundaries of the partition are bundled into new RayPackets and sent to the nodes responsible for the neighboring partitionings. Rays that strike surfaces or acquire full opacity are then handled. Two new empty messages are initialized: a FrameBufferMessage, to carry results to a remotely hosted frame buffer, and a new RayPacket to hold any generated secondary rays.

If a processed ray is a *primary* ray that strikes a surface, we determine material properties, evaluate a lighting model, and place the result in the FrameBufferMessage. If the lighting model calls for shadows and/or ambient occlusion, secondary rays are generated and placed in the secondary-ray RayPacket (note that these secondary rays will necessarily begin in the current partition); *shadow* rays are tagged to terminate at light sources, and *ambient occlusion* rays are tagged to terminate at some specific distance.

If a processed ray is a *secondary* ray, its reason for terminating is examined. Secondary rays that terminate early indicate that light *was not* received at the hit point. Those secondary rays that don't terminate early represent light that *was* received at the hit point. We note here that we use a *subtractive* lighting model: primary rays are lit as if they are unshadowed and receive full ambient light, and shadows are *subtracted* from the pixel buffer if it is determined that the surface point is unlit. *Early terminated* secondary rays are appended to the FrameBufferMessage and the remainder are dropped.

4.5 Asynchronous Rendering in Galaxy

In the previous section we have outlined the basic distributed Galaxy ray tracing algorithm that moves rays between processes as the rays pass through partitions of the computation space. One process, responsible for accumulating the final image, receives FrameBufferMessages and adds the contents of each to the frame buffer. The image updates progressively over time, with primary Phong-lit contributions tending to come in first, and shadows and ambient occlusion contributions coming later. In an interactive session, this provides a form of incremental refinement. As the user moves the camera, new frames are initiated, causing the many rendering worker processes to spawn new primary rays, and also to ignore any further in-flight ray- or framebuffer-messages. When we need to know that an image has been completed (for example, to write a file), we need to know that all ray- and frame-buffer messages have been retired. In a distributed system this is non-trivial. Each ray- and framebuffer-message must be fully accounted for; a message sent from node to node is the responsibility of the sender until a receipt message is received from the recipient.

Galaxy uses a binary-tree message-based algorithm to verify that all messages have been retired. A subtree is determined to be idle if there are no ray- or framebuffer messages “alive” in the subtree. When a subtree enters or exits the idle state, it sends a notification message to its parent. These messages cause the parent to consider its own state. Its next state will be a combination of its children’s last received state and any live messages it is working on. Again, when its state changes, a notification message is passed upward. When the root node enters the idle state, it knows (a) that it itself does not own any live messages and (b) that when last informed, each child subtree was also idle. Unfortunately, strange conditions may occur. An exchange of message ownership from a leaf of one subtree to a leaf of the other may cause one subtree to enter the idle state and the other to exit it; however, the idle notification from one subtree may reach the root before the not-idle notification arrives from the other subtree. This necessitates a final downward pass through the tree when the root process enters the idle state. This is effectively a synchronization point of the process, though it may be performed either using low-level synchronous communication (e.g., MPI collectives) or asynchronous point-to-point messages.

4.6 Visualization Specification with Galaxy

The capabilities described above support much of what is required to do simple ray tracing: for each ray, find the closest surface hit point, determine appropriate pixel based on surface properties, materials and the lighting environment, and store the results into a frame buffer. Then, transfer sets of rays across processors to evaluate them with respect the global data of the scene.

However, for visualization, we want to do more than simply intersect rays against pre-existing surfaces; we want to do processing *along the ray*, including integration for volume rendering, the intersection of rays with planes that slice volumetric data, and the procedural visualization of isosurfaces. Thus Galaxy requires the camera, a lighting environment, a destination frame buffer, the objects to be displayed, and a specification for how to visualize each data object. The visualization of an object is, in fact, independent of the object (potentially, multiple visualizations might be simultaneously operating on the same object). For this reason, Galaxy visualizations do not annotate objects with parameters (isovalue, clipping planes, colormaps, etc.) but instead refer to constituent datasets indirectly, through a Galaxy object referred to as a `Vis` object, subclassed for the several currently-supported data types: volumes, surfaces, particle sets and pathline sets.

- **SurfaceVis** The simplest data object to render are surfaces. Surfaces consist of a triangle mesh with data values associated with vertices. The `SurfaceVis` object refers to a surface object and associates a color map with it.
- **ParticleVis** Particles are 3D points with data values associated. Particles are rendered in Galaxy as procedural spheres, where both the radius and color of each is determined by the data value. Thus, the `ParticleVis` also associates a colormap with a particles data object, but also includes a linear map of data to radius: (v_0, R_0) and (v_1, R_1) . Particles with values less than or equal to v_0 will be radius R_0 , values between v_0 and v_1 result in a linear interpolation between R_0 and R_1 , and remaining particles will have radius R_1 .
- **PathLineVis** Pathlines are sequences of 3D points with associated data values. Like Particles, Pathlines assign color via a colormap and determine stream-tube radius using a $(v_0, R_0) - (v_1, R_1)$ linear map.
- **VolumeVis** Volumes have the most degrees of freedom when it comes to visualization. A volume can be volume rendered; a `VolumeVis` therefore includes a boolean flag. If the flag is true, both a colormap and opacity map are required and the volume is volume rendered. If the volume is *not* being volume rendered it can be visualized as sets of isosurfaces and slicing planes. A `VolumeVis` object therefore also manages a set of isovalue and planar equations. These also require a colormap; all surfaces rendered using the same `Vis` object use the same colormap.

4.7 Galaxy Ray Processing

As described, RayList processing involves tracing rays as they pass through a region of space. This is implemented, at the lowest level, in ISPC [8]—an Intel parallel computing language that effectively vectorizes kernels over multiple rays simultaneously. This ray processing procedure receives a set of rays and a set of Vis objects that identify the objects to visualize and how to visualize them. It assumes that the rays begin inside the local subspace, either at an entry boundary surface, or fully inside it.

The first action intersects the rays with the exiting surfaces of the subspace. This determines the longest interval of the ray that *might* be traced locally. It then intersects the rays with the explicit surfaces (e.g., triangle sets) and procedurally defined implicit surfaces (e.g., particle spheres, pathline tubes, volume slicing planes) and if any are hit, updates the farthest point of the ray interval to the hit point. If any VolumeVis objects request isosurfaces or volume rendering, the processing iterates along the ray from entry to closest hit point, accruing volume contributions and searching for isosurface crossings along the way. If an integration interval traps an isovalue, a linear interpolation is done to approximate the hit point.

4.8 Galaxy and Cinema

Cinema [2] is a means by which limited interactive exploration of the dataset can be performed post-hoc by building a database of rendered images using visualization parameter sweeps (e.g. camera rotation, isovalue iteration) and combining the resulting images to mimic direct data exploration. Galaxy supports the efficient generation of Cinema databases by rendering many visualization settings simultaneously on each data timestep. Each of these visualizations is defined by rendering context; thus, each ray list, with its reference to a global rendering context, contains all the information necessary to process it. Since multiple rendering contexts (each including a reference to a destination frame buffer) can co-exist, it is possible to launch any number of visualizations simultaneously. Since Cinema databases very often contain many viewpoints, this serves to load-balance; rays will enter the computational space from many different sources, and will therefore engage different sets of worker processes. We present performance results for Cinema database generation in Sect. 5.

5 Galaxy Performance

This section presents Galaxy performance on representative datasets across two current hardware configurations. A deeper evaluation of Galaxy performance compared to Intel OSPRay is presented in [1].

We evaluate Galaxy’s performance using two currently-deployed advanced computing resources at the Texas Advanced Computing Center (TACC): Stampede2, with Intel Xeon Platinum 8160 “Skylake” processors and an Intel Omni-Path (OPA) 100-Gb/s fat-tree interconnect; and Hikari, with Intel Xeon E5-2690 v3 “Haswell-EP” processors and a Mellanox EDR 100-Gb/s fat-tree interconnect. Each Stampede2 Skylake node has 192 GB of RAM and two 24-core processors with two hardware hyperthreads per core (96 total threads). Each Hikari Haswell node has 64 GB of RAM and two 12-core processors with two hardware hyperthreads per core (48 total threads).

For each hardware configuration, we simulate tightly-coupled *in situ* analysis by rendering a camera orbit of 500 frames, each rendered at 1080p resolution. We render fifty frames simultaneously across the available hardware resources, with new frames initiated asynchronously as prior frames are completed. We perform these tests using three datasets: a volumetric simulation data of a deep water asteroid impact from Los Alamos National Laboratory ($1380 \times 840 \times 720$ grid of floats; 3.184 GB); a geometric isosurface extraction of a limestone karst core sample from Florida International University (5.8M vertices comprising 11.6M triangles; 0.266 GB); and an n-body dark matter “cosmic web” particle simulation from the Enzo team at National Center for Supercomputing Applications (1.07B particles; 12.2 GB).

As Fig. 2 shows, Galaxy achieves interactive rendering rates across node counts. We note increasing performance for the Skylake configuration as node count increases, though with sub-linear increase from 64 to 128 nodes likely due to an exhaustion of parallel work available. In the Haswell configuration, we attribute the relative performance degradation at higher node counts due to relative inefficiencies of the Mellanox interconnect versus the OPA interconnect: while both have the same

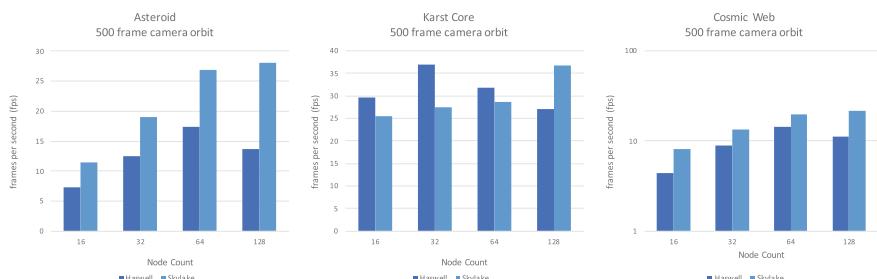


Fig. 2 Frames per second (fps) when rendering a 500 frame camera orbit for *Asteroid*, *Karst Core*, and *Cosmic Web* datasets (higher is better). Galaxy achieves interactive rendering for both Haswell and Skylake architectures

nominal bandwidth and latency, OPA can sustain higher messaging rates, particularly for the relatively small message sizes used by Galaxy, which enables the Skylakes on Stampede2 to sustain performance at higher node counts.

6 Conclusion

In this chapter, we have presented the Galaxy *Multiserver*, which provides multi-tenant distributed data filtering and analysis for a variety of in situ visualization scenarios. Galaxy can use ray tracing operations both for rendering and for data filtering, leveraging the same hardware-optimized components for multiple steps in a data analysis workflow. We have demonstrated that Galaxy provides interactive performance for distributed rendering tasks on current-generation hardware. Since Galaxy utilizes vendor-provided libraries for its internal tasks (currently the OSPRay and Embree libraries for Intel Xeon-optimized instructions), it can seamlessly benefit from vendor-provided improvements for current and future architectures. We plan to continue developing Galaxy’s interfaces, both for direct use and for integration into third-party applications that already support ray tracing, such as ParaView and VisIt.

Acknowledgements This work was funded in part by a DOE ASCR program award, Dr. Laura Biven, program manager; US NSF award ACI-1339863; and an Intel Graphics and Visualization Institute of eXcellence award.

References

1. Abram, G., Navrátil, P., Grossett, P., Rogers, D., Ahrens, J.: Galaxy: asynchronous ray tracing for large high-fidelity visualization. In: 2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV), pp. 72–76 (2018)
2. Ahrens, J., Jourdain, S., O’Leary, P., Patchett, J., Rogers, D., Petersen, M.: An image-based approach to extreme scale in situ visualization and analysis. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’14, pp. 424–434. IEEE Press, Piscataway, NJ, USA (2014). <https://doi.org/10.1109/SC.2014.40>
3. Ayachit, U., Bauer, A., Geveci, B., O’Leary, P., Moreland, K., Fabian, N., Mauldin, J.: Paraview catalyst: enabling in situ data analysis and visualization. In: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, pp. 25–29 (2015)
4. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: 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. IEEE (2016)
5. Kuhlen, T., Pajarola, R., Zhou, K.: Parallel in situ coupling of simulation with a fully featured visualization system. In: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV), vol. 10, pp. 101–109. Eurographics Association Aire-la-Ville, Switzerland (2011)
6. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible io and integration for scientific codes through the adaptable io system (adios). In: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments, pp. 15–24 (2008)

7. Park, H., Fussell, D., Navrátil, P.: Spray: Speculative ray scheduling for large data visualization. In: 2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV), pp. 77–86 (2018)
8. Pharr, M., Mark, W.R.: ispc: A SPMD compiler for high-performance CPU programming. In: 2012 Innovative Parallel Computing (InPar), pp. 1–13. IEEE (2012)
9. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit, 4th edn. Kitware (2006)

Proximity Portability and *in Transit*, M-to-N Data Partitioning and Movement in SENSEI



E. Wes Bethel, Burlen Loring, Utkarsh Ayachit, Earl P. N. Duque,
Nicola Ferrier, Joseph Insley, Junmin Gu, James Kress, Patrick O’Leary,
Dave Pugmire, Silvio Rizzi, David Thompson, Will Usher, Gunther H. Weber,
Brad Whitlock, Matthew Wolf, and Kesheng Wu

Abstract In high-performance parallel in situ processing, the term *in transit* processing refers to those configurations where data must move from a producer to a consumer that runs on separate resources. In the context of parallel and distributed computing on an HPC platform one of the central challenges is to determine a mapping of data from producer ranks to consumer ranks. This problem is complicated by the heterogeneity that arises in producer-consumer pairs, such as when producer and consumer codes have different levels of concurrency, different scaling characteristics, or different data models. The resulting mapping and movement of data from M producer to N consumer ranks can have a significant impact on aggregate application performance, particularly when the data consumer requires only a subset of the over-

E. W. Bethel (✉) · B. Loring · J. Gu · G. H. Weber · M. Wolf · K. Wu
Lawrence Berkeley National Laboratory, Berkeley, CA, USA
e-mail: ewbethel@lbl.gov

B. Loring
e-mail: loring@lbl.gov

J. Gu
e-mail: jgu@lbl.gov

G. H. Weber
e-mail: ghweber@lbl.gov

K. Wu
e-mail: kwu@lbl.gov

U. Ayachit · P. O’Leary · D. Thompson
Kitware, Inc., Clifton Park, NY, USA
e-mail: utkarsh.ayachit@kitware.com

P. O’Leary
e-mail: patrick.oleary@kitware.com

D. Thompson
e-mail: david.thompson@kitware.com

J. Kress · D. Pugmire
Oak Ridge National Laboratory, Oak Ridge, TN, USA
e-mail: james@jameskress.com

all data for its task. This chapter focuses on the design considerations that underlie SENSEI’s implementation to this challenging problem. These design considerations extend the core SENSEI architecture and include ideas like the need to accommodate flexibility in the choice of different partitioning methods, the ability for a data consumer to request and receive only the subset of data needed for its particular operation, and the ability to leverage any of several different data transport tools. The idea of *proximity portability*, being able to use different data transport methods as part of an *in transit* workflow, is illustrated through the use of three different transport layers where switching from one transport tool to another is accomplished with only a configuration file change. The chapter also includes a performance analysis summary showing the performance gains that are possible in terms of multiple metrics, such as memory footprint, time to solution, and amount of data moved, when using optimized partitioners in an *in transit* setting, gains that are made possible by the implementation shaped by specific design considerations.

1 Introduction and Overview

In the regime of in situ processing, one of many particular configurations is a scenario where data is moved across a network as it is produced to a different application, such as analysis or visualization, which runs on a separate set of hardware resources. In this scenario, data produced by M simulation ranks is then consumed by N consumer ranks, where typically $M \gg N$.

In this configuration, a central challenge is the problem of M-to-N data redistribution from producer to consumer ranks. Data redistribution refers to the process of

D. Pugmire
e-mail: pugmire@ornl.gov

E. P. N. Duque · B. Whitlock
Intelligent Light, Rutherford, NJ, USA
e-mail: epd@ilight.com

B. Whitlock
e-mail: bjw@ilight.com

N. Ferrier · J. Insley · S. Rizzi
Argonne National Laboratory, Lemont, IL, USA
e-mail: nferrier@anl.gov

J. Insley
e-mail: insley@anl.gov

S. Rizzi
e-mail: srizzi@anl.gov

W. Usher
University of Utah, USA and Intel Corp., Salt Lake City, UT, USA
e-mail: will@willusher.io

mapping data from M to N ranks, and also moving data from one place to another. One challenge is when producer and consumer run at different levels of concurrency resulting in no clear and obvious mapping from one to another. Second is when producer and consumer each use different data models, which would entail some level of data model reconciliation. Another is when producer and consumer have vastly different scaling characteristics, which in turn lead to different levels of concurrency M and N for a given producer-consumer pairing on a given problem configuration. Finally, it is often the case that the consumer ranks do not require the complete problem domain from the producer, a situation that can occur during data reduction or subsetting operations, such as slicing or isocontouring.

This chapter focuses on design, implementation, and performance analysis issues of a general purpose solution to the M-to-N data redistribution problem encountered in *in transit* processing scenarios. The design considerations (Sect. 2) include topics related to SENSEI’s adaptor architecture that focus on *in transit* scenarios, the central role of metadata, and how a partitioner uses metadata to compute a mapping from M-to-N ranks. These design principles allow SENSEI’s implementation of the M-to-N *in transit* data redistribution methods to achieve *proximity portability*, whereby SENSEI-instrumented codes can make use of one of several different tools, such as HDF5, libIS, and ADIOS, for moving data between *in transit* processing stages with only a configuration file change (Sect. 3). A summary of an in-depth performance study of SENSEI’s M-to-N *in transit* implementation at scale on a large HPC platform with multiple applications (Sect. 4) includes use of a full-scale physics code that uses adaptive mesh refinement (AMR), and analyzes performance across a number of metrics that include runtime performance, amount of data moved, time to solution, cost of solution, and memory footprint. The results show generality, broad applicability across a set of different data producers, data consumers, levels of concurrency, and varying partitioning algorithms.

2 Data and Execution Model Design Considerations for M-to-N, *In Transit* Processing

In transit processing is fundamentally different than *in situ* processing when considering how data is decomposed across the parallel ranks of producer and consumer. For example, consider an *in situ* configuration where a simulation code is running M -way parallel and invokes *in situ* methods that are also run, implicitly, at M -way concurrency. The simulation’s data decomposition dictates what data is processed by each of the *in situ* ranks: the *in situ* method’s data decomposition is imposed by the simulation’s data decomposition. In contrast, in an *in transit* scenario, while an M -way parallel simulation code uses one data decomposition, the N -way parallel analysis code may use a completely different data decomposition. The central challenge is to determine how to map from one data decomposition to another in the M-to-N setting, as well as to actually move the data from the M producer ranks to N consumer ranks.

This section presents several design considerations on different aspects of this complex problem: defining a mapping of data from M-to-N ranks, doing so in a way that can accommodate a variety of different producer/consumer code pairs, executing at varying concurrency, and using a number of different potential mechanisms for moving data.

The discussion begins with background material about SENSEI's endpoint and adaptor design patterns, which are foundational to the ability to swap in and out different endpoints without having to recompile the simulation, or data producer code. The focus here is primarily on topics related to *in transit* processing scenarios. Metadata, which is required to describe the producer's data model to the *in transit* consumer ranks (Sect. 2.3), is input to the partitioner (Sect. 2.4), which is responsible for computing a mapping of data from M producer to N consumer ranks. These elements are brought together to enable data movement (Sect. 3), which includes the idea of *proximity portability*, or the use of multiple data transport tools with the ability to switch between tools at runtime simply by changes to a configuration file.

2.1 Endpoint

The *in transit* configurations of interest are those consisting of multiple MPI parallel applications that run concurrently on HPC systems, where one parallel MPI job is a data producer and the other parallel MPI job is a data consumer. The term *endpoint* refers to these parallel applications that are compiled and linked with SENSEI and other related libraries and that consume and process data. In some circumstances, a data producer might be a data consumer, in which case it would be considered to be an endpoint as well.

Figure 1 shows an illustrative *in transit* example, with the endpoint shown as Fig. 1f. The challenge in this particular configuration is to find a partitioning from a simulation with 5 blocks of data distributed on 5 MPI ranks to the endpoint, which is running on 2 MPI ranks.

The endpoint is universal in the sense that it may be configured at run time to receive and process data from any SENSEI-instrumented simulation without modifications to either the endpoint or the simulation. This is achieved via XML configurations files. One type of configuration file specifies the methods and their parameters that will be used *in situ* or *in transit*; the other type of configuration file specifies the method of data transport.

2.2 Adaptor Pattern

SENSEI's design pattern consists of two actions: *invoke* and *fetch*. *In situ* processing and by extension, *in transit* processing, which includes analysis, visualization, and

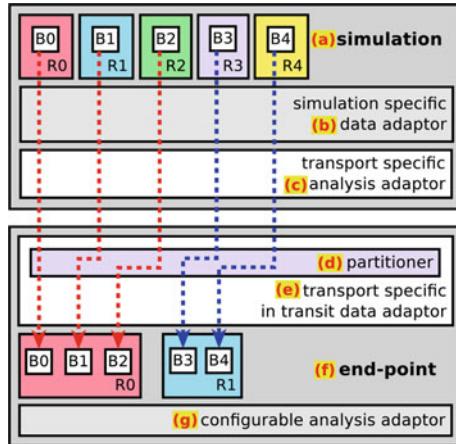


Fig. 1 SENSEI *in transit* architecture. A transport comprised of a pair of adaptors moves data from the simulation running on M MPI ranks to the endpoint running on N MPI ranks. Data is processed by any of the usual SENSEI analyses via its configurable analysis adaptor. The system is comprised of the following components: **a** simulation, **b** simulation specific data adaptor, **c** transport specific analysis adaptor, **d** partitioner, **e** transport specific data adaptor, **f** endpoint, **g** configurable analysis adaptor. As shown on the left, the simulation has 5 blocks distributed on 5 ranks. This image adapted from our previous work [15]

I/O, is periodically *invoked* by a simulation. In response to the *invoke* action, the data consumer code *fetches* data to process.

This design pattern is realized by two fundamental SENSEI adaptor types, the *analysis adaptor*, which has APIs used by the producer to *invoke* processing, and the *data adaptor*, which has API's to *fetch* the data to be processed. The adaptors define APIs that enable the invocation of a specific action without the need for the invoking code to know anything about the underlying implementation. The pattern allows for a change in the underlying implementation without the need to modify the code that invokes the adaptor methods.

The *data adaptor* fetches data in both *in situ* and *in transit* configurations. Any consumer of data, whether it be for I/O, visualization, or analysis, makes use of the data adaptor to fetch data. Every simulation needs to provide a *simulation specific data adaptor* (Fig. 1b) that is passed as part of invoking *in situ* processing.

Similarly, the fetch operations of every I/O library are exposed to the system via an *I/O library specific data adaptor* (Fig. 1e), which is passed as part of the invocation of processing on the endpoint. Through the use this adaptor design pattern, data processing codes that consume data need not be modified to run specifically in either an *in situ* or an *in transit* configuration. A detailed figure illustrating these relationships appears in other publications (cf. Loring et al. [15], Fig. 4).

To cope with the additional complexities of the *in transit* configuration, I/O library specific data adaptors are derived from the *in transit data adaptor* (Fig. 1e), which

is itself derived from the `DataAdaptor` type. The *in transit data adaptor* adds APIs for use by the endpoint for managing and controlling the connection and data movement from the simulation as well as APIs for interfacing with partitioning mechanisms (Fig. 1d and Sect. 2.4).

The *analysis adaptor* is used by the simulation to invoke processing for both in situ and *in transit* configurations. New analysis and I/O capabilities are exposed to simulators through the introduction of *transport specific analysis adaptors* (Fig. 1c). The role of transport specific analysis adaptors is to initialize and configure an in situ or I/O library for some user-specified processing or data movement. For example, in response to invocation by the simulation, the transport specific analysis adaptor will fetch and transform data, and then potentially provide that data to the I/O library for processing or movement to the endpoint.

The simulation initiates the invocation in both in situ and *in transit* configurations. In the *in transit* configuration, this invocation initiates a data movement phase where data is transferred to the endpoint for processing. In the endpoint, an I/O library specific *in transit* data adaptor (Fig. 1e) listens for the invocation, and forwards it into a library specific analysis adaptor.

The system implements dynamic configurability through a process of runtime delegation. Configurable adaptor implementations create and initialize a library-specific adaptor instance from a user-provided XML configuration file (cf. Fig. 1g) and SENSEI’s Configurable Analysis Adaptor.

Calls made to a configurable analysis adaptor are forwarded directly to the library-specific instance. Therefore, a simulation or endpoint instrumented to use the configurable analysis adaptor gains access to all available I/O and in situ libraries.

2.3 *Metadata*

In the context of in situ and *in transit* processing, the term *metadata* refers to “information about data” that is shared and exchanged between producer and consumer. This metadata includes values like the size and dimensionality of the mesh on the sender side. It also includes much more detailed information that spans four different categories, including: data sets, arrays, data blocks, and where appropriate, AMR-specific information.

The role of metadata is central to manage data in support of M-to-N redistribution for *in transit* use scenarios. It describes simulation data and its mapping onto the simulation’s M MPI ranks. Partitioners (Sect. 2.4) use the simulation metadata to compute the desired mapping of data onto the endpoint’s N MPI ranks. Data transports use metadata from multiple sources—the simulation and partitioner—to coordinate data movement between producer and consumer. The rich metadata object in SENSEI, which includes 34 different metadata values [15], enables transports to perform in-flight data transformations such as mapping one mesh onto another, and enables partitioners to implement a number of load balancing strategies.

2.4 Partitioner

Data partitioning in an M-to-N *in transit* scenario refers to the process of defining a mapping from an M -way parallel data producer to an N -way parallel data consumer. The process of partitioning may be straightforward, such as when $M == N$ in the case of traditional in situ processing, or it may be substantially more complex. Consider for example how some global parallel FFT implementations may require “pencil” or “slab” domain decomposition [17], or how two common approaches for parallelizing streamline computations—parallelize over blocks or parallelize over seeds—each require a different type of data partitioning [3] that may not be the same as the partitioning used by the simulation that produces the data. Given the diversity of potential ways data may need to be transformed and redistributed in *in transit* use cases, one of our design objectives is to make it straightforward to use any of a number of different potential partitioning methods.

One implementation characteristic that results from this design objective is that the partitioner is separate and distinct from both data producer and data consumer. The idea is to allow use of different partitioning methods with different combinations of producer and consumer. Similarly, the partitioner is separate and distinct from the data transport so that it is possible to use any of a number of potential implementations of *in transit* data movement tools (Sect. 3).

A later section that examines the performance of *in transit* M-to-N configurations at scale (Sect. 4) makes use of two types of partitioners: a *default* partitioner and an *optimized* partitioner. A default partitioner will move the entire dataset regardless of how much is needed by the consumer. The default partitioner simply invokes a default block-based equipartitioning algorithm over the entire simulation domain. In contrast, an *optimized partitioner* will provide only the subset of data actually needed by the consumer. An optimized partitioner uses the metadata provided by the simulation as well as the characteristics of the analysis or visualization operation to be performed to determine the minimum subset of data blocks needed for a particular operation.

By way of example for these two partitioner types, default and optimized, consider two use scenarios where only a subset of data is needed to complete the operation. One is a slice extract, which takes a 2D slice from a 3D volume, and another is an isosurface extract, which is computing an isosurface from a 3D volume. In the case of the slice extract, per-block bounding box metadata is tested for intersection with the slice plane. Only those blocks intersecting the plane are needed to compute the extract. In the case of the isosurface extract, per-block array range metadata are tested for intersection with the set of isocontouring values. Only those blocks where the data range brackets an isocontouring value are needed to compute the extract. Blocks not needed in the calculation are not assigned to any rank and as a result are not moved to nor processed by the consumer. Once the needed set of blocks are identified the block-based equipartitioning algorithm assigns them to available endpoint ranks.

3 Proximity Portability and SENSEI's Use of Multiple Data Transport Tools

One of SENSEI's design objectives is to enable a “write once, run everywhere” capability. In this approach, the key idea is the addition of SENSEI instrumentation code to a data producer, like a numerical simulation, then enables use of multiple potential in situ endpoints and tools without the need for any further instrumentation code changes on the data producer side. In other words, once a simulation is instrumented with SENSEI, selection of a variety of different in situ tools is accomplished by changes to a configuration file.

This concept of *tool portability* extends to the notion of *in transit* processing as well, where a SENSEI-instrumented data producer code can run either *in situ* or *in transit* without any instrumentation changes, and may also take advantage of a growing collection of tools that perform data movement, a concept referred to as *proximity portability*. This design objective is shown in Fig. 2, where an M -way parallel simulation produces data that is sent over one of several potential transports to an N -way parallel endpoint that performs visualization, analysis, or some other data-intensive operation. The subsections that follow present information about the SENSEI's *in transit* proximity portability through its use of three different mechanisms for moving data between producer and consumer: HDF5, libIS, and ADIOS.

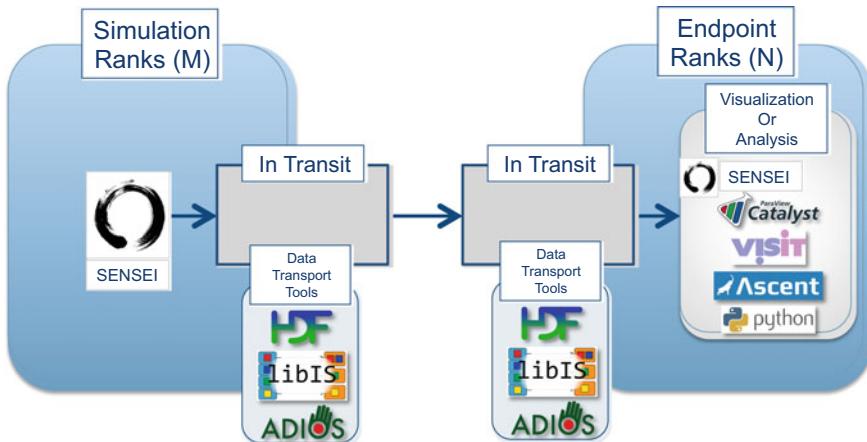


Fig. 2 An M -way parallel simulation sends data to an N -way parallel data consumer over one of several different potential data transport tools: HDF5, libIS, or ADIOS. Image courtesy W. Bethel and B. Loring

3.1 HDF5 In Transit Data Transport

HDF5 is a mature parallel I/O library and file format that is widely deployed and used on HPC systems around the world [8] by projects ranging from parallel I/O for some of the world’s largest computer codes, to providing a storage format for long-lived data from observations and experiments. Its prevalence as a parallel I/O library motivated recent work aimed at cultivating a better understanding of design and performance issues that would arise when leveraging HDF5 for *in transit* data staging and movement.

Gu et al. [10] studied use of HDF5 for *in transit* data movement and staging, and specifically with a configuration that uses NVRAM Burst Buffers (BB) presented as a filesystem on an HPC platform. The resulting design and implementation is an HDF5-based mechanism for *in transit* data transport that is accessible to SENSEI-instrumented codes. This data transport capability for M-to-N, *in transit* processing is accessible as one of several potential *in transit* transport mechanisms that can be selected through an XML-based configuration file.

Figure 3 shows a block diagram of the SENSEI-HDF5 transport mechanism, where HDF5 in this case uses NVRAM-based Burst Buffers (BB) for data staging. Following the design patterns used by other data transport mechanisms in SENSEI, the SENSEI-HDF5 transport mechanism consists of two related adaptors, the *Analysis Adaptor* and the *Data Adaptor*. The Analysis Adaptor implements the SENSEI interface for outputting data from data producers, and the Data Adaptor implements the input interface for consumers to ingest data. As of the time of this writing, SENSEI uses the VTK data model as a bridge between Analysis and Data adaptors. Therefore, in the SENSEI-HDF5 transport, the Analysis Adaptor receives VTK data, and stores to HDF5, while the Data Adaptor reads from HDF5 and returns VTK data for SENSEI.

The HDF5-based *in transit* data transport is able to leverage specialized hardware, such as BBs, for data staging. When the BB is presented as a filesystem to users, then the HDF5-based transport can make use of the BB when the filename in the HDF5-SENSEI configuration file points to a location on the BB-resident filesystem. One benefit of using BBs for *in transit* data transport is capacity: this staging mechanism may provide a larger maximum data footprint than is possible with conventional DRAM, memory-based methods. In other words, the total amount of BB storage on an HPC system may be significantly larger than the total distributed memory footprint. When using a BB presented as a filesystem one might expect its use for data staging would be slower than a purely memory-based approach for staging.

Recent studies [10] measure and compare the performance of the SENSEI-HDF5 *in transit* data transport over BB with one that uses a socket-based, memory-memory copy from one node to another. What is unexpected is that the BB configuration often completes the analysis use cases in less time than the socket-based transport mechanism as seen in both sets of test results reported in Fig. 4. When compared to using a traditional disk-based filesystem for staging, both socket- and BB-based approaches are significantly faster, as is visible in the left image of Fig. 4.

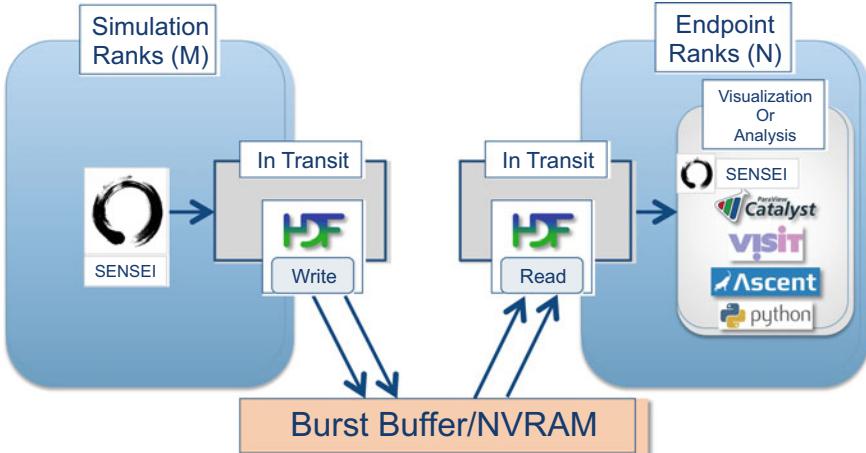


Fig. 3 The HDF5 transport layout. This particular illustration shows a configuration where Burst Buffers serve to stage data as it is moved from parallel producer to consumer. This image adapted from our previous work [10]

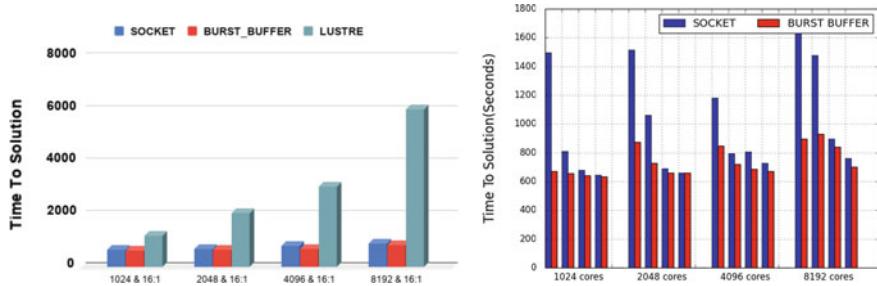


Fig. 4 Left: Time to solution for socket-based (ADIOS-Flexpath), NVRAM/Burst Buffer (HDF5), and disk (HDF5) approaches. Right: Time to solution for socket-based (ADIOS-Flexpath) and Burst Buffer (HDF5) approaches. These images are reprinted from our previous work [10]

The right of Fig. 4 shows only two data transport options: socket and BB, but with more test configurations than in the chart on the left. At the outset of this experiment, the expectation was that the socket-based option would run more quickly than the BB configuration because it uses memory-memory transfers. However, these particular experiments show the opposite: the BB configurations run more quickly than the memory-memory configuration; this result was a surprise. The most likely reason for this performance difference is that HPC systems architects and developers tend to optimize for file-based I/O, as opposed to socket-based operations. In this case, the NVRAM/BB, file-based approach is faster than a socket-based approach due to system software architecture and its optimizations for file-based I/O.

In addition to runtime as a performance measure, this same work also examines other metrics, such as memory consumption. One observation is that the memory

```

<sensei>
  <analysis type="hdf5"
    filename="/burst/buffer/file"
    method="stream" enabled="1" />
</sensei>

<sensei>
  <analysis type="adios1"
    filename=".//test"
    method="FLEXPATH" enabled="1" />
</sensei>

```

(a) Configuration file for specifying HDF5 as the data transport mechanism for *in transit* data movement. Here, the filename path points to the NVRAM/Burst Buffer.

(b) Configuration file for specifying ADIOS-Flexpath as the data transport for *in transit* data movement.

Fig. 5 Configuration files that show how to use HDF5 (left, a) or ADIOS-Flexpath (right, b). Changing from one transport to the other requires no coding changes, only a different configuration file

footprint requirements of the memory-based staging approach in these tests are significantly larger than the memory footprint of the HDF5/BB-based approach. On the simulation side for the memory-based staging, data is buffered in memory pools, the size of which grows as a function of how quickly the data can be moved to the *in transit* method. In contrast, the HDF5/BB approach does not exhibit these same memory requirements. See Gu et al. [10] for more details.

Switching between data transports is as simple as changing a configuration file. For the test configurations in these studies, Fig. 5 shows two configuration files: one for HDF5/BB, and the other for ADIOS-Flexpath. In the HDF5 configuration file, the pathname points to a location on the BB filesystem. In the ADIOS configuration file, the “method” of FLEXPATH requires a unique name, which in this case, is encoded into the `filename` field of the configuration file.

3.2 libIS In Transit Data Transport

libIS [19] is a lightweight library for *in transit* data transport. It uses a client-server model where clients (consumers) can request data from servers (producers) on an as-needed basis. libIS has been specifically designed for asynchronous, *in transit* analysis, where the simulation and analysis run decoupled from one another. The simulation-side library, coupled to the simulation either through our SENSEI interface or directly, exposes the simulation as a data server. The client-side library queries this server for new timesteps (Fig. 6). libIS supports the ability for clients to connect and disconnect over the course of a run, so viewers may be used sporadically, e.g., to check in on a long running simulation, or to begin monitoring after some event without requiring that additional nodes be set aside for the entire run.

libIS has been demonstrated at scale on Theta (Argonne National Laboratory) and Stampede 2 (Texas Advanced Supercomputing Center) supercomputers. Usher et al. [20] explored interactive *in transit* visualization of molecular dynamics simulations, the visual results of which appear in Fig. 7. They instrumented the LAMMPS molecular dynamics simulation code with SENSEI and libIS to move simulation

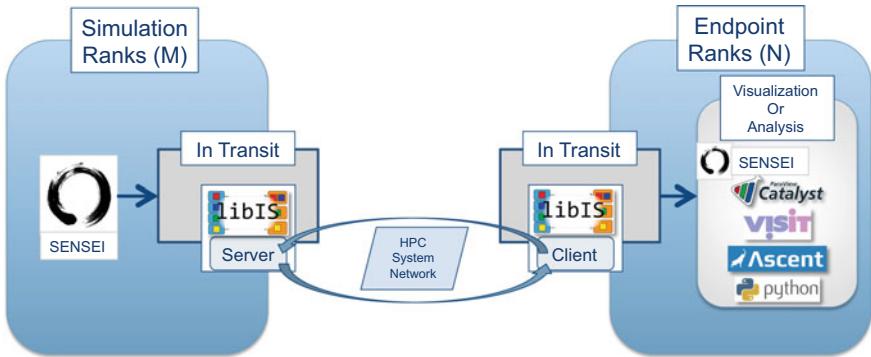
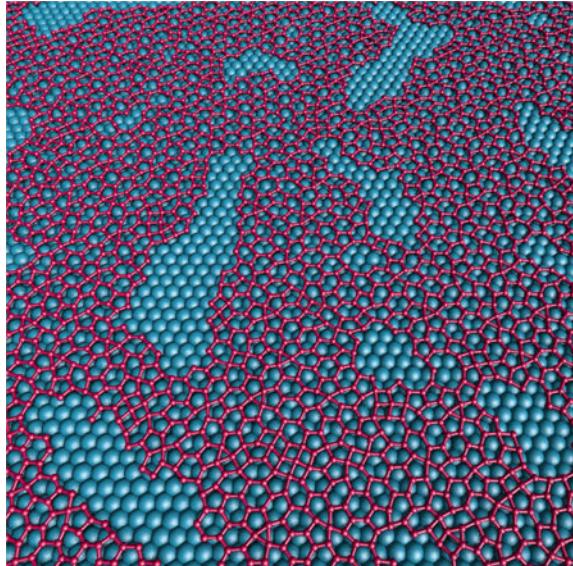


Fig. 6 libIS *in transit* architecture with SENSEI. Image courtesy S. Rizzi, N. Ferrier, and W. Bethel

Fig. 7 Interactive *in transit* visualization of a 172k atom simulation of silicene formation with 128 LAMMPS ranks sending to 16 OSPRay renderer ranks, all executed on Theta in the `mpi-multi` configuration. This image is reprinted from our previous work [20]



data over the network to a set of nodes running OSPRay, a high performance ray tracer for CPUs [21]. In this M-to-N configuration, the distributed-memory parallel renderer continues to query for data as the simulation runs, thus enabling the user to monitor the simulation state while it evolves over time. Along with rendering the data, the render side also supports running some local pipelines to process data, e.g., to compute bonds between atoms. The libIS performance study used a simulation input deck that is part of an active scientific research project at Argonne National Laboratory that performs atomic-level simulations of the formation of silicene [4].

3.3 ADIOS *In Transit* Data Transport

ADIOS is a parallel I/O library with a POSIX-like API [13]. The API requests to write/read (or put/get) are separated from the engines that perform the requested services, allowing for a variety of optimizations for functionality and performance, such as *in transit* implementations utilizing RDMA interconnect hardware when available. SENSEI has implementations based on the 1.X and 2.X versions of the ADIOS API. The ADIOS 2.X series has a complete redesign of the internals in order to better prepare for exascale computing needs [14], so some terminology changes between these release versions. In ADIOS 1.X, *in transit* processing through memory-based staging methods is provided by several transports: Flexpath [6], Dataspaces [7], and DIMES [22]. Similarly, in ADIOS 2.X there are different engines that provide both *in transit* and *in situ* solutions: SST, SSC, and Inline.

From a SENSEI perspective, these differences only show up as slight differences in the configuration file. A simple cartoon of this process and how ADIOS serves to connect them within SENSEI can be seen in Fig. 8. ADIOS supports data movement and I/O through a number of engines.¹ When using ADIOS, SENSEI selects and configures the ADIOS engine based on settings in user provided XML.

Figure 9 shows an example of utilizing the 2.X version of the ADIOS library with the SST transport for *in transit*. ADIOS provides both metadata queries and data selection methods. An ADIOS reader sees a global summary of all of the data available from all of the M writers without respect to who originally wrote it. By default, when moving data between a simulation and an analysis job, SENSEI applies an equipartitioning algorithm mapping from the simulation’s P data blocks decomposed on the simulation’s M MPI ranks to the N ranks of the analysis.

SENSEI’s data model incorporates lightweight metadata that describe both the simulation data available and its mapping onto the simulation’s MPI ranks. The availability of compact, cheap-to-move metadata in *in transit* applications facilitates load balancing and remeshing operations, and makes it possible to improve the performance of *in transit* execution through judicious data downselection as described in Sect. 4.

Among a number of useful features, such as easily switching between conventional disk based I/O and network based data movement, ADIOS also includes advanced features like dynamic disconnection and reconnection and a number of buffering controls. An example of the benefit of ADIOS buffering is shown in Fig. 10. Here, ADIOS hides the time spent by a slow analysis from the simulation by buffering data until the analysis can catch up. After each time step is buffered, control returns immediately to the simulation. The simulation completes in less than 400 s (label (c)) while the analysis continues well passed 1400 s. Were it not for ADIOS’s buffering scheme, the simulation would run substantially slower.

¹ Called transports in ADIOS 1.x.

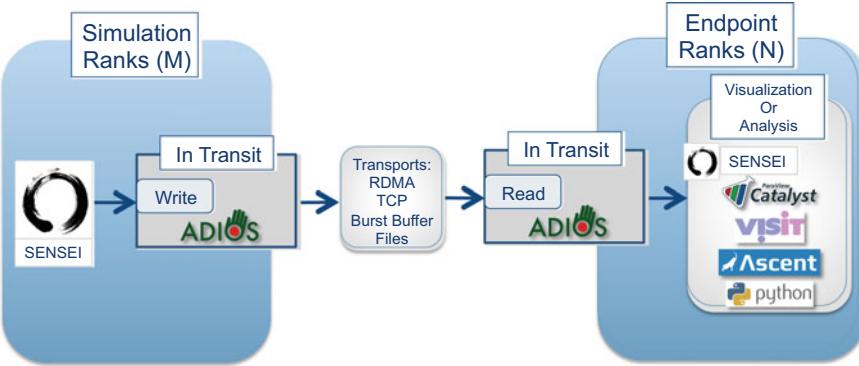


Fig. 8 The ADIOS transport layout in a SENSEI *in transit* use scenario. Image courtesy M. Wolf and W. Bethel

```
<sensei>
  <transport
    type="adios1"
    filename="data_iso_has.bp"
    method="FLEXPATH" />
</sensei>
```

```
<sensei>
  <transport type="adios2"
    filename="iso_has.sst"
    engine="sst" buffer_size="1">
  </transport>
</sensei>
```

(a) The XML file used with both the simulation and the endpoint to configure the ADIOS adaptor to use ADIOS 1.X API with the Flexpath transport.

(b) The XML file used with both the simulation and the endpoint to configure the ADIOS adaptor to use ADIOS 2.X API with the SST transport.

Fig. 9 SENSEI XML used to select and configure ADIOS

4 Performance Analysis of SENSEI's M-to-N *In Transit* Infrastructure

The main focus of the performance study summaries that follow is to quantify the performance gains that can result when leveraging metadata so that only the data needed to solve a particular problem is moved from the producer to consumer in realistic, at-scale M-to-N scenarios. These studies were conducted at scale on an HPC system using a benchmark miniapplication from the SENSEI distribution (Sect. 4.1), and also using a state-of-the-art production science code that uses adaptive mesh refinement configured for a Rayleigh-Taylor instability calculation (Sect. 4.2). These studies reveal the types of significant performance gains that are made possible when the data partitioning step can leverage metadata and consumer-side knowledge to make data requests from the producer. The complete studies appear in other publications [15].

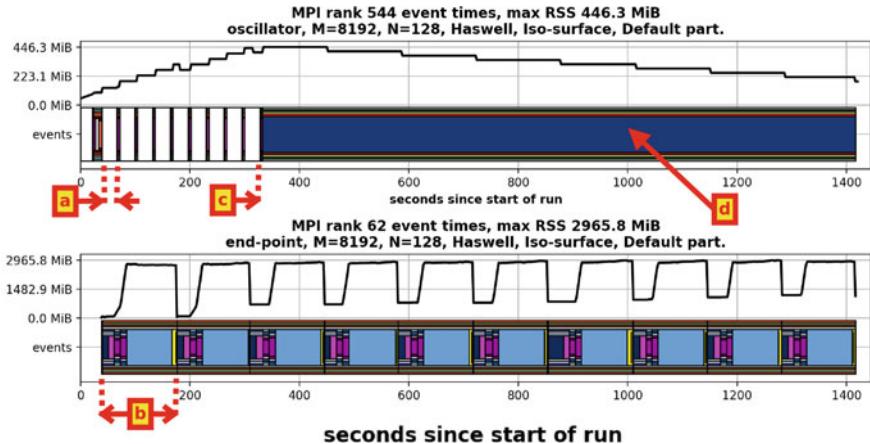


Fig. 10 ADIOS provided buffering hides the cost of a slow analysis from the simulation. Gantt charts from the simulation and analysis ranks with the largest resident set size (RSS) high water mark. Black line above the Gantt chart shows memory usage. Labels indicate: **a** time spent in the simulation computing one time step; **b** time spent in the analysis processing one time step; **c** simulation completes; and **d** ADIOS internally serves buffered data as fast as the analysis can consume it. Image courtesy B. Loring

4.1 Data Source: Oscillators Miniapplication

One of the design objectives for our M-to-N architecture and implementation is to make it possible to move only the data that is needed for a particular operation. This section summarizes an earlier study [15] that focuses on the cost savings that result when leveraging metadata to move only the relevant data needed to solve a problem.

The study focuses on two types of *in transit* operations where only a subset of data is required. One operation is a slice extract: in this case only those mesh cells or data blocks that intersect the slice plane are required to be moved to the *in transit* data consumer. The other operation is an isosurface extract: in this case, only those mesh cells that intersect the isosurface are required to be moved to the *in transit* data consumer. These operations are representative of two classes of methods that need only a subset of the original domain: one is based on some geometric constraints, in this case, a slice plane; the other is a data-dependent criteria, in this case, an isocontour.

The data producer is one of the SENSEI miniapplications, `oscillators`, which was run on NERSC's Cori Cray XC40 supercomputer. The code was configured to perform its computations on a 4096^3 mesh, which was decomposed evenly across 8192 MPI ranks. This study uses ADIOS 1.13.1 I/O library with the Flexpath staging method as the *in transit* data transport mechanism.

For each of the two extract operations, the study examines four different performance measures under varying levels of endpoint concurrency when using default and optimized partitioners. The battery of tests aims to better understand a broad set

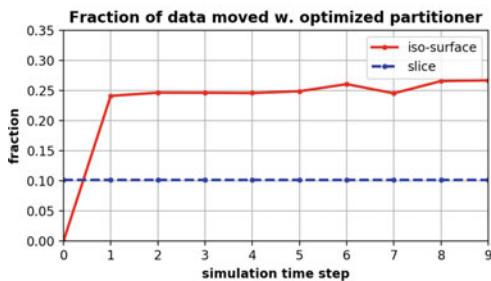
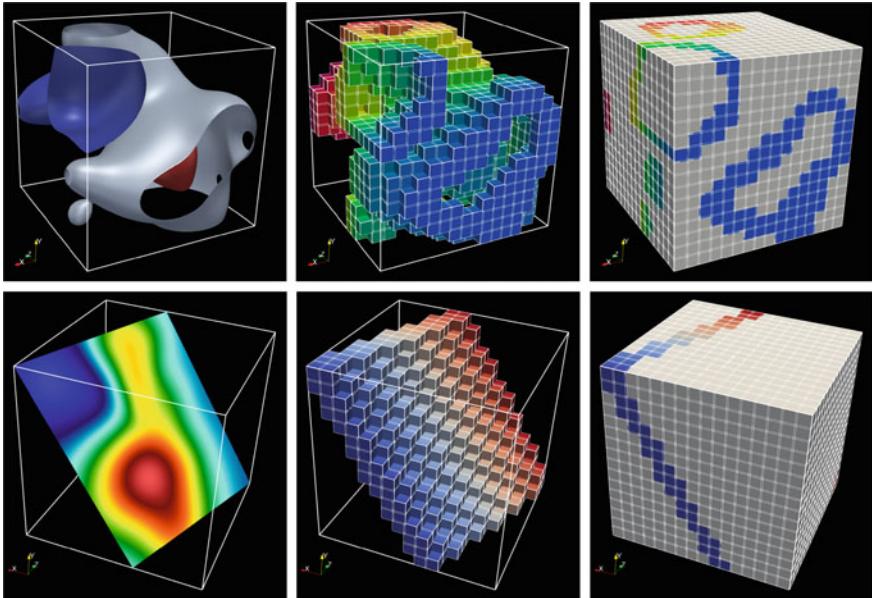


Fig. 11 Optimized partitioners determine which blocks are needed by the consumer and assign them to endpoint ranks while excluding all other blocks from the dataset. Top row: isosurface extract. Middle row: planar slice extract. Left column: extracted geometry. Center column: blocks needed to compute the extract. Right column: all the blocks. Line plot: the fraction of simulation cells moved as a function of simulation time. This image is reprinted from our previous work [15]

of performance metrics and at varying levels of concurrency. The slice plane value and three isocontour levels are intended to result in non-trivial extracts, which are shown in the top portion of Fig. 11. The left column shows the extracted geometry, the middle column shows the set of blocks that were used in the calculation, and the right columns shows all the blocks. When the optimized partitioner is in use, the white colored blocks were not moved from the simulation to the endpoint nor processed by the endpoint.

In the bottom portion of Fig. 11, the chart shows, at varying simulation time steps, the fraction of total data moved from producer to consumer in each of the optimized

partitioner configurations. For the three isosurfaces computed, about 25% of the data contributes to a solution, and only this portion of data needs to be moved. An exception is at the first timestep, where the simulation has not evolved the computation to the point where the output has any cells that contain the isovalue. In the case of the slice, about 10% of the mesh cells intersect the slice plane and contribute to the final solution.

The original study goes into much greater depth examining configurations using default and optimized partitioners, with multiple endpoints, and measuring multiple metrics at varying concurrency. Those metrics include the amount of data moved from producer to consumer in bytes, the maximum memory footprint across all endpoint ranks, time to solution (which is akin to runtime), and cost of solution in terms of CPU hours.

Two of the lessons learned from the study are that different ratios of M -to- N produce vastly different results in terms of performance, and that problem specific factors can influence the choices for M and N . For example, when comparing the isosurface and slice endpoints, many more data blocks intersect the isocontour than the slice plane, about 25 versus 10%, as shown in Fig. 11. As a result, when increasing the concurrency level of an endpoint, one encounters diminishing returns at different concurrencies, depending on the nature of the problem. There is no “one-size-fits-all” ratio that works best in all settings.

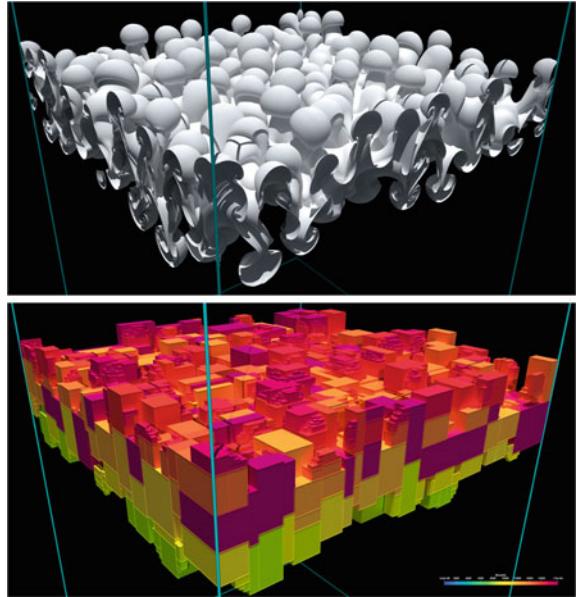
4.2 Data Source: AMReX-Based IAMR Code

The summary of an earlier study [15] continues with a focus on examining the potential cost savings that might result when using an optimized rather than a default partitioner but with a full-scale scientific simulation rather than a miniapplication. The authors instrumented the AMReX framework [23] for use with SENSEI. That gave access to a wide variety of block-structured adaptive mesh refinement (AMR) simulations for testing. Those experiments made use of the AMReX-based IAMR compressible Navier-Stokes code [1] configured for the simulation of a Rayleigh-Taylor instability modeling the mixing of two fluids of different densities under the influence of gravity. The Rayleigh-Taylor instability produces a set complex isosurfaces that evolve in time.

The runs consisted of IAMR configured for a base level of $1024^2 \times 2048$ cells, one level of refinement, and executed on $M = 8192$ ranks, with 4 OpenMP cores per rank, on 1025 KNL nodes of NERSC’s Cori system. The total number of cores used by IAMR was 32768. The endpoint was run on 9 nodes with $N = 128$ MPI ranks. This ratio of M to N was chosen based upon the results of the miniapplication study in Sect. 4.1, which showed better performance gains for the optimized partitioner when $M \gg N$.

One challenge in processing AMR data is that data blocks from refined levels duplicate and cover, either partially or fully, data blocks from coarser levels. Care must be taken when computing metadata and applying partitioning algorithms. When

Fig. 12 Top: Isosurface extracted *in transit* from the AMR mesh produced by the IAMR code. Middle: Blocks, colored by block identifier, from the AMR mesh that contain the isosurface. These images are reprinted from our previous work [15]

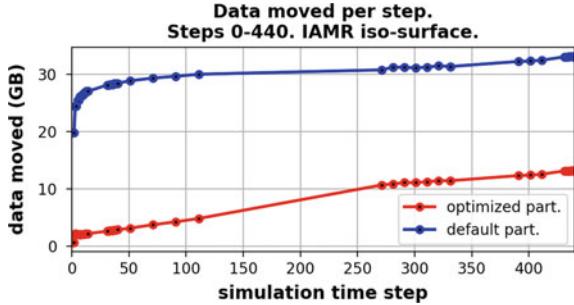


computing the per-block array minimum and maximum data values to determine if an isosurface intersects a block, care must be taken to not make use of data from the cells of blocks that are covered by cells from a block in a more refined level. The reason is that covered cells are duplicated in the refined level and hence the isosurface will be duplicated as well. The AMReX-specific data adaptor handles this aspect of the metadata calculation. As a result, the optimized isosurface partitioner can run without any modification for AMR-based meshes. The ability to handle complex dataset types such as AMR meshes illustrates the flexibility of SENSEI M-to-N *in transit* design and implementation.

The isosurface extracted at time step 420 is shown in Fig. 12 in the top panel, along with the 4771 level-1 blocks that intersect this isosurface in the bottom panel. Figure 13 shows the amount of the data moved at each time step for isosurface extraction with the optimized partitioner (red line) compared to the total data size (blue line). In the worst case, the optimized partitioner moves no more than 40% of the data.

The original study goes into more depth, including additional performance measures like maximum memory footprint, time to solution, and cost of solution in terms of CPU hours [15]. Collectively, the studies described here and in Sect. 4.1 show the flexibility and performance gains that can result when leveraging rich metadata to solve a complex M-to-N data mapping problem.

Fig. 13 The amount of data moved during runs of the IAMR Rayleigh-Taylor problem using the optimized partitioner (red line) is substantially less than when using the default partitioner (blue line). This image is reprinted from our previous work [15]



5 Related Work

The idea of processing data as it is generated has been around for decades, with some of the earliest work consisting of a direct-to-film recording process from the 1960s. That work, along with a thorough survey of work in the *in situ* and *in transit* space is in a 2016 Eurographics STAR report [2]. Early work on *in transit* infrastructure in the HPC space includes the CUMULVS project, which is middleware for coupling codes running at different levels of concurrency and for moving data between them in a M-to-N fashion [9]. A more recent CUMULVS report from [11] includes a survey of related projects focusing on M-to-N data partitioning and distribution on HPC platforms, some of which go back to the mid 1990s.

Over the years, several efforts have studied whether an *in situ* or *in transit* configuration will produce lowest cost for a given problem configuration typically measured in runtime or time-to-solution. Oldfield et al. [18] evaluate *post hoc*, *in situ*, and *in transit* in the context of analysis and tracking of features in simulation output. They identify situations in which *in transit* or *in situ* approaches are more or less advantageous, such as *in transit* being advantageous when analysis computations are more complex and time consuming. Morozov and Lukić [16] examine *in situ* and *in transit* configurations of a cosmological simulation coupled with a two-stage analysis pipeline and find it advantageous to use *in transit* configurations when the analysis and simulation codes have different scaling properties. Kress et al. [12] study scalable rendering and aim to find the best balance of M to N producer and consumer ranks across different levels of concurrency by considering cost models that measure time-to-solution for both *in situ* and *in transit* configurations.

The focus of our work here is on the design, implementation, and performance evaluation of a method for performing robust, flexible, and general purpose M-to-N data redistribution for use in an *in transit* setting at scale on HPC platforms. In particular, we are interested in understanding the performance gains that can result when leveraging metadata for partitioning and moving data from producer to consumer ranks in an *in transit* configuration. A similar idea appears in Childs et al. [5], which describes the “contract” system in the VisIt application. That capability results in optimizations of data movement through the visualization pipeline: downstream

processing stages inform upstream stages of the data subsets needed to perform a specific computation.

Our performance study uses some of the same use scenarios, namely planar slicing and isocontouring, to illustrate the gains that result from moving only the data the consumer needs to perform its computation. Whereas the Childs et al. project measured and reported runtime improvement in the setting of an interactive GUI based post processing visualization application where the stages of their pipelines ran in the same process address space and data was provided by disk based I/O, our work here investigates a similar approach but applied to M-to-N *in transit* processing and that is truly distributed-memory parallel along the direction of the pipeline. We look beyond runtime to examine deeper levels of performance analysis and consider the impact of heterogeneity in concurrency that occurs in M-to-N processing. For each of the default and optimized partitioner configurations, we measure and report, in addition to runtime, the amount of data moved between producer and consumer ranks, and the memory footprint of producer and consumer ranks. These additional measurements and at varying concurrency levels for M and N provide significantly deeper insight into the benefit of the optimizations for *in transit* data partitioning and placement.

6 Conclusion and Future Work

In *in transit* processing, one of the central challenges is determining a mapping of data from M producer ranks to N consumer ranks. This chapter has presented design considerations and design patterns for a flexible, general purpose solution to this challenging problem. An in-depth performance study consisting of 32K-way parallel runs of a production scientific simulation code, AMReX/IAMR, on a large HPC platform demonstrates its usefulness and also reveals the performance gains that result when leveraging the system characteristics to send only the data needed by an *in transit* consumer to solve its particular problem. The performance evaluation measures runtime, amount of data moved, and time to solution to help reveal the nature of performance gains possible when using an optimized partitioner, which moves only those portions of the data needed by the consumer.

A central theme of the design considerations is the idea of *proximity portability*, which means having the ability to run either *in situ* or *in transit* without any code modifications, and if run *in transit*, to be able to leverage any number of different potential tools for moving data between producer and consumer ranks. SENSEI's adaptor design pattern makes this possible, and this work shows use of three different mechanisms for moving data between *in transit* producers and consumers: HDF5, libIS, and ADIOS. This capability opens new avenues of research that leverage the advantages offered by an *in transit* approach, namely being able to overlap computation and communication, and being able to better load balance between M simulation ranks and N consumer ranks. We anticipate an explosive growth in applications of methods for doing learning, analysis, and cooperative computing when

science code teams are easily able to couple codes in a flexible way as shown by the design principles, examples, and performance studies we have presented in this chapter.

Acknowledgements This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract Nos. DE-AC02-05CH11231 and DE-AC01-06CH11357, through the grant “Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery,” program manager Dr. Laura Biven. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Argonne National Laboratory’s work was supported by and used the resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy, Office of Science User Facility supported under contract DE-AC02-06CH11357.

References

1. Almgren, A.S., Bell, J.B., Colella, P., Howell, L.H., Welcome, M.L.: A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *J. Comput. Phys.* **142**(1), 1–46 (1998)
2. Bauer, A.C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., Moreland, K., O’Leary, P., Vishwanath, V., Whitlock, B., Bethel, E.W.: In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms, a State-of-the-art (STAR) Report. Computer Graphics Forum (Special Issue: Proceedings of EuroVis 2016) vol. 35, issue 3 (2016). LBNL-1005709
3. Camp, D., Garth, C., Childs, H., Pugmire, D., Joy, K.I.: Streamline integration using MPI-hybrid parallelism on a large multicore architecture. *IEEE Trans. Vis. Comput. Graph.* **17**(11), 1702–1713 (2011). <http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.259>
4. Cherukara, M.J., Narayanan, B., Chan, H., Sankaranarayanan, S.K.R.S.: Silicene growth through island migration and coalescence. *Nanoscale* **9**, 10186–10192 (2017)
5. Childs, H., Brugger, E., Bonnell, K.S., Meredith, J.S., Miller, M.C., Whitlock, B., Max, N.L.: A Contract-Based System for Large Data Visualization. In: Proceedings of IEEE Visualization (Vis05). Minneapolis, MN (2005)
6. Dayal, J., et al.: Flexpath: Type-based publish/subscribe system for large-scale science analytics. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 246–255. IEEE (2014)
7. Docan, C., Parashar, M., Klasky, S.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Clust. Comput.* **15**(2), 163–181 (2012)
8. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the HDF5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, pp. 36–47. ACM (2011). Software at <http://www.hdfgroup.org/HDF5/>
9. Geist, G.A., Kohl, J.A., Papadopoulos, P.M.: CUMULVS: providing fault-tolerance, visualization and steering of parallel applications. *Int. J. High Perform. Comput. Appl.* **11**(3), 224–236 (1997)
10. Gu, J., Loring, B., Wu, K., Bethel, E.W.: HDF5 as a vehicle for in transit data movement. In: Proceedings of the SC19 Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV ’19, pp. 39–43. Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3364228.3364237>
11. Kohl, J.A., Wilde, T., Bernholdt, D.E.: Cumulvs: interacting with high-performance scientific simulations, for visualization, steering and fault tolerance. *Int. J. High Perform. Comput. Appl.* **20**(2), 255–285 (2006)

12. Kress, J., et al.: Comparing the efficiency of in situ visualization paradigms at scale. In: International Conference on High Performance Computing, pp. 99–117. Springer (2019)
13. Liu, Q., et al.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurr. Comput.: Pract. Exp.* **26**(7), 1453–1473 (2014)
14. Logan, J., Ainsworth, M., Atkins, C., Chen, J., Choi, J.Y., Gu, J., Kress, J.M., Eisenhauer, G., Geveci, B., Godoy, W., et al.: Extending the publish/subscribe abstraction for high-performance i/o and data management at extreme scale. *Bull. Tech. Comm. Data Eng.* **43**(1) (2020)
15. Loring, B., Gu, J., Ferrier, N., Rizzi, S., Shudler, S., Kress, J., Logan, J., Wolf, M., Bethel, E.W.: Improving performance of m-to-n processing and data redistribution in in transit analysis and visualization. In: EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV). Norrköping, Sweden (2020)
16. Morozov, D., Lukic, Z.: Master of puppets: Cooperative multitasking for in situ processing. In: Proceedings of the Symposium on High-Performance Parallel and Distributed Computing (HPDC), pp. 285–288 (2016)
17. Mortensen, M., Dalcin, L., Keyes, D.: mpi4py-fft: Parallel fast fourier transforms with mpi for python. *J. Open Source Softw.* **4**, 1340 (2019)
18. Oldfield, R.A., et al.: Evaluation of methods to integrate analysis into a large-scale shock shock physics code. In: Proceedings of the 28th ACM International Conference on Supercomputing, ICS ’14, pp. 83–92 (2014)
19. Usher, W., Rizzi, S., Wald, I., Amstutz, J., Insley, J., Vishwanath, V., Ferrier, N., Papka, M.E., Pascucci, V.: Libis: A lightweight library for flexible in transit visualization. In: Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV ’18, pp. 33–38. Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3281464.3281466>
20. Usher, W., Rizzi, S., Wald, I., Amstutz, J., Insley, J., Vishwanath, V., Ferrier, N., Papka, M.E., Pascucci, V.: libis: a lightweight library for flexible in transit visualization. In: 2018 Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV) (2018)
21. Wald, I., Johnson, G.P., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navrátil, P.: Ospray-a CPU ray tracing framework for scientific visualization. *IEEE Trans. Vis. Comput. Graph.* **23**(1), 931–940 (2016)
22. Zhang, F., et al.: In-memory staging and data-centric task placement for coupled scientific simulation workflows. *Concurr. Comput.: Pract. Exp.* **29**(12), e4147 (2017)
23. Zhang, W., Almgren, A., Beckner, V., Bell, J., Blaschke, J., Chan, C., Day, M., Friesen, B., Gott, K., Graves, D., Katz, M.P., Myers, A., Nguyen, T., Nonaka, A., Rosso, M., Williams, S., Zingale, M.: Amrex: a framework for block-structured adaptive mesh refinement. *J. Open Source Softw.* **4**(37), 1370 (2019). <https://doi.org/10.21105/joss.01370>