Big Data From Scientific Simulations

John Edwards, Sidharth Kumar, Valerio Pascucci[1]

University of Utah

Abstract. Scientific simulations often generate massive amounts of data used for debugging, restarts, and scientific analysis and discovery. Challenges that practitioners face using these types of big data are unique. Of primary importance is speed of writing data during a simulation, but this need for fast I/O is at odds with other priorities, such as data access time for visualization and analysis, efficient storage, and portability across a variety of supercomputer topologies, configurations, file systems, and storage devices. The computational power of high-performance computing systems continues to increase according to Moore's law, but the same is not true for I/O subsystems, creating a performance gap between computation and I/O. This chapter explores these issues, as well as possible optimization strategies, the use of in situ analytics, and a case study using the PIDX I/O library in a typical simulation.

Keywords. Big data, data storage, data visualization and analysis, parallel I/O

Introduction

Many scientific questions involve complex interactions between physical entities in complicated domains. Biology researchers might want to understand airflow patterns in mammalian lungs; car manufacturers need to understand how a vehicle will respond in different crash scenarios; climatologists predict the path and intensity of hurricanes; movie makers save money by rendering simulations of stormy seas. These, and many other important questions and tasks, involve complexities that can only rarely be solved using analytic, closed-form methods.

A simulation takes a number of inputs: the most important are equations and rules governing behaviors and interactions. The equations are most often in the form of partial differential equations, and rules range from collision detection codes to electron exchange mechanisms. Other inputs are the initial conditions, or the state of the system at the beginning of the simulation, resolution, and number of time steps to take.

Spatial resolution deals with how finely we want the state represented at each time step, and temporal resolution is the length of time step. For example, a video game may run a realtime simulation of an exploding building. As the explosion must be simulated and rendered in realtime, and since it will be running

[1]Corresponding author: pascucci@sci.utah.edu

on commodity gaming hardware, the resolution is likely to be very low. That is, most fine details may be represented using other techniques (e.g. billboarding) while the simulation results are used only for the overall behavior. On the other hand, a simulation of a combustion engine requires extreme detail in order to design the engine for even small efficiency gains. Generally spatial and temporal resolution go hand-in-hand. But regardless of how resolution is balanced among the dimensions, the net effect on the data is the same: as resolution increases, the amount of data produced from the simulation increases. For example, if we store the entire system state at every time step, then halving the time step doubles the amount of data. If the spatial resolution is doubled in each dimension in a 3D simulation, then the amount of data jumps by a factor of 2^3.

Scientific simulations present particularly interesting challenges in terms of data. It is extremely rare for a scientific simulation to run in realtime. As a result, simulations can be run at high resolutions, requiring days, weeks, and even months to complete. Further, often these simulations are not feasible on desktop hardware and require a computing cluster. Clusters present an additional level of complexity since the data, at simulation time, is distributed among compute nodes. Transferring the results to a storage medium, such as a hard drive is, in general, a slow process. While computation speed is increasing as described by Moore's law, I/O subsystem speeds are not improving nearly as fast, so the gap between computation and I/O speeds continues to widen [1].

I/O library performance is affected by characteristics of the target machine, including network topology, memory, processors, file systems, and storage hardware. Performance is also affected by data characteristics, file formats, and tunable algorithmic parameters. Because the data may be extremely large, gigabytes and even terabytes *at each timestep*, the practitioner may choose to save the state of the system, or "dump the data," infrequently in order to avoid slowing the simulation down with costly I/O, as well as to save on storage space. In a simulation that dumps every n timesteps, we call n the "output interval".

It is important to understand what simulation data is used for. There are three general uses: scientific insight, restart, and debugging.

Scientific insight The most obvious use of simulation data is to answer the scientific question at hand. For example, what did the universe look like 2 billion years ago, and what will it look like 2 billion years from now [2]? This question can be answered by running a large-scale simulation to completion and visualizing the results. However, to use only this result would be short-sighted. An additional answer that we can obtain essentially for free is "how could the universe evolve from now until 2 billion years in the future?" To answer this question, we simply dump state data as the simulation is running. The important concept here is that simulations not only answer questions, but they lead to additional questions to answer through visualization and analysis.

Restart For any of a number of reasons, a simulation may fail prematurely. Maybe the code has a bug, or the supercomputer time allocation ran out, or a power failure occurred. Whatever the reason, the scientist probably will not want to restart the simulation from the beginning. Most practitioners schedule frequent data dumps so that they can start where they left off if the simulation fails to complete. This is called checkpointing and restart.

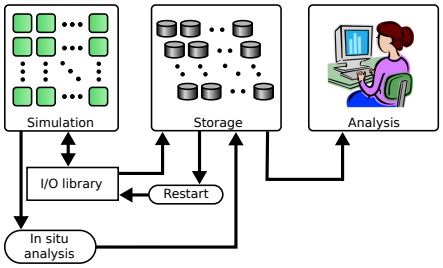


Figure 1. Overview of the simulation/analysis pipeline.

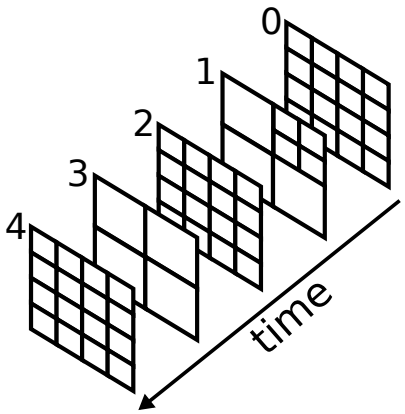


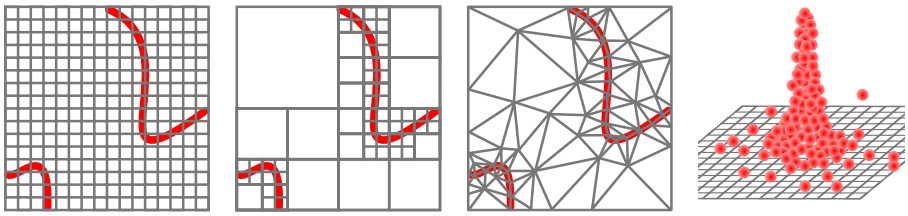
Figure 2. Data may not be dumped at every simulation timestep, and different views of the data may be dumped at different times. Timesteps 0, 2, and 4 store full-resolution versions of the data. Timesteps 1 and 3 store decimated versions of the data.

Debugging Scientific simulations usually comprise extremely complex code. As a result, the heaviest use of simulation output data is in developing and debugging the simulation itself. Simulation code may be written over a long period of time using well-understood data. Data dumps from test runs allow the developer to compare against expected results and debug the code. The ability to efficiently dump data at any given time step is critical in streamlining development.

Figure 2 shows an example of data dumps during a simulation. Dumps 1 and 5 are full-resolution state representations of the initial and final states of the system, respectively. Dump 2 is a checkpoint, or full-resolution data dump. Dumps 1 and 3 store only a subset of the data, either storing only certain regions, or storing at lower resolution, or a combination of the two.

Most simulations are done in one of three different frameworks (Figure 3). The most common framework used in a simulation on a supercomputer is a Cartesian, regular, or rectilinear grid approach. The highly structured format of rectilinear simulations simplifies processor allocation and communication. However, it leads to some waste in resources, as less important areas of the simulation are treated with the same spatial resolution as more important areas.

Adaptive Mesh Refinement (AMR) simulations attempt to solve this by simulating at higher resolutions in areas of interest, but still partitioning into rectangular cells. Mesh construction, processor allocation, and processor communi-



(a) Rectilinear (b) AMR (c) Unstructured (d) Particle

Figure 3. Types of data structures used in simulation.

cation are necessarily more complex, but gains from adaptivity often justify the complexity.

Unstructured grids use elements other than rectangles, often using simplices (triangles in 2D; tetrahedra in 3D). These types of simulations provide tremendous flexibility and are frequently used for many types of simulations, but present challenges when moving to a parallel environment, especially on a supercomputer.

Rectilinear, AMR, and unstructured simulations typically solve a partial differential equation over a domain decomposition. Another type of simulation is particle simulation, where particles are followed in their paths timestep by timestep. These simulations require accurate computations of particle interactions and behaviors.

From a data management perspective, the type of decomposition of the domain is important both for how the data is stored on the storage device, and how the data is transferred to the storage device. For example, in a rectilinear simulation it may make sense to store some data dumps adaptively, while it makes little sense to store an unstructured grid as a Cartesian grid. As mentioned, the decomposition type also affects the size of the dumps, however, steps can be taken to reduce dump size in certain cases. If a suitable decimation strategy is possible, then some dumps for debugging or analysis can represent a low-resolution version of the data, or can represent a region of interest of the data.

1. Supercomputer Storage Infrastructure

Supercomputers are identified by their immense computational capability stemming from their thousands and millions of parallel processing units. As an example, leadership class machines such as the IBM Blue Gene/Q supercomputer at the Argonne National Laboratory [3] and the Cray XT system at the Oak Ridge National Laboratory [4] respectively consist of 750K and 225K processing units and have peak performances of 10 and 1.75 petaflops.

Supercomputers often differ from each other with respect to architecture, inter-connect network, operating system environments, file system and many other parameters. A common concept is that of processing units (cores) and nodes. Processors are grouped into nodes when then are linked together with the interconnect network. Compute nodes perform the simulation while I/O nodes provide I/O functionality. Compute nodes generally don't have access to the file system, so data is routed through I/O nodes while interface with file system software.

1.1. Parallel File System

The file system is the direct interface to the the storage system of the supercomputer. Being able to use and access the file system optimally is key to I/O performance. File systems often have several tunable parameters, such as stripe count and size, that need to be set optimally. In the next subsection, we explore the two most commonly used file-systems, Lustre [5] and GPFS [6] in more detail. Both Lustre and GPFS file systems are scalable and can be part of multiple computer clusters with thousands of client nodes and several petabytes (PB) of storage. This makes these file systems a popular choice for supercomputing data centers, including those in industries such as meteorology, oil, and gas exploration.

1.1.1. Lustre

Lustre [5] is an open-source, high performance parallel file system that is currently used in over 60% of the Top100 supercomputers in the world. It is designed for scalability and is capable of handling extremely large volumes of data and files with high availibility and coordination of both data and metadata. For example, the Spider supercomputer at Oak Ridge National Laboratories has 10.7 petabytes of disk space and moves data at 240 GB/second [7]. The architecture is based on storage of distributed objects. Lustre delegates block storage management to its back-end servers and eliminates significant scaling and performance issues associated with the consistent management of distributed block storage metadata.

1.1.2. GPFS

The General Purpose File System (GPFS) [6] is a parallel file system for high perfomance computing and data intensive applications produced by IBM. It is based on a shared storage model. It automatically distributes and manages files while providing a single consistent view of the file system to every node in the cluster. The filesystem enables efficient data sharing between nodes within and across clusters using standard file system APIs and standard POSIX semantics. IBM is transitioning GPFS to a new product, Spectrum Scale [8], which uses GPFS technology and requires a Spectrum Scale server license from IBM for each node dedicated to the network file system.

1.2. Network Topology

Network topology describes how compute nodes in a supercomputer are connected, and plays an important role in data movement. Network topologies are designed to optimize internode communications for simulation and not I/O. Data movement for I/O typically is designed around the network topology. I/O libraries such as GLEAN [9] leverage the network topology in moving data between cores.

Network topologies come in several varieties. The Mira supercomputer at Argonne National Laboratories is an IBM Blue Gene/Q system [10], that uses a 5D torus network for I/O and internode communication. The Edison machine employs the "Dragonfly" topology [11] for the interconnection network. This topology is a group of interconnected local routers connected to other similar router

groups by high speed global links. The groups are arranged such that data transfer from one group to another requires only one route through a global link.

Processor counts in supercomputers continue to grow, and as a result it becomes more essential to exploit the structure and topology of the interconnect network.

2. Strategies in Storing Data

There are three considerations to be made in deciding on a file format for data storage. The first is I/O speed. Given a supercomputer configuration, we must consider how costly it is to transfer data from compute nodes to the storage medium in the storage format. The more closely a data format matches the data in memory, the more efficient the I/O will be. If the format doesn't match up well, then the data must be reformatted in memory, and the performance of this operation is subject to the network infrastructure.

The second consideration is what the data will be used for. If the data is to be used for restarts, one format may be suitable, but if the data is to be visualized, an entirely different format may be the best.

The third consideration is size. The smallest size possible for uncompressed data is the size of the data itself. But this size may grow with metadata, replicated data, and unused data. Metadata is required in some form in all formats. It may simply describe the dimensions of the data, or more complex things like data hierarchy. Depending on the format, metadata size may be negligible, or it may rival the size of the data itself.

Some file formats have replicated data. For example, a pyramidal scheme of a rectilinear grid may store pixel values at every level of the pyramid. Similarly, unused data, such as filler pixels in a data blocking scheme, can increase storage footprint.

2.1. Number of Files

The number of actual files is also a consideration. In one approach, commonly called file-per-process I/O or $N - N$ output, each processor writes to its own file. N is the number of processors. This approach limits our choice of file format. That is, many formats require global knowledge of the data, such as in the case of storing data hierarchically. If each processor writes its own file then global knowledge of the data is not available and sophisticated formats cannot be used. Further, I/O nodes must write to a large number of files, creating a bottleneck. And finally, the number of files produced may also put a burden on downstream visualization and analysis software. Nevertheless, $N - N$ strategies are popular due to their simplicity.

$N - 1$ approaches route all data to a single file. These approaches are more complex due to the required inter-node communication, but there is much more flexibility to optimize. A straightforward optimization is that of larger block writes to disk. That is, large chunks of data can be written at a time, making writes more efficient. Optimization in file structure is also possible. If the data is to be

used for visualization, a hierarchical stream-optimized format such as IDX [12] may be used. Or, if only a subset of the data is needed, sampling or other type of decimation may be done before the write.

Subfiling approaches ($N - M$) provide ultimate flexibility, where the number of files is tuned for maximum performance. Subfiling is discussed in Section 4.3.

2.2. Formats and Libraries

The most popular file formats in large-scale simulation are of the $N - 1$ variety, of which PnetCDF and HDF5 are the most common. Formats generally have an accompanying I/O library, easing use of a particular format. This section discusses both formats and libraries.

2.2.1. MPI-IO

MPI-IO is a standard, portable interface for parallel file I/O that was defined as part of the MPI-2 (Message Passing Interface) Standard in 1997. It can be used either directly by applications programmers or by writers of high-level libraries as an interface for portable, high performance I/O in parallel programs. MPI-IO is an interface that sits above a parallel file system and below an application or high-level I/O library as illustrated in Figure 6. Here it is often referred as "middleware" for parallel I/O. MPI-IO is intended as an interface for multiple processes of a parallel program that is writing/reading parts of a single common file. MPI-IO can be used for both independent I/O where all processes perform write operations independent of each other, as well as in collective I/O mode, where processes coordinate amongst each other and a few processes end up writing all the data. MPI-IO supports both blocking and nonblocking modes of I/O, with the latter one providing potential for overlapping I/O and computation.

2.2.2. HDF5

HDF5 [13] short for "Hierarchial Data Format, version 5" is designed at three levels: data model, file format and I/O library. The data model consists of abstract classes such as files, groups, datasets, and datatypes, which are instantiated in the form of a file format. The I/O library provides applications with an object-oriented programming interface that is powerful, flexible and highly performant. The data model allows storage of diverse data types, expressed in a customizable, hierarchical organization. The I/O library extracts performance by leveraging MPI-IO collective I/O operations for data aggregation. Owing to the very customizable nature of the format, HDF5 allows users to optimize for their data type, making performance tuning partially the responsibility of the user.

2.2.3. PnetCDF

Parallel NetCDF [14] is another popular high level library with similar functionality to HDF5 but in an file format that is compatible with serial NetCDF from Unidata. PnetCDF is a single shared file approach ($N - 1$), and is optimized for dense, regular datasets. It is inefficient for hierarchical or region of interest (ROI) data in both performance and storage, and so is used only in rectilinear simulation environments.

2.2.4. ADIOS

ADIOS [15] is another popular library used to manage parallel I/O for scientific applications. One of the key features of ADIOS is that it decouples the description of the data along with transforms to be applied to that data from the application itself. ADIOS supports a variety of back-end formats and plug-ins that can be selected at run time.

2.2.5. GLEAN

GLEAN [9] developed at Argonne National Laboratoy, provides a topology-aware mechanism for impoved data movement, compression, subfiling, and staging for I/O acceleration. It also provides interfaces for co-analysis and in-situ analysis, requiring little or no modification to the existing application code base.

2.2.6. PIDX

The PIDX I/O library [16,17] enables concurrent writes from multiple cores into the IDX format, a cache-oblivious multi-resolution data format inherently suitable for fast analytics and visualization. PIDX is an $N - M$ approach, contributing to better performance. The number of files to generate can be adjusted based on the file system. This approach extracts more performance out of parallel file systems, and is customizable to specific file systems. Further, PIDX utilizes a customized aggregation phase, leveraging concurrency and leading to more optimized file access patterns.

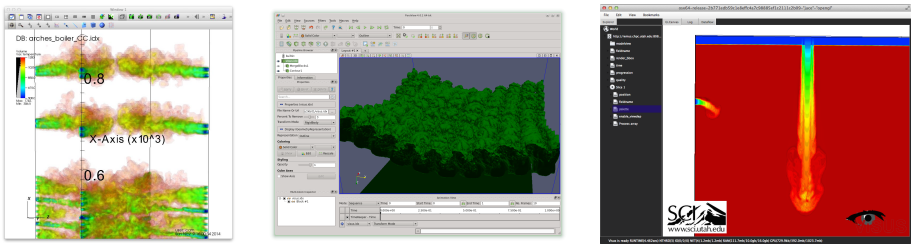
PIDX is naturally suited to multiresolution AMR datasets, adaptive region of interest (ROI) storage of rectilinear grids, and visualization and analysis. IDX does not need to store metadata associated with AMR levels or adaptive ROI; hierarchical and spatial layout characteristics are implicit.

3. Visualization and Analysis

How data is stored has direct impact on how it is visualized and analyzed. The vast majority of simulations undergo significant analysis after completion. Figure 4 shows examples of visualizations. We briefly describe three popular visualization packages.

3.1. VisIt and ParaView

VisIt [18] and ParaView [19] are popular distributed parallel visualization and analysis applications. They are typically executed in parallel, coordinating visualization and analysis tasks for massive simulation data. The data is typically loaded at full resolution, requiring large amounts of system memory. Both packages utilize a plugin-based architecture, so many formats are supported by both. They are open source and platform-independent and have been deployed on various supercomputers as well as on desktop operating systems such as Windows, Mac OS X, and Linux.



(a) VisIt (b) ParaView (c) ViSUS

Figure 4. Visualizations using different software packages.

3.2. ViSUS

ViSUS [20] is designed for streaming of massive data and uses hierarchical data representation technology, facilitating online visualization and analysis. It's design allows interactive exploration of massive datasets on commodity hardware, including desktops, laptops, and hand-held devices. Rather than running in a distributed environment, PIDX supports thread-parallel operation. The IDX streaming data format is supported natively.

4. Optimizations

No single I/O strategy is suitable for all systems. Supercomputers come in all flavors of size, topology, and design. Ideal I/O libraries are flexible and customizable to the type of system they're running on.

4.1. Restructuring and Aggregation

The process of encoding data is that of organizing data in memory as it is going to be stored after the write. This simplifies the write to storage, but costs some memory, as the data will effectively be duplicated. Very often each node writes its data to storage directly. On systems with dedicated I/O nodes, data is transferred to the I/O nodes and then written. Two problems with these approaches are that, first, file format optimizations are challenging and second, if a node has fragmented data, it will require many writes to get all the data written without overwriting data from other nodes. This becomes particularly expensive when the storage medium is a hard drive. Further, if the data is fragmented, then if the node encodes into a single array, then there will be a lot of unused allocations. Two optimizations help alleviate these issues.

The first is aggregation (see Fig. 5). After each node encodes its data, an inter-node communication phase called aggregation is executed, which reorganizes the data onto "aggregation nodes." The data is organized such that each aggregation node will need to perform only a single, large block write. Aggregation significantly speeds up the write to storage. But there still exists the problem of possible excessive memory usage when encoding.

Restructuring is the process of reallocating spatial regions to nodes so that encoding arrays will have far fewer unused allocations, and so that aggregation

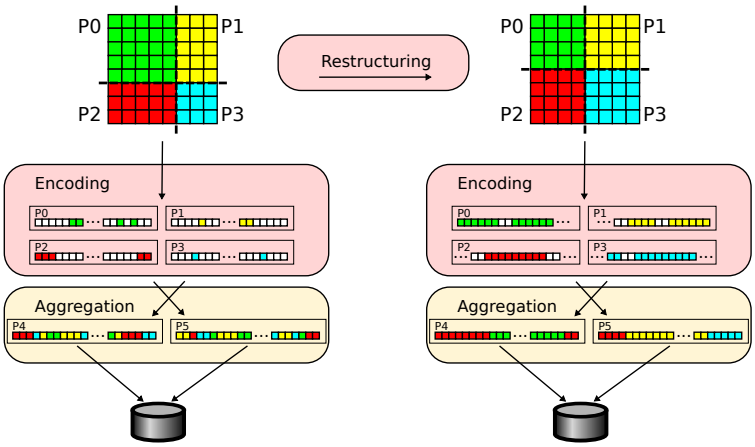


Figure 5. Restructuring and aggregation.

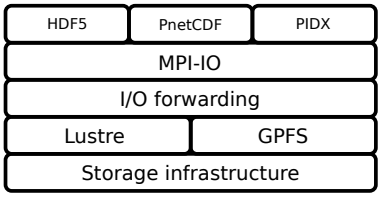


Figure 6. I/O software stack with I/O forwarding.

will require nodes to communicate with fewer aggregation nodes. For example, in Fig. 5, because restructuring was done, node P0 will need to send data only to aggregation node P4, instead of to both P4 and P5 if restructuring isn't done. Further, much less memory need be allocated on P0 for encoding since the data isn't fragmented.

4.2. I/O Forwarding

Previous sections have referred to I/O nodes. These nodes perform I/O operations on behalf of compute nodes. I/O forwarding is the process of shipping I/O calls from compute nodes to I/O nodes in order to relieve compute nodes from potentially costly I/O operations (see Figure 6). I/O forwarding software is an interface between the file system and the remainder of the software stack, so any optimization made in the I/O forwarding layer is transparent to applications and high-level I/O libraries. The I/O forwarding subsystem optimizes by aggregating, rescheduling, and caching I/O requests.

4.3. Subfiling

The number of output files for an application have significant effect on I/O performance. This number can vary from a single shared file for all processes to one file per process. Too few files (such as $N-1$ or shared file) or too many files ($N-N$

or file per process) both result in I/O bottlenecks. Since I/O nodes manage the file metadata, too many files per I/O node or too many I/O nodes sharing a file both lead to a bottleneck in metadata management. As an example, on Mira, a Blue Gene/Q machine with GPFS file system, Bui et al [21] write one output file per I/O node to avoid inter-node communication or file metadata management overhead, resulting in better performance than the shared file and file per process options.

4.4. Parameter Learning

Realizing high I/O performance for a broad range of applications on all HPC platforms is a major challenge, in part because of complex inter-dependencies between I/O middleware and hardware (see Fig 6). The parallel file system and I/O middleware layers all offer optimization parameters that in theory can result in optimal I/O performance. Unfortunately, it is not easy to derive a set of optimized paameters, as it largely depends on the application, HPC platform, problem size, and concurrency. In order to optimally use HPC resources, an auto-tuning system can hide the complexity of the I/O stack by automatically identifying parameters that accelerates I/O performance.

Earlier work in autotuning I/O research has proposed analytical models, heuristic models, and trial-and-error based approaches. The models can then be used to obtain optimal parameters. Unfortunately, all these methods have known limitations [22] and do not generalize well to a wide variety of settings. Modeling techniques based on machine learning overcome these system limitations and build a knowledge-based model that is independent of the specific hardware, underlying file system, or custom library used. Based on the flexibility and independence to a variety of constraints, machine learning techniques have achieved tremendous success in extracting complex relationships just from the training data itself. As an example, PIDX I/O library builds a machine learning-based model using regression analysis [23] on data sets collected during previously conducted characterization studies. The model has the ability to predict performance and identify optimal tuning parameters for a given scenario. With approaches such as this, the efficiency of the performance model increases over time when more training data comes available.Similarly, Parallel HDF5 [24], uses a genetic algorithm to search a large space of tunable parameters and to identify effective settings at all layers of the parallel I/O stack (see Figure 6).

5. Case Study

Traditional I/O libraries and corresponding data formats such as HDF5 and PnetCDF are general-purpose in nature, and generally do not customize to analysis and visualization. On the other hand the chief advantage of PIDX I/O library is that it reorders individual samples in a manner that enables realtime multi-resolution visualization and analysis through use of the IDX data format. It is also tunable for different supercomputer configurations, and has demonstrated excellent write perfomance [16].

This section follows the data from setup of an S3D [25] uniform simulation of the lifted ethylene jet (one of the largest combustion simulations performed by S3D) through to visualization and analysis. This simulation has 16 fields of interest, including temperature, pressure, velocity and chemical species. The field we focus on is temperature.

We choose the Hopper supercomputer [26] for the simulation. We set the simulation up to run for 2500 timesteps. Because full dumps are expensive, we choose to dump full-resolution checkpoint data (Fig. 7a) only every 100 timesteps, primarily for restart in case the simulation halts prematurely.

Using an advanced I/O library like PIDX gives us options for intermediate dumps for monitoring and analysis. We set two thresholds on the temperature field, breaking the domain into three regions of high, medium, and low temperature. We are primarily interested in regions of high temperature, so we save those at full resolution, while medium temperature is saved at 1/64 resolution and low temperature regions are saved at 1/512 resolution (Fig. 7b). With this ability to store data adaptively at varying resolution, we are able to save on both storage space as well as compute time, making it possible to store intermediate snapshots evey 10 time steps.

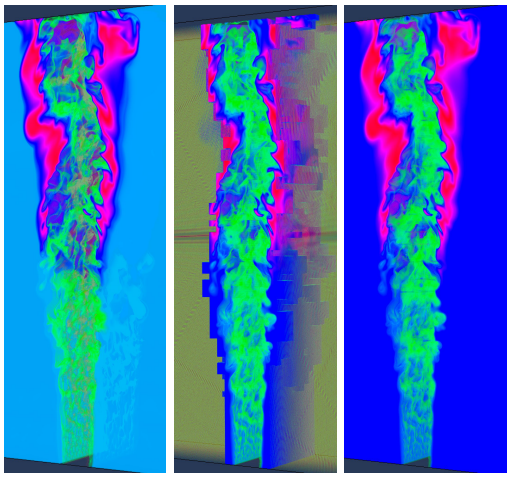
The two flame sheets most easily distinguished on the bottom of Fig. 7 burn very hot and thus get preserved at full resolution. The outside coflow on the left and right is heated by the central flame and thus resides in the medium temperature region. Finally, the channel in between the sheets contains the relatively cool fuel stream which gets classified as low temperature. Together, this configuration creates many sharp resolution drops and isolated regions as well as a significantly uneven data distribution. The resulting adaptive resolution IDX output takes only 39% of the full resolution output time, while writing 30% of the 12.8 GB of full resolution data. As shown in Fig. 7c, the resulting volume rendering using up-sampling to create a uniform resolution grid preserves the regions of inteest (ROI) almost perfectly while showing the expected artifacts especially in the center of the flame.

In order to monitor the simulation, we set up a server that updates the intermediate results in our visualization system. In this case, we use the ViSUS visualization framework (VisIt also has a plugin to support the IDX format) to see the updates as they come in. The data coming in is suitable for visualization as-is. However, we can upsample the data (Fig. 7c) to obtain a full-resolution version of the domain and run intermediate analyses, such as topological, statistical, or shape analyses. This allows us to start understanding the data even before the simulation has completed.

If the simulation halts for some reason, we retrieve one of the checkpoint dumps, use that data as the initial conditions, and restart the simulation. Once the simulation is complete we transfer the intermediate dumps and final results from the supercomputer to our own storage.

6. In Situ Analytics

Many practitioners use data reduction, such as the IDX format, to handle massive amounts of data. Another approach is to perform analytics on the data dur-



(a) Full (b) ROI (c) Upsampled

Figure 7. Volume rendering of the temperature field of the lifted ethylene jet. (a) Full resolution data. (b) Adaptively sampled data. (c) Adaptively sampled data, up-sampled to create a uniform image.

ing the simulation, with the analytics code running on the supercomputer nodes themselves, called in situ analytics. The big advantage to in situ is that data I/O, typically a major bottleneck, can be reduced by transferring only analysis results in timesteps between full checkpoint dumps. For example, one can use a parallel algorithm to compute merge trees [27], resulting in a vastly smaller version of the data. In situ analyses can include topological, decimated, shape descriptive, statistical, and other summary views of the data.

One shortcoming of in situ analytics is that an analysis code must be customized to the supercomputer running the simulation, and thus must address parallel programming, node communication, and scaling considerations. Decisions as to how much analysis to perform, and its tradeoffs with slowing down the simulation must also be made.

7. Conclusions

Big data from simulations is inextricably linked with the simulation itself. For example, if I/O is a major bottleneck in the simulation, only a small subset of state data may be stored and available for analysis. The data may be written in a format optimized for simulation I/O, and not for analysis. And analysis may be better done in situ. In short, the entire simulation and analysis pipeline must be looked at holistically when addressing data considerations. Of course, data may be converted to desired formats, but this can come at considerable processing and storage expense, and is not a solution for retrieving missing data from decimation. When planning a simulation run, data tranfer and usage needs should be carefully weighed against I/O performance needs, and suitable I/O libraries, formats, and spatiotemporal resolution parameters should be chosen accordingly.

References

[1] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P Sadayappan. Scalable i/o forwarding framework for high-performance computing systems. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009.

[2] Volker Springel, Simon DM White, Adrian Jenkins, Carlos S Frenk, Naoki Yoshida, Liang Gao, Julio Navarro, Robert Thacker, Darren Croton, John Helly, et al. Simulating the joint evolution of quasars, galaxies and their large-scale distribution. *arXiv preprint astro-ph/0504097*, 2005.

[3] Preparing applications for mira, a 10 PetaFLOPS IBM blue gene/q system. http://www.alcf.anl.gov/files/PrepAppsForMira_SC11_0.pdf.

[4] Cray XT5. http://en.wikipedia.org/wiki/Cray_XT5.

[5] Lustre home page. http://lustre.org.

[6] Frank B Schmuck and Roger L Haskin. GPFS: A shared-disk file system for large computing clusters. In *FAST*, volume 2, page 19, 2002.

[7] Spider up and spinning connections to all computing platforms at ORNL. http://www.hpcwire.com/2009/07/09/spider_up_and_spinning_connections_to_all_computing_platforms_at_ornl/.

[8] IBM Spectrum Scale. http://public.dhe.ibm.com/common/ssi/ecm/dc/en/dcw03051usen/DCW03051USEN.PDF.

[9] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E. Papka. 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*, SC '11, pages 19:1–19:11, New York, NY, USA, 2011. ACM.

[10] Dong Chen, Noel Eisley, Philip Heidelberger, Sameer Kumar, Amith Mamidala, Fabrizio Petrini, Robert Senger, Yutaka Sugawara, Robert Walkup, Burkhard Steinmacher-Burow, et al. Looking under the hood of the IBM Blue Gene/Q network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 69. IEEE Computer Society Press, 2012.

[11] Edison Dragonfly topology. https://www.nersc.gov/users/computational-systems/edison/configuration/interconnect/.

[12] Valerio Pascucci and Robert J Frank. Global static indexing for real-time exploration of very large regular grids. In *Conference on High Performance Networking and Computing, archive proceedings of the ACM/IEEE Conference on Supercomputing*, 2001.

[13] HDF5 home page. http://www.hdfgroup.org/HDF5/.

[14] Jianwei Li, Wei-Keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of SC2003: High Performance Networking and Computing*, Phoenix, AZ, November 2003. IEEE Computer Society Press.

[15] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. 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*, pages 15–24, New York, June 2008. ACM.

[16] S. Kumar, V. Vishwanath, P. Carns, J.A Levine, R. Latham, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M.E. Papka, J. Chen, and V. Pascucci. Efficient data restructuring and aggregation for I/O acceleration in PIDX. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, Nov 2012.

[17] Sidharth Kumar, John Edwards, Peer-Timo Bremer, Aaron Knoll, Cameron Christensen, Venkatram Vishwanath, Philip Carns, John A Schmidt, and Valerio Pascucci. Efficient i/o and storage of adaptive-resolution data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 413–423. IEEE Press, 2014.

[18] VisIt home page. https://wci.llnl.gov/codes/visit/.

[19] ParaView home page. http://www.paraview.org/.

[20] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, S. Philip, and S. Kumar. The ViSUS visualization framework. In E Wes Bethel, Hank Childs, and Charles Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, 2012.

[21] Huy Bui, Hal Finkel, Venkatram Vishwanath, Salma Habib, Katrin Heitmann, Jason Leigh, Michael Papka, and Kevin Harms. Scalable parallel i/o on a blue gene/q supercomputer using compression, topology-aware data aggregation, and subfiling. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 107–111. IEEE, 2014.

[22] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 83:1–83:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[23] Sidharth Kumar, Avishek Saha, Venkatram Vishwanath, Philip Carns, John A Schmidt, Giorgio Scorzelli, Hemanth Kolla, Ray Grout, Robert Latham, Robert Ross, et al. Characterization and modeling of pidx parallel I/O for performance optimization. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 67. ACM, 2013.

[24] Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Prabhat, Ruth Aydt, Quincey Koziol, and Marc Snir. Taming parallel i/o complexity with auto-tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 68:1–68:12, New York, NY, USA, 2013. ACM.

[25] J H Chen, A Choudhary, B de Supinski, M DeVries, E R Hawkes, S Klasky, W K Liao, K L Ma, J Mellor Crummey, N Podhorszki, R Sankaran, S Shende, and C S Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. In *Computational Science and Discovery Volume 2*, January 2009.

[26] Hopper home page. https://www.nersc.gov/users/computational-systems/hopper/.

[27] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jacqueline Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031. IEEE Press, 2014.