Online Data Analysis and Reduction: An Important Co-design Motif for Extreme-Scale Computers

Ian Foster¹, Mark Ainsworth³, Julie Bessac¹, Franck Cappello¹, Jong Choi⁴, Sheng Di¹, Zichao Di¹, Ali Murat Gok¹, Hanqi Guo¹, Kevin A. Huck⁸, Christopher Kelly², Scott Klasky⁴, Kerstin Kleese van Dam², Xin Liang⁴, Kshitij Mehta⁴, Manish Parashar⁵, Tom Peterka¹, Line Pouchard², Tong Shu¹, Hubertus Van Dam², Matthew Wolf⁴, Justin M. Wozniak¹, Wei Xu², Igor Yakushin¹, Shinjae Yoo², Todd Munson¹

Abstract
A growing disparity between supercomputer computation speeds and I/O rates means that it is rapidly becoming infeasible to analyze supercomputer application output only after that output has been written to a file system. Instead, data-generating applications must run concurrently with data reduction and/or analysis operations, with which they exchange information via high-speed methods such as interprocess communications. The resulting parallel computing motif, online data analysis and reduction (ODAR), has important implications for both application and HPC systems design. Here we introduce the ODAR motif and its co-design concerns, describe a co-design process for identifying and addressing those concerns, present tools that assist in the co-design process, and present case studies to illustrate the use of the process and tools in practical settings.

Keywords
Online data analysis and reduction, data analysis, in situ, exascale computing

Introduction
Computer architect Gene Amdahl argued in 1965 that a balanced computer should support one bit of I/O per second for each instruction per second (Gray and Shenoy 2000), which for an exascale computer (one sustaining $10^{18}$ operations per second), would mean $10^{17}$ bytes per second (B/s). In practice, exascale computers will support I/O rates of little more than $10^{10}$ B/s. This large disparity between compute and I/O speeds—a disparity that has grown substantially over the past decade—makes it increasingly infeasible for programs to output large quantities of computed data for later analysis. Either analyses must be performed online (i.e., while the application is running) or data reduction computations must be performed, again online, to downscale the data written to disk. The result is a new parallel program structure, or motif (Asanovic et al. 2006), online data analysis and reduction (ODAR).

The ODAR motif has broad implications for both HPC applications and HPC systems. Application developers must carefully balance the costs and information content that results when data are produced via different methods. HPC system architects need to consider ODAR concerns when designing and configuring programming models, libraries, runtime systems, and storage systems. Understanding and addressing these concerns requires a co-design process so that different aspects of application and system design can be considered and optimized simultaneously. In response, the Co-design Center for Exascale Machine Learning Technologies (CODAR) (Foster et al. 2017), part of the U.S. Department

¹ Argonne National Laboratory, Lemont, IL, USA
² Brookhaven National Laboratory, Upton, NY, USA
³ Brown University, Providence, RI, USA
⁴ Oak Ridge National Laboratory, Oak Ridge, TN, USA
⁵ Rutgers University, New Brunswick, NJ, USA
⁶ Southern Illinois University, Carbondale, IL, USA
⁷ University of Chicago, Chicago, IL, USA
⁸ University of Oregon, Eugene, OR, USA

Corresponding author:
Ian Foster, Argonne National Laboratory, Lemont, IL 60439, USA
Email: foster@anl.gov
of Energy’s Exascale Computing Project (ECP), has worked closely with ECP application projects (Alexander et al. 2020a) and technology projects on co-design questions raised by ODAR methods.

In the sections that follow, we: 1) introduce the ODAR motif, its importance for high-performance computing, and the co-design questions that it raises for HPC applications and systems; 2) propose a co-design process for applications that use the ODAR motif; 3) describe tools that can assist with the execution of this co-design process; 4) present case studies of this co-design process in various practical settings; and 5) discuss lessons learned from these studies about the co-design process.

Background

As motivation for the HPC application motifs and co-design studies considered in this article, we review briefly both challenges associated with online data analysis and reduction, and the need for a co-design approach to tackle those challenges.

Challenges in online data analysis and reduction

An early motivation for online data processing was to visualize the results of a computation as it was running (Beazley and Lomdahl 1996; Foster et al. 1999; Johnson et al. 1999). Due to difficulties in running multi-component applications on earlier HPC systems, data might be communicated to a specialized visualization computer. As data volumes grew, online processing became even more important, and distributed computing less attractive. New structures then emerged in which data were generated and processed on the same system, a subset of online processing often referred as \textit{in situ} processing (Ma et al. 2007; Insley et al. 2007; Klasky et al. 2011; Bauer et al. 2016; Larsen et al. 2017; Childs et al. 2020).

The following example may give more of a sense of why online analysis and reduction are so important in exascale computing. A high-resolution (3 km) climate simulation model running on an exascale computer may maintain an internal state of around 80 TB, and update that state twice per second, for an aggregate state update rate of 160 TB/s (R. Jacob, personal communication). Even at a sustained output rate of 1 TB/s (already infeasible, given that the model would then output 86 PB per compute-day, leading to capacity limits and making subsequent analysis extremely difficult), only 0.6% of those data could be output to secondary storage.

Decades of work in online data processing has led to many innovations in data reduction methods, programming models, communication libraries, and other areas. It has also exposed challenging questions regarding HPC application and system design (Ayachit et al. 2016). For example: What reduction and analysis algorithms best balance potentially conflicting needs for performance, fidelity, and flexibility in subsequent offline analysis? How should application, analysis, and reduction components be mapped across increasingly complex hardware for maximal performance? If it proves to be more efficient to map different components to different processors, rather than replicating each component on all processors in a single program multiple data (SPMD) structure, how should the resulting multiple program multiple data (MPMD) structures be implemented? How should component programs exchange information? Does online data processing demand different memory, communication, and storage system capabilities?

Online processing as a co-design problem

Many of these questions have implications for multiple elements of the HPC system, from application to analysis and reduction libraries, programming libraries, system software, and HPC systems design.

In other words, they are \textit{co-design problems}.

The term co-design was first used in embedded systems to refer to the process of “meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design” (De Michell and Gupta 1997). The term was introduced in the HPC context to refer to a similar process of hardware-software co-design (see Figure 1), but it is increasingly used to refer to any design process that reaches across component boundaries to encompass, for example, the design of applications, reduction and analysis methods, and component coupling methods. In that context, a co-design problem is one that is not concerned only with optimizing a single program on a single computer, but with developing understanding of the interrelationships among the structure and properties of application, programming libraries, system software, and other system components. It is in that latter sense that we use the term in this article.

The Online Analysis and Reduction Motif

The ODAR motif is needed most in situations in which the data generated by one application component (Simulate) for consumption by another application component (Analyze) are too large, or the required turnaround time (in terms of analysis response) is too short, for the data to be exchanged offline via the file system. (Here and elsewhere, Simulate and Analyze...
can be any program that generate and consume data, respectively.) In such situations, the application developer must either reduce the size of the data prior to writing it to storage, and/or run Simulate and Analyze concurrently, with data passed via means other than the file system.

In practice, Simulate may generate a range of data that need to be treated in different ways. Thus, a single application may involve one or more of the components and communication paths shown in Figure 2.

**Perspectives on ODAR and co-design**

We structure our discussion of ODAR co-design questions in terms of four perspectives.

*Information-theoretic perspectives.* How can we maximize the useful information produced for a fixed budget of computation time and storage? To answer this question, the application developer needs first of all to know which data produced by a computation are most likely to be needed subsequently. For each data analysis and reduction method that may be executed online, the application developer will need to understand its impact on data size, information content, and computational costs. Armed with what will likely be imperfect knowledge of intended uses and component performance, the developer must then decide to output some fields and not others, compress certain fields, and perform selected analyses online (to avoid the need to re-read data)—all with the goal of maximizing the useful information stored.

HPC applications often address such questions via simple heuristics, such as computing statistical summaries of some fields and performing temporal or spatial decimation in others. However, such simple approaches may be inefficient and can easily hide important phenomena, especially as the disparity between computer speed and I/O rates grow. Alternatively, while lossless compression of scientific data is rarely effective (Ratanaworabhan et al. 2006), such methods can be used, albeit at the risks of losing information or introducing systematic errors.

*Resource management perspectives.* The ODAR motif can also be viewed from a resource management perspective. The developer is typically faced with the problem of allocating limited resources across a set of heterogeneous Simulate, Reduce, and Analyze components. Should one place all or a subset of the components on the same resources, in an SPMD structure, or on different resources (MPMD)? How should different resource types (CPUs, GPUs) be allocated to different components? Should a Reduce or Analyze component be placed on a single node (thereby simplifying its implementation, but limiting its performance) or on multiple nodes? Different placement choices may make different intra- and inter-component communications mechanisms available, each with different performance characteristics (Choi et al. 2018; Malakar et al. 2015, 2016, 2018).

*HPC systems perspectives.* The ODAR motif also has implications for the design of HPC programming models, libraries, system software, and other components. For example, many HPC system components have historically been designed to support primarily SPMD computations, in which every computer node assigned to an application runs the same program. Applications that involve online analysis and reduction, and other MPMD applications, can benefit from the ability to create multiple SPMD computations, to manage the placement of application components, and to manage, monitor, and organize communications among those computations.

The ODAR motif also has implications for HPC hardware and architecture. For example, non-volatile memory, intermediate in speed between memory and storage, can allow for large-scale buffering of data produced by communicating online components. The quantity, organization, and programming interfaces
of such hardware can make a big difference to ODAR applications. Understanding those tradeoffs is important for both evaluating portability and designing future machines.

**Software engineering perspectives.** Finally, the ODAR motif has implications for software engineering. In keeping with principles of information hiding in modular design (Parnas 1972), we want to avoid committing to design decisions early in an implementation that may need to be revisited later. For example, linking application and reduction code into a single executable will make it difficult to experiment later with configurations in which application and reduction components run on different nodes, or to substitute different data reduction modules. Conversely, designing for flexibility may incur additional programming costs and performance overheads.

**Variants of the ODAR motif**

The ODAR motif can be instantiated in many ways. Each of the following variants, frequently encountered in ECP applications, raise specialized co-design questions.

**Multiple-component applications.** Applications may comprise multiple distinct *Simulate* programs (e.g., multiphysics applications) and/or multiple instances of the same program (e.g., ensemble studies). In the latter case, the number of instances may be fixed, with each running for the duration of the computation, or the number and/or duration of instances can vary over time. For example, different application instances may explore different parts of a complex phase space of molecular dynamics trajectories; periodically, results are aggregated and compared, noninteresting trajectories discarded, and new trajectories initiated (Lee et al. 2019).

In multi-component applications, each component may produce results asynchronously, leading to a need to cache many results until a sufficient amount of data is available for analysis. Asynchronous coordination structures such as DataSpaces (Docan et al. 2012) can be useful, as we later discuss in our CANDLE case study. Specialized reduction methods such as deep learning may be used to identify interesting elements in ensembles (He et al. 2019) and to evaluate the quality of trajectories (Lee et al. 2019).

**Two-way coupling.** Another variant of the ODAR motif involves two-way coupling between *Simulate* and *Analyze*, as when analysis output motivates a change to the simulation component: for example, detection of turbulent flow spurs mesh refinement.

---

**The ODAR Co-design Process**

Co-design questions are concerned with understanding tradeoffs among different elements of a multi-component system. If I change one component, what demands does that place on a second? Can I simplify or accelerate a third component by changing the behavior of a fourth? A good co-design process helps to expose such dependencies, build understanding of their implications, and avoid expensive mistakes such as locking in to a bad design of one component due to inadequate understanding of dependencies.

**ODAR Co-design Challenges**

ODAR co-design is complicated by a much larger configuration space than is found in many conventional application design problems. In the case of a single application, the number of configurable parameters is often small, and thus we can identify good values for configuration parameters via a mix of performance modeling and experiments (Foster 1994; Hoefler et al. 2011; Duplyakin et al. 2016; Balaprakash et al. 2018). In contrast, consider an ODAR code that couples one or more simulation, analysis, and reduction applications. In addition to problem size and computer characteristics, we may be concerned with the resources allocated to each component, their placement, interprocess communication methods, compression method, compression parameters, analysis method, and analysis frequency, among other factors. Interdependencies among components mean that it is rarely feasible to optimize each component individually (Shu et al. 2020); thus, the number of possible configurations can be many orders of magnitude larger than in the case of a single application.

Complex evaluation metrics are a second complicating factor. For example, while for a simulation application we may care only about speed, for a coupled simulation-analysis-reduction application we will likely also want to evaluate tradeoffs between time spent on data reduction and the volume and quality of the reduced data.

A third common complicating factor in ODAR co-design is the need to be sensitive to the varying costs and time scales of different design decisions. For example, it will typically be both more costly and time-consuming to alter an I/O system than an application’s internal logic. We may think of such considerations as more or less strict constraints on parameter values.

**Sketch of an ODAR Co-design Process**

Design processes for both general-purpose and HPC software have been extensively studied and are in...
many respects well understood (Curtis et al. 1988; Ousterhout 2018). Nevertheless, the complicating factors just noted can require modifications to those processes when co-designing applications that implement the ODAR motif, as we now discuss.

The much larger configuration space and the variety of components to be coupled means that it rarely possible to develop accurate models of all possible design alternatives. However, it is usually feasible to identify both major design decisions (e.g., online or offline analysis, data reduction or not, co-locate or distribute components) and minor design decisions (e.g., mapping to cores, communication mechanisms) and then to explore the tradeoffs inherent in these decisions in an organized fashion. Figure 3 and the following text outline how we view the process.

First, we define the goal of the co-design process: the question(s) to be answered (e.g., “how can we use compression to reduce output size?”) and the associated success metric(s) (e.g., “information content divided by size of output, change in application time”).

Next, we identify the components to be included in the application, potential couplings, and free parameters associated with the component and couplings that may provide opportunities for configuration. For example, an application with Simulate and Reduce components may allow for two alternative coupling strategies: running the two components in sequence on the same nodes, or running them concurrently on different nodes. Free parameters may include number of nodes allocated to each component, mappings of components to nodes and cores, interprocess communication methods, compression method used, and compression parameters.

We then characterize the computational performance of individual components. Depending on overall goals, this may be involve just measuring their execution times for a few different configurations, or we may aim for an analytical model that relates performance to problem size and computer characteristics.

We can then estimate the performance of the coupled system, with a view to determining which couplings may be advantageous. For example, if simulation and reduction run concurrently on different nodes, then we may expect the overall execution time to be the maximum of the simulation and reduction time, plus time for coupling communication; if they run consecutively on the same nodes, the maximum, across all nodes, of the sum of the simulation and reduction times. Based on these estimates, we may decide that some configurations are impractical.

At this point we are ready to define and run experiments, as informed by our initial performance estimates, the overarching questions and evaluation metrics, and constraints such as limited computational budget for experiments, an analysis method that runs only on a power-of-two processors, or a compression method that does not run on GPUs. While exhaustive search of the resulting configuration space may be impractical, a combination of model composition and active learning can be used to achieve far more efficient search (Shu et al. 2020).

If we feel that we have obtained adequate answers to our questions and satisfactory values for our success metrics, we can terminate the co-design process at this point. If not, we may iterate, for example by considering modifications to components to expose more free parameters, refining performance estimates or models for individual components, etc.

### Co-design Technologies

ODAR co-design can be assisted by technologies that support exposing co-design opportunities (e.g., by making it easy to explore alternative mappings of processes to processors, to use alternative communication mechanisms, or to switch among different data reduction methods), evaluating the
Exposing co-design opportunities

The complexity and heterogeneity of modern HPC architectures mean that where computation is performed, and how data are exchanged between processes, can have major performance implications (Choi et al. 2018). We want to allow programmers to express such choices and to experiment with alternative choices, without requiring them to write entirely machine-specific code. In defining and developing mechanisms to provide this ability, we must balance tradeoffs between the application developer’s competing needs for both fine-grained control and simplicity, and the capabilities offered by programming libraries and HPC systems software.

Manage process placement. The first capability that we consider is how to control which processes are placed where—on what nodes, and even on which processes within a node. While the HPC community has standardized the Message Passing Interface (MPI) application programming interface (API) for organizing concurrency and communications within HPC applications, there has been no equivalent standardization of process mapping constructs. Thus, we have defined a virtual node abstraction and interface to provide fine-grained control, when needed, over the mapping of application processes to specific hardware processors. While tedious to employ, this interface provides flexibility and clarity when users need granular placement of processes from multiple applications, as shown in Listing 1. We discuss below its implementation in the Savanna library.

Manage communication methods. Modern HPC systems support a wide variety of inter-process communication mechanisms with different performance characteristics, including MPI, RDMA, TCP/IP, and shared memory. By allowing communication mechanisms to be changed without changes to the application, we expose the choice of communication mechanism as a free parameter. We use the ADIOS2 API and library for this purpose (Godøy et al. 2020). This API abstracts I/O and communication, so that an application can first be written to perform data input and output operations, and then configured at runtime, without any change to application code, to access a file system or to communicate over the network via different mechanisms. ADIOS2 libraries implement these different behaviors, via for example its Sustainable Staging Transport (SST) engine, which uses RDMA mechanisms, and Strong Staging Coupler (SSC) engine, which uses MPI methods for communications.

An application programmer who wants to avail themselves of this flexibility must (re)write their application to use ADIOS2 APIs. CODAR provides a library of implementations of popular compression techniques that use ADIOS2 APIs, including SZ (Di and Cappello 2016), ZFP (Difenderfer et al. 2019), and MGARD (Ainsworth et al. 2019), each packaged to expose tunable parameters such as error tolerance.

As noted earlier, multi-component applications can motivate a need for alternative communication structures such as asynchronous data spaces in which an application may place many data objects for later analysis. CODAR has explored the use of asynchronous coordination structures such as DataSpaces (Docan et al. 2012) for that purpose, for example in the Model Store (Wozniak et al. 2018a) data structure.

Programming models and libraries. HPC programmers often use the Message Passing Interface (MPI)

```
class SummitNode:
    def __init__(self):
        self.cpu = [None] * 42 # 42 CPUs
        self.gpu = [None] * 6 # 6 GPUs

shared_node_layout = SummitNode()
# Create 10 simulation ranks on each socket
for i in range(10):
    # Socket 1: cores range 0-20; place sim on 0-9
    shared_node_layout.cpu[i] = 'sim:' + str(i)
    # Socket 2: cores range 22-42; place sim on 22..31
    shared_node_layout.cpu[22+i] = 'sim:'+ str(10+i)
# Place 2 simulation ranks on each of GPUs 0 and 3
shared_node_layout.gpu[0] = ['sim:0', 'sim:1']
# Create 4 analysis ranks on cores 38..41 of socket 2
shared_node_layout.cpu[38] = 'al:1'
shared_node_layout.cpu[39] = 'al:2'
shared_node_layout.cpu[40] = 'al:3'
shared_node_layout.cpu[41] = 'al:4'
Listing 1: A sample virtual node mapping of simulation and analysis tasks to the 42 user-accessible cores and 6 GPUs of a Summit node. This example does not fully populate the node in order to evaluate internal interference effects. We place simulation tasks on the first ten cores of each socket and use two of the six GPUs, while analysis tasks are placed only on four cores of the second socket.
```
application programming interface (API) to describe concurrency and communications in their applications. However, MPI is an SPMD programming model. Applications that involve the ODAR motif, and indeed a growing number of other applications, can benefit from the ability to create multiple concurrent SPMD computations and to manage, monitor, and organize communications among those computations. Such multiple program multiple data (MPMD) computations have implications for both programming models and implementations. These needs motivated CODAR work on both MPI extensions—the MPI _Comm_launch_ of Wozniak et al. (2019)—and libraries for creating and coupling subcomputations (Mehta et al. 2019).

**Quantify performance of co-design choices**

The application developer may need to consider a variety of factors when evaluating co-design decisions, from computational performance to the quality of data produced by data reduction methods. To assist with such evaluations, we have developed two tools, Z-Checker and Chimbuko.

**Z-Checker: Quantify data reduction quality.** Whether a particular lossy compression method is appropriate for a given variable in a specific science application depends on many considerations: not only the compression speed and compression ratio, but also (depending on the application) other measures such as entropy, error distribution, power spectrum, and autocorrelation. To enable assessment of such factors in a systematic and convenient way, CODAR provides the Z-Checker tool (Tao et al. 2019), which can be used both offline, to produce a detailed report concerning the performance of specific compressor(s) on specific dataset(s), and also online.

The Z-Checker architecture is shown in Figure 4. Support for I/O libraries such as HDF5, NetCDF, and ADIOS2 make it easy to assess data in different formats. The analysis kernel applies a battery of analysis modules to a supplied data file to characterize both its inherent properties and the behavior obtained when a specific compressor is applied to its contents.

The data properties that Z-Checker characterizes are all closely related to compressibility, and enable deep understanding of how hard it is to compress supplied data. The reports characterize how well a specific compressor will perform on the data by quantifying more than 30 metrics garnered from user requirements across a wide range of domains.

A visualization engine, Z-server, can be used to display analysis results online, while libpressio (Underwood et al. 2020) provides a uniform and efficient interface to compressors, making it easy to integrate new (lossy or lossless) compressors into the Z-Checker framework.

**Chimbuko: Study performance of MPMD applications.** Performance measurement tools such as TAU (Shende and Malony 2006) are often used to diagnose performance problems in individual HPC applications with an SPMD structure. However, applications that implement the ODAR motif often have an MPMD structure and thus require performance analysis tools that can handle multiple, concurrent streams of performance data. It is also desirable for such tools to enable identification of performance anomalies in large trace data volumes, in order to avoid repeated performance analysis runs at exascale.

Inevitably, online data analysis and reduction of performance data is required, and thus not only do exascale ODAR applications require new measurement methods, but those new methods require ODAR techniques. If each core of an exascale system generates interesting trace events at just 1000 Hz, then on one million cores we need first to process one billion events per second and then to identify and communicate anomalies in this events to application scientists in understandable forms. The CODAR Chimbuko tool (Ha et al. 2020), a performance analysis framework, seeks to address these requirements. Building on TAU’s profiling machinery, Chimbuko implements distributed, in situ methods for detecting anomalies in trace event data generated on many nodes; see Figure 5. Chimbuko can reduce the overall cost of performance analysis by allowing developers to
Figure 5. Chimbuo distributed performance analysis and anomaly detection (AD) framework, with TAU profiling and ADIOS communication.

identify multiple trace data anomalies in a single run, rather than requiring multiple runs to study different performance hot spots. A visual analytics interface allows for the interrogation of performance anomalies at runtime (Ha et al. 2020; Xie et al. 2019).

Run co-design experiments

A common need in co-design experiments is to run the same application multiple times while varying parameters such as the compression method applied, the compression parameters, the computing resources used, and the mapping of application components to computing resources. This use case is essentially a specialized form of a parameter sweep, as supported, for example by Nimrod (Abramson et al. 1995), but with the parameters that are varied across experiments being concerned primarily with system configuration, rather than application inputs. Facing a need to run many such experiments, we developed the Cheetah and Savanna tools shown in Figure 6.

Cheetah’s specification format allows a user to provide a high-level description of a co-design campaign and target system(s); see Listing 2. Cheetah allows for monitoring of a campaign as it runs. Once a campaign completes, a performance generation engine can aggregate performance results from all experiments for user analysis.

Savanna ingests such a specification and manages the execution of the campaign, translating the specification into scripts and system scheduler calls that run the campaign and its experiments. It also provides the ability to compose workflows on different target platforms. By thus hiding low-level details, such as scheduler options for orchestrating process placements, these tools allow users to specify what they want to test, rather than how testing is performed.

Co-Design Case Studies

We use examples to illustrate the breadth of co-design questions that can be motivated by the ODAR motif.

Online Data Reduction and Analysis: Gray-Scott

This case study seeks to determine the best lossy compressor and parameters to use for data produced by the Gray-Scott reaction-diffusion benchmark code (Sims 2020), configured as in Yakushin et al. (2020). Figure 7 shows the $V$ variable after 10,000 timesteps for one system configuration; note the extreme variability in this slice through the output. An interesting aspect of this study is that the metric used to assess the quality of the compressed output is defined in domain-specific terms, namely that certain features are preserved in the compressed output. In particular, we use the Feature Tracking Kit (FTK) (Guo et al. 2020) to extract local maxima in the data.

The complete workflow, shown in Figure 8, comprises multiple components: output from the Simulation is passed both to FTK (FTK1) for feature detection and to a compressor (Comp); the compressed data is passed both to the store and to a decompressor (Decomp), which passes the decompressed data to a second FTK (FTK2); finally, a Check component compares...
class ReactionDiffusion(Campaign):
    name = "ReactionDiffusion"

    # DEFINE APPLICATIONS
    codes = [
        ("sim",
         dict(exe="prod.py", adios_xml_file='adios2.xml'),
        ("mean_calc",
         dict(exe="m_calc.py", adios_xml_file='adios2.xml')

    # CAMPAIGN SETTINGS
    supported_machines = ['local', 'theta', 'summit']
    kill_on_partial_failure = True
    run_dir_setup_script = None
    run_post_process_script = None
    scheduler_options = {'summit': {"project":''}}
    app_config_scripts = {'summit': 'env_setup.sh'}

    # PARAMETER SWEEPS
    sweep1_params = [
        # sweep over list values
        ParamRunner ('sim', 'nprocs', [2048,4096]),
        ParamRunner ('mean_calc', 'nprocs', [128]),
        ParamCmdLineArg ('sim', 'size_per_pe', 1,
                         ["1M","2M","4M"]),
        ParamKeyValue ('sim', 'steps','settings.conf',
                       ['steps', [10,25,50,100]]),
        ParamADIOS2XML ('sim', 'producer', 'engine',
                        ["{}"], # coupling
        ParamEnvVar ('sim', 'openmp', 'OMP_NUM_THREADS',
                      [1,4]),

        # Create a sweep
        sweep1 = Sweep (node_layout = {'summit': [shared_node_layout] },
                        # see Listing 1 for a node layout example
                        parameters = sweep1_params, rc_dependency=None
                      )

        # Create sweep group from above sweep.
        sweepGroup1 = SweepGroup ("sg-1", walltime=300, per_run_timeout=60,
                           parameter_groups=[sweep1], launch_mode='default' )

Listing 2: A campaign specification file. A Campaign is a collection of SweepGroups, which are collections of Sweeps. SweepGroups group experiments with similar characteristics: e.g., same number of nodes. A SweepGroup represents a batch job on the underlying system; associated properties specify allocation size, walltime limit, experiment timeout, etc.

dashed line), feedback to the compressor can be used to vary compression parameters if the compression quality is inadequate. For this example, the compression quality is the number of local maxima in the data.

Cheetah and Savanna were used to experiment with many different configurations, including SZ (Di and Cappello 2016; Tao et al. 2017; Liang et al. 2018; Zhao et al. 2020), ZFP (Diffenderfer et al. 2019), and MGARD (Ainsworth et al. 2020) as alternative compressors. Figure 9 shows some representative results, where we include the sign of the error. A positive error indicates that fewer features were detected (the compressor is smoothing the data), while a negative error indicates that more features were detected (the compressor introduces artifacts). As the compression ratio increases, it is not surprising that the error in the number of features detected increases. The SZ compressor in the configuration tested has negligible errors (~1%) for compression ratios less than 25 and a positive 10-20% error for a compression ratio of 35. Interestingly, in some regimes, MGARD introduces artifacts (negative error), while in other regimes it smoothes the data (positive error). SZ consistently acts as a smoother on this data set and quality metric, while ZFP introduces artifacts. Depending on the science to be performed with the compressed data, the end user might choose a compressor in part based on its ability to smooth the data, rather than add artifacts. The relationship between compressor, compression ratio, and nature of the compressor (smoothing and/or artifact introducing), however, is not apparent without the co-design study.

As a simulation advances in time, the nature of the compressed output and thus the compressor configuration required to achieve a particular threshold change. Online monitoring of compression quality with feedback to the compressor may be required to achieve the necessary output quality at each timestep.

Multi-physics Code Coupling: WDMApp

The development of a whole device model (WDM) is critical for the science of magnetically confined...
A complete WDM application (WDMApp) must couple multiple physics codes implementing different models and approximations (XGC, GENE, GEM), a separate coupler, and various online data analysis and reduction components. We report here on two co-design studies, the first aimed at understanding the appropriate fraction of available nodes to allocate to each code, and the best placement of the codes, and the second focused on understanding the cost of integrating a particular analysis directly into the XGC code. We also discuss data reduction issues.

**Coupling of simulation codes.** In this co-design study, we explore questions concerning the optimal allocation of resources for each component, placement of components on resources, and inter-component data exchange mechanisms. We use a simplified mini-app code composed of two instances of the XGC code, where we run one version to simulate the edge of the fusion reactor, and the other version to simulate the core. The only differences between this mini-app and the WDMApp are that the WDMApp can use the XGC, GENE, or GEM codes for the core and can use a separate coupler. Since we used ADIOS for the communication strategy, we can switch coupling approaches to exchange data via the GPFS storage system, node-local NVME (when the simulations are run on the same node), or ADIOS’s SST which uses RDMA for communication.

The WDMApp contains two types of ODAR multi-component application coupling, a fluid-based coupling and a kinetic-based coupling. In the fluid-based coupling (Dominski et al. 2018), data are exchanged in the overlap region of the codes: 3D charge density and potential fluctuations data, once per Runge-Kutta step (four times per simulation timestep in our implementation). One important co-design question concerns the placement of these two components on resources, which can have a profound effect. The coupled code can be over 20% faster when using a task-based graph embedding to optimize which MPI ranks are used by each code (Choi et al. 2017, 2018).

Analysis and visualization. WDM scientists commonly couple analysis codes to WDMApp to obtain physics results in near-real time, for example to detect instabilities that should cause a run to be halted. These analysis codes typically do not scale well to many processors and are commonly run on separate cores or nodes. Tracer particle analysis enables the detailed understanding of energy flux focused on the edge, including through the X-point region. The challenge is to reduce the cost of this analysis, which can often be over 30% of the entire runtime. In the left part

---

Figure 9. FTK error vs. compression ratio, Gray-Scott, Experiment 1, $V, step=10\,000$. From Yakushin et al. (2020).

Figure 10. WDM mini-app performance, demonstrating XGC edge-core coupling, on 32 Summit nodes with different node layouts (shared nodes and separate nodes) and ADIOS methods (GPFS and SST). The x-axis shows different node or core ratios assigned to the edge and the core application.
of Figure 11 we show the original version of XGC running with this analysis on 64 Summit nodes. Our co-design study was to understand if we could run the analysis on a separate set of nodes to reduce the overall cost of the analysis. Our co-design study showed that the “optimal” configuration was to use an additional 4 nodes, reducing the overall cost from 31% to approximately 6%. Since the analysis routine requires a large amount of communication, we find that the benefits of this approach increase as XGC is scaled to a larger number of nodes.

**Figure 11.** The XGC code contains an analysis routine that advects tracer-particles in the plasma to understand the flow around the different sections of the fusion reactor. This co-design study, on Summit, showed that moving the analysis from the 64 XGC nodes to four additional nodes reduced overall cost by 31%, from 64 nodes × 76 sec = 4864 node-sec to 68 nodes × 49 sec = 3332 node-sec. The y-axis is time per XGC step, and the ADIOS times are for moving ~10 GB state between XGC and the analysis.

**Data reduction.** XGC codes can produce 100 PB in a week-long simulation on pre-exascale computers, with output taking about 4% of the overall run time. Both numbers will increase considerably on exascale systems. Thus we encounter co-design questions concerning the tradeoffs associated with writing data uncompressed versus writing compressed data in terms of runtime requirements for the simulation code, I/O volumes, and accuracy implications for common analysis routines run by WDMApp scientists. The CODAR and WDMApp teams are currently investigating these questions.

**Online Data Analysis: NWChem**

The NWChemEx ECP project is targeting a range of computational chemistry methods, from molecular dynamics to high-order, many-body methods. A key focus of their exascale code demonstrations are large (a million atoms or more), dynamic biomolecular processes. In molecular dynamics (MD), these biomolecular complexes are simulated over microsecond timescales, corresponding to 1B timesteps or more in the simulation. Outputting all trajectory data for 1M atoms over 1B timesteps would generate 32 PB of output. Therefore, implementations of the MD method have traditionally changed the frequency with which they output conformations along the trajectory to reduce the data volume, such as outputting only 1 out of every 100 timesteps. However, that frequency would still generate 320 TB of output, and, furthermore, may miss key phenomena. Thus online data analysis and/or more intelligent reduction are of interest.

We describe here three co-design studies involving ODAR motifs in NWChemEx. The first is concerned with accelerating, but not reducing, the output process by identifying efficient mappings of sorting, output, and principal component analysis (PCA) tasks to processing nodes. The second investigates an online adaptive sampling method that uses machine learning to identify moments in an MD trajectory where the structure of the molecule undergoes significant changes. The third demonstrates the massive reduction in performance data achieved by storing only anomalous events. As NWChemEx development was in progress, all studies used NWChem (Valiev et al. 2010) as a proxy.

**Online data analysis: Sorting and PCA.** A first study investigated methods used to output molecular dynamic simulation state. Scientists generally want to record periodically the spatial coordinates for every atom in a molecule. This information is written to disk in a table format, one row per atom. In order to track individual atoms over time, the table needs to be sorted by atom ID, which we can think of as a simple form of analysis that we would prefer to do online to simplify later activities.
A first co-design question is how best to organize the sorting and output step. We have two components (the simulation and a sort/output component) that are coupled via the atomic coordinates produced by the simulation and consumed by the sort/output component. As the sort can run concurrently while NWChem is computing, we identify as free parameters the choice of whether to run the simulation and sort operations on the same or different nodes, and how many nodes (or cores) to allocate to each.

To explore possible mappings, we require performance data. We start by measuring the performance of the conventional NWChem code, which sends all atoms to one core for sorting and writing. The first, second, and fifth columns of Figure 12 show relevant results. The first column gives the time for a 1000-step 36 536-atom simulation without sort and output: 188 secs ($t_{nw}$=0.188 sec/step) on $N_{nw}$=224 cores (14 nodes) of Rhea, an OLCF cluster with 16-core, InfiniBand-connected x86 nodes. The second column gives the time taken when sorting and output is enabled on the same 224 cores. Total output is 2.3 MB of atom state per step (2.3 GB over 1000 steps), but total time increases to 386 seconds, suggesting a time for sorting and output of $t_{sort}$=0.198 sec/step on 224 cores. The last column shows, for reference, the time taken when using the same strategy, but outputting only every 25 steps, which produces an overhead of 4%.

We then explored an alternative design that runs the NWChem simulation and sorting and output components on distinct processors. On 224 cores, simulation takes 0.188 sec/step and sorting and output takes 0.198 sec/step, which are roughly comparable. But clearly we should be able to sort the 2.3 MB of atom state in less than 0.198 secs if fewer cores are involved. We thus investigated gathering the atoms to a small number, $N_{sort}$, of separate cores, and sorting the atoms on them. The latter approach incurs the additional costs of gathering the atoms and requires allocating the dedicated cores, but avoids the need for a highly distributed sort. Moreover, the sort can be performed in parallel with other NWChem computation. Assuming that the time for the offloaded sort, $t_{sort}$, is less than that for the NWChem computation, then total core-sec are reduced from $(t_{nw} + t_{sort}) \times N_{nw}$ to $t_{nw} \times (N_{nw} + N_{sort})$.

Using Cheetah to perform runs in which we varied $N_{sort}$, we determined that allocating two extra cores, to which all unsorted atoms were sent and on which atoms were sorted and then written to disk, reduced the total runtime to 190s (0.190s/step), as shown in the third column in Figure 12. In other words, we can sort and store every atom position at every step with insignificant application slowdown, and only increasing the total computational costs from $188 \times 224 = 42112$ core-sec with no output to $194.7 \times 224 = 44002$ core-sec—in contrast to $386 \times 224 = 86464$ core-sec for the traditional inline sort.

Next we consider an analysis that is commonly applied to atom trajectory data, namely principal component analysis (PCA). An example motivation for this capability is shape analysis, which can be used to discover the interesting movements of a molecule by filtering out uninteresting wiggles. To this end, we developed an in situ, motion-adjusted PCA analysis routine within the pbdR package (Schmidt et al. 2017). This routine runs PCA repeatedly on a fixed size window of 10 steps.

Having added PCA to the application, we compared performance obtained for the same ~1M atom problem, once again on Rhea. We find, as shown in the fourth column of Figure 12, that the PCA can run on a single additional core without slowing down the overall computation significantly, for a total computational cost of $197.6 \times 224 = 44851$ core-sec. Overall, by allocating three extra cores for sorting, running PCA, and writing out the data, we are able to run NWChem with just a 6% overhead relative to running NWChem without sort, output, or PCA.

An additional benefit of this strategy of running analyses concurrently with NWChem is that there is still idle time on the additional nodes, which can then be used for running visualization. In the future, we will extend our work to include online visualization, subtracting off the average motion from the molecule to understand its motion.

**Online adaptive sampling using machine learning.** MD simulations of biomolecules tend to spend much time moving only slightly in conformation space, with occasional transitions to other parts of that space. For many scientific purposes, it is these transitions that are interesting, as biochemistry is regulated by conformation changes of proteins: molecular structures at other times are uninteresting. Thus a method for identifying distinct structures can both provide insights into chemical processes and greatly reduce the amount of data to stored.

To this end, we developed a machine-learning method—specifically, a weighted reservoir sampling algorithm (Efraimidis and Spirakis 2006)—that can be applied to streaming atom trajectory data to detect changes in molecular structure. To stably measure such changes in the presence of vibrational noise from the atoms, we used matrix sketching (Zhang et al. 2018) to compute low-dimensional embeddings of the trajectories, and then output conformations.
Figure 13. One-dimensional embedding values via PCA (blue) for a 5000-timestep trajectory where a protein first collides (first change) and then binds (second change) with DNA. The red points (y values have no significance) indicate the 32 configurations selected for output by online sampling.

only at time when the low-dimensional embeddings show significant changes. As shown in (Yoo et al. 2016), a conventional non-streaming low-dimensional embedding calculation requires $O(m^2 \cdot n_t)$ operations, where $m$ is the number of atoms and $n_t$ is the time window; our streaming approach requires only $O(m \cdot n_t)$ operations, i.e., it is linear in the number of atoms.

Figure 13 shows how this method can greatly reduce output while retaining key molecular structures from the simulation data. While storing all trajectory data for this 5000-step, 8993-atom system would take 2.3 GB, our adaptive online analysis stores only 32 configurations, or 15 MB: a reduction of $156 \times$. For longer simulations, even fewer conformations may be required, as “interesting” events tend to occur infrequently. For example, in a longer, 10 000-step simulation, the adaptive online analysis identified only 37 conformations (17 MB), only five more than in the 5000-step simulation, and a reduction of $270 \times$ from the full output of 4.6 GB.

Performance data reduction. We used the Chimbuko tool described earlier to study the performance of different NWChem data analysis solutions and implementations. We first configured the NWChem workflow with TAU to generate function execution events and with Chimbuko to flag anomalously long function execution times, defined as those greater than two standard deviations from the mean. Then, we ran a number of different sized use cases, the largest being a 1.2M atom simulation on 2560 MPI ranks (128 nodes) of Summit. A total of 117.5 GB of raw data of performance trace events were generated during a ~140 sec NWChem run, which Chimbuko reduced to 5.5 GB of anomalous events with 97% accuracy. Analysis of these anomalies revealed that the barrier synchronization used in NWChem was causing excessive delays (Pouchard et al. 2018), an observation that has motivated the replacement of the block synchronous implementation with a put-notify driven implementation to remove all barriers.

Deep Learning Workflows: CANDLE

The Cancer Distributed Learning Environment (CANDLE) ECP is developing methods and software to support scalable deep learning on supercomputers, with a particular focus on cancer research applications (Wozniak et al. 2018c). Many CANDLE applications involve large numbers of quasi-independent learning and inferencing tasks, each of which can typically be run on a single node of a supercomputer. While each such task may generate only modest amounts of data, large data volumes can be produced in aggregate across the many tasks, leading to a need for online data analysis and reduction.

As an example, the CANDLE hyper-parameter optimization (HPO) workflow (Wozniak et al. 2018b) generates, and evaluates via training and testing, many single-node neural network (NN) configurations that differ along such dimensions as number of layers, number of neurons per layer, and activation functions. A typical HPO workflow runs many instances of the NN training process at once, selecting parameter values to explore via methods such as exhaustive search, grid search, random search, simple gradient descent, and multiple gradient descent. Regardless of the method used, many NN configurations are evaluated, each characterized by their quality measure(s), such as the validation error on a testing set, and the network weights produced during training.

The need for online data analysis and reduction in HPO arises from the large number of outputs produced and the benefits that can accrue from analyzing those outputs online rather than offline. If $M$ is the average size of a NN state (configuration plus network weights) and a computer has $N$ nodes, each able to evaluate a configuration in $T$ seconds, then data can be generated at a rate of $M \times N / T$ bytes/second. While $M$ and $T$ vary greatly across HPO applications, aggregate data rates can easily reach 10s of GB/s on today’s computers and TB/s on exascale systems. Therefore, the application scientists would strongly prefer to avoid writing this data to disk only to have to read them later for analysis to select the interesting NN configurations.

Reduction is thus desirable, but turns out to be not straightforward. While the vast majority of NN states can normally be discarded because they are
uninteresting, determining whether or not a state is interesting can be non-trivial. For example, the HPO process wants a set of states that are diverse along multiple dimensions, rather than just those with the best validation error. Thus, to support online filtering of states we want, in general, to collect many states as the HPO proceeds, analyze those outputs as they are produced, and selectively prune uninteresting states over time, ultimately outputting only the most interesting ones. Thus HPO workflows require, in general, the ability to maintain, perform random access on, and compute over large quantities of data.

We have worked with CANDLE to define a general architecture for incorporating ODAR methods into HPO. As shown in Figure 14, HPO workers and analysis routines are coupled via a shared Model Cache (Wozniak et al. 2018a) based on DataSpaces (Docan et al. 2012), a distributed, in-memory storage system. DataSpaces provides a highly scalable implementation of a tuple-space programming model (Carriero and Gelernter 1989) that permits data producers to insert tuples (essentially, key-value pairs), and data consumers to both retrieve tuples and subscribe to notifications of new tuples with specified properties. The DataSpaces implementation is designed to scale across many nodes and to enable high-speed data access, so that data produced by one application component can be efficiently indexed and then asynchronously accessed and processed by other components. In HPO, the data being written to the Model Cache are the NN states, and the components that retrieve and process those states are analysis routines that may compute summary statistics, eliminate uninteresting configurations, and/or compress configurations before outputting them.

This general architecture can be configured in different ways, depending on the characteristics of a specific HPO application. Experiments to date have used synthetic workloads. For example, Figure 15 shows results from an experiment in which an increasing number of nodes, organized in a CANDLE-like workflow, stores and retrieves 1 MB files in either a parallel file system (PFS) or the Model Cache (implemented here by a single DataSpaces server). We see that the Model Cache delivers significantly better performance than does PFS.

As the number of nodes increases, the rate between the Model Cache and PFS implementations seems to be converging. This convergence indicates a need for further co-design studies involving multiple Model Cache servers to determine the optimal number of Model Cache servers as a function of the number of nodes and other parameters, the placement of Model Cache servers across nodes, and the mechanisms by which Model Cache servers coordinate.

**Lessons Learned**

Our co-design experiences in ECP motivate the following cross-cutting observations about the power of co-design, the ODAR motif, and interactions between the co-design process and motif.

**Co-design process:** It is good practice to use models to inform the design and workflow optimization process, as a grid search over a large design space is computationally intensive and inefficient, and a coarse discretization of the search space can accidentally remove optimal designs that a model-driven, analytic plan would find. While a precise analytical model will often be impractical, the process of constructing and refining underlying models and applying a systematic,
model-driven co-design process is critical, precisely because the co-design spaces are extremely large.

**ODAR as motif:** High-end computational science has been dominated for several decades by powerful monolithic applications. Science investigations that address more complex scenarios require support for modular, multi-physics integration (Foster and Kesselman 2006). The ODAR motif offers a programmatic approach for addressing these needs. Integrating ODAR into an existing, large code base, however, can be difficult and expensive. Therefore, we see power in considering ODAR as a first-class motif of extreme-scale applications in order to achieve both software management and performance goals.

**Tools aid the process:** In HPC, we must deal with components (application software, libraries, systems software, and architecture and hardware) that are developed at different times and frequently on different time scales. We may be able to modify an application in an afternoon, but changes to systems software can take months (ad hoc modifications to external libraries are antithetical to long-term software lifecycle management), and hardware changes years. Moreover, managing and tracking changes over time can be extremely difficult. Thus, we have found a need for tools like Cheetah, Savanna, Z-Checker, and Chimbuko that enable consistent and verifiable testing capabilities within co-design campaigns. Such campaigns are rarely circumscribed investigations, but instead stretch over multiple application versions, hardware changes, extensions to cover new algorithms, and so on. Managing a co-design campaign so that one can continue design exploration over the entire lifecycle of an application can significantly reduce costs and improve the quality of the results.

**Co-design of ODAR workflows as a new model:** A strength of today’s HPC ecosystem is that its performance optimization tools work well in improving the performance of individual applications. However, independent optimization of individual components is rarely sufficient for optimizing a multi-component ODAR system, due to the need to account for interactions and interference among components. Exhaustive search of all possible configurations is rarely feasible, especially given resource limitations. The co-design of ODAR workflows offers a strong model for achieving the new science and high performance needed in the future. This unified software development and execution motif addresses the need for model-driven optimization choices, while also making explicit the reasons for those optimizations so that the future evolution of high performance science can continue to benefit from co-design study insights as the HPC landscape changes.

**Conclusions**

The online data analysis and reduction (ODAR) motif is becoming increasingly important in HPC applications due to changes in both systems and applications. Efficient implementation of applications that implement this motif can raise challenging co-design questions, due to tradeoffs that must be navigated between simulation performance, data reduction and analysis performance, output data volume and quality, and the capabilities of different applications and HPC system components, among other factors. We have described an ODAR co-design process that we have found useful for answering these questions. This co-design process is assisted by tools developed by the CODAR ECP project, often in partnership with other ECP projects, such as ADIOS, ExaLearn (Alexander et al. 2020b), ExaWorks, SZ, ZFP, and TAU, to facilitate experimentation with design alternatives (e.g., different mappings, communication structures, reduction methods) and to evaluate the data quality and computational performance achieved by these alternatives.

The case studies presented here illustrate some of the ways in which ODAR methods and tools are being applied to accelerate ECP applications. ODAR tools can also be used in other ways. For example, the fusion whole device modeling project’s EFFIS coupling framework (Suchyta et al. 2020) uses Savanna to place components on nodes and processors, Z-Checker is used to evaluate compression methods in many contexts, and Chimbuko is used to study the performance of various ECP applications. The co-design of ODAR workflows remains a fruitful activity that will benefit applications far into the future.

**Acknowledgements**

We thank ECP leadership and the many ECP colleagues with whom we have interacted in performing this work.

**Funding**

This article reports on work supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and National Nuclear Security Administration. This research used resources of the Argonne and Oak Ridge Leadership Computing Facilities and NERSC, DOE Office of Science User Facilities supported under Contracts DE-AC02-06CH11357, DE-AC05-00OR22725, and DE-AC02-05CH11231, respectively.
References


730–739.


Foster I (1994) Designing and Building Parallel Programs. Addison-Wesley.


Author biographies

Ian Foster is Senior Scientist and Distinguished Fellow, and director of the Data Science and Learning Division, at Argonne National Laboratory, and the Arthur Holly Compton Distinguished Service Professor of Computer Science at the University of Chicago.

Mark Ainsworth is the Francis Wayland Professor of Applied Mathematics at Brown University.

Julie Bessac is an Assistant Computational Statistician in the Mathematics and Computer Science Division at Argonne National Laboratory.

Franck Cappello is a Senior Computer Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory.

Jong Choi is a Scientist in the Scientific Data Group, Computer Science and Mathematics Division, Oak Ridge National Laboratory.

Sheng Di is a Computer Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory and a Scientist-At-Large at the University of Chicago.

Zichao Di is a Computational Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory.

Ali Murat Gok is a postdoctoral appointee at Argonne National Laboratory, contributing to the ECP VeloC/SZ (scientific data compression) and CODAR (data analysis and reduction) projects.

Hangi Guo is an Assistant Computer Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory.

Kevin A. Huck is an Faculty Research Associate at the University of Oregon.

Christopher Kelly is a Scientist at Brookhaven National Laboratory.

Scott Klasky is a distinguished scientist and group leader for Scientific Data in the Computer Science and Mathematics Division at Oak Ridge National Laboratory.

Kerstin Kleese van Dam is Director of the Computational Science Initiative at Brookhaven National Laboratory.

Xin Liang is a Computer Scientist in the Computer Science and Mathematics Division at Oak Ridge National Laboratory.

Kshitij Mehta is a Computer Scientist in the Scientific Data Group in the Computer Science and Mathematics Division at Oak Ridge National Laboratory.

Manish Parashar is a Distinguished Professor of Computer Science at Rutgers University.

Tom Peterka is a computer scientist at Argonne National Laboratory, a scientist at the University of Chicago Consortium for Advanced Science and Engineering, a fellow of the Northwestern Argonne Institute for Science and Engineering, and an adjunct assistant professor at the University of Illinois at Chicago.

Line Pouchard is a Senior Researcher in the Computational Science Initiative at Brookhaven National Laboratory.

Tong Shu was a Postdoctoral Appointee in the Data Science and Learning Division at Argonne National Laboratory and is currently an Assistant Professor at Southern Illinois University Carbondale.

Hubertus Van Dam is an Application Architect in the Computational Science Initiative at Brookhaven National Laboratory.

Matthew Wolf is a Senior Computer Scientist in the Computer Science and Mathematics Division at Oak Ridge National Laboratory.

Justin Wozniak is a Computer Scientist in the Data Science and Learning Division at Argonne National Laboratory and a Scientist-At-Large at the University of Chicago.

Igor Yakushin is a Computational Scientist in the Data Science and Learning Division at Argonne National Laboratory. In a previous job, he worked on the online search for gravitational wave bursts in the LIGO project.

Wei Xu is a Computer Scientist in the Computational Science Initiative at the Brookhaven National Laboratory.
Shinjae Yoo is a Computational Scientist in the Computational Science Initiative at Brookhaven National Laboratory.

Todd Munson is a Senior Computational Scientist at Argonne National Laboratory, a Senior Scientist for the Consortium for Advanced Science and Engineering at the University of Chicago, and the Software Ecosystem and Delivery Control Account Manager for the Exascale Computing Project.