


Online data analysis and reduction: An important Co-design motif for extreme-scale computers

The International Journal of High
Performance Computing Applications
2021, Vol. 0(0) 1–19
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211023549
journals.sagepub.com/home/hpc


Ian Foster^{1,2} , Mark Ainsworth³, Julie Bessac¹, Franck Cappello¹, Jong Choi⁴, Sheng Di¹, Zichao Di¹, Ali M Gok¹, Hanqi Guo¹, Kevin A Huck⁵, Christopher Kelly⁶, Scott Klasky⁴, Kerstin Kleese van Dam⁶, Xin Liang⁴, Kshitij Mehta⁴, Manish Parashar⁷, Tom Peterka¹, Line Pouchard⁶, Tong Shu^{1,8}, Ozan Tugluk³, Hubertus van Dam⁶, Lipeng Wan⁴, Matthew Wolf⁴, Justin M Wozniak¹, Wei Xu⁶, Igor Yakushin¹, Shinjae Yoo⁶ and 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

Data analysis, in situ, exascale computing, online data analysis and reduction

1. 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). While applications have changed a great deal since 1965, the fact that exascale computers will support I/O rates of little more than 10^{10} B/s is striking. 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 high-performance computing (HPC) applications and HPC systems. Application developers must carefully balance the costs and information content that results when data are produced via different methods. High-performance computing system architects need to consider ODAR concerns

¹Argonne National Laboratory, Lemont, IL, USA

²University of Chicago, Chicago, IL, USA

³Brown University, Providence, RI, USA

⁴Oak Ridge National Laboratory, Oak Ridge, TN, USA

⁵University of Oregon, Eugene, OR, USA

⁶Brookhaven National Laboratory, Upton, NY, USA

⁷Rutgers University, New Brunswick, NJ, USA

⁸Southern Illinois University, Carbondale, IL, USA

Corresponding author:

Ian Foster, Data Science and Learning Division, Argonne National Laboratory, 9700 S Cass, Lemont, IL 60439, USA.

Email: foster@anl.gov

when designing and configuring programming models, libraries, runtime systems, and storage systems. Understanding and addressing these concerns require 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 Online Data Analysis and Reduction (CODAR) (Foster et al., 2017), part of the US Department of Energy's Exascale Computing Project (ECP), has worked closely with ECP applications (Alexander et al., 2020a) and technology projects on ODAR co-design questions.

In the sections that follow, we (1) introduce the ODAR motif, its importance for HPC, 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.

2. Background

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

Our focus in this article is an important subset of the entire ODAR design space—coupling simulations to analysis and reduction routines via adaptations that allow their current I/O to be made available for other applications to use for online reduction, analysis, and two-way coupling. In all cases discussed in this article, the analysis and reduction routines could be run either online or off-line. Thus, only minimal changes were required to the simulation and analysis codes.

Other optimizations not considered in this article, such as zero-copy coupling, can require that data structures and concurrency levels in all codes match. Requirements can also be substantially different for computational steering and when machine learning methods incorporated into a simulation require high-frequency analysis and feedback. Such challenging problems that may require substantive changes to the application and simulation code are outside the scope of this article.

2.1. 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). Because of difficulties in running multicomponent 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 *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?

2.2. 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 *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

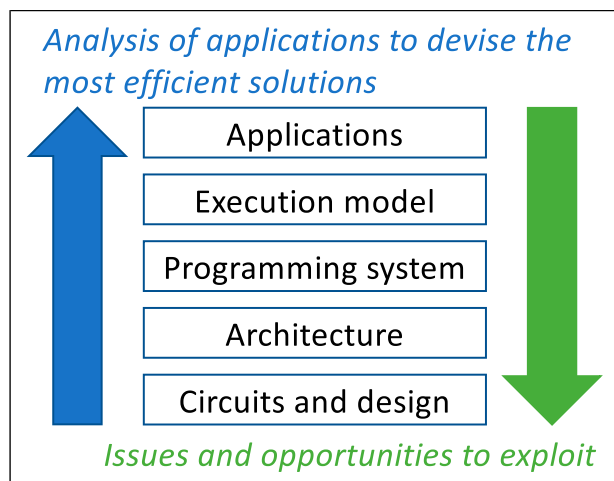


Figure 1. Co-design process (Barrett et al., 2013).

computer but with developing the 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 of full-stack design that we use the term in this article.

3. Online data 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 either too large or the required turnaround time in terms of analysis response is too short for the data to be exchanged off-line via the file system. Here and elsewhere, *Simulate* and *Analyze* can be any programs that generate and consume data, respectively, either may *both* generate and consume data, such as when two-way coupling is required. When the file system is the bottleneck, the application developer must either reduce the size of the data prior to writing it to storage 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 differently.

The ODAR motif can be instantiated in many ways, each having unique requirements. Thus, a single application may involve one or more of the components and communication paths shown in Figure 2. We now describe two important use cases of the ODAR motif that provide a frame of reference for co-design opportunities.

3.1. Multicomponent 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

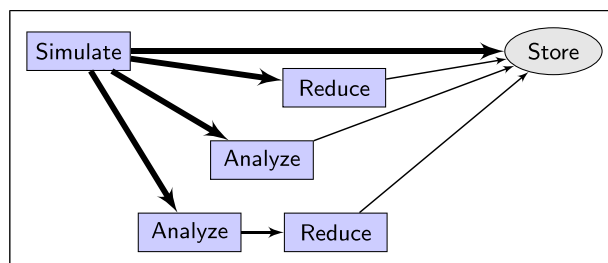


Figure 2. Some possible couplings between *Simulate*, *Analyze*, and *Reduce* components.

the latter case, the number and duration of instances may be fixed or vary over time. For example, different application instances may explore different parts of a complex phase space of molecular dynamics (MD) trajectories; periodically, results are aggregated and compared, non-interesting trajectories discarded, and new trajectories initiated (Lee et al., 2019).

In multicomponent applications, each component may produce results asynchronously, leading to a need to cache 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 analysis 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).

3.2. Two-way coupling

Another example 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 cases can require rapid data exchange between components and thus make optimizing the data layout and movement among components particularly important.

4. Perspectives on ODAR and co-design

The ODAR motif is a large, complex space, requiring a co-design process to answer application relevant questions. We structure our discussion of ODAR co-design questions in terms of four perspectives.

4.1. 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 reread data)—all with the goal of maximizing the useful information stored.

High-performance computing applications often address these issues via simple heuristics, such as computing statistical summaries of some fields and performing temporal or spatial decimation in others. Such simple approaches may be inefficient, however, and can easily hide important phenomena, especially as the disparity between computer speed and I/O rates grows. Alternatively, lossless compression of scientific data can be used, although this method is rarely effective because the compression ratios are typically small (Ratanaworabhan et al., 2006), while lossy compression comes at the risk of potentially losing information or introducing systematic errors.

4.2. 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 components on the same resources, in an SPMD structure, or on different resources MPMD? How should resource types (CPUs and GPUs) be allocated to components? Should certain components be placed on a single node (simplifying implementation, but limiting performance) or on multiple nodes? Different choices may make different intra- and intercomponent communication mechanisms available, each with different performance characteristics (Choi et al., 2018; Malakar et al. 2015, 2016, 2018).

4.3. HPC systems perspectives

The ODAR motif also has implications for HPC programming models, libraries, system software, and other HPC system components. These elements have historically been designed to support primarily SPMD computations, in which every computer node assigned to an application runs the same program. ODAR applications can benefit from the ability to create multiple SPMD computations; manage the placement of application components; and manage, monitor, and organize communications among those computations.

The ODAR motif also has implications for HPC hardware and architecture. For example, nonvolatile 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 trade-offs is important for both evaluating portability and designing future machines.

4.4. Software engineering perspectives

The ODAR motif also has implications for software engineering. In keeping with principles of information hiding in modular design (Parnas 1972), we want to avoid committing prematurely to design decisions 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.

Data layout choices can have important implications for performance when coupling applications (Subhlok et al., 1993) as when the producer and consumer of a data structure prefer different layouts. In all cases examined in this study, we treat data layouts within individual applications as fixed and address any discrepancies during intercomponent communication. However, in the general case, it can be desirable to be able to vary data layouts within each application component as part of ODAR co-design.

5. ODAR co-design process

Co-design questions are concerned with understanding trade-offs among different elements of a multicomponent 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 expose such dependencies, build understanding of their implications, and avoid expensive mistakes such as locking in to a bad design of one component because of inadequate understanding of dependencies.

5.1. ODAR co-design challenges

Online data analysis and reduction 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; Hoefer 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 trade-offs 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 timescales 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.

5.2. 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 mean that rarely can one develop accurate models of all possible design alternatives. Usually, however, one can identify both major design decisions (e.g., online or off-line analysis, data reduction or not, and co-locate or distribute components) and minor design decisions (e.g., mapping to cores and communication mechanisms) and then explore the associated trade-offs in an organized fashion, as depicted in Figure 3.

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 and 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

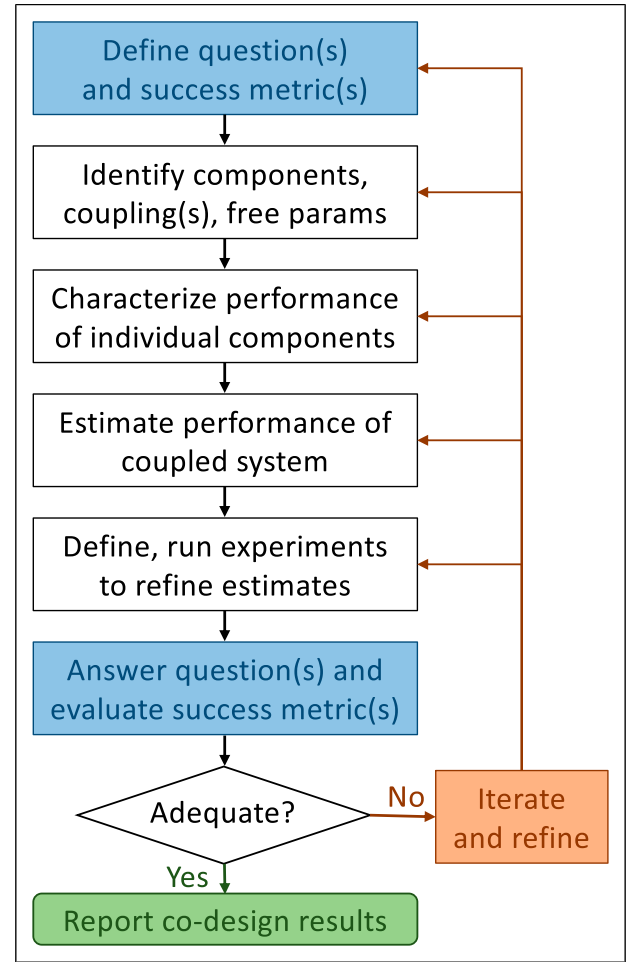


Figure 3. Online Data Analysis and Reduction co-design process, described in the text, involves the iterative application of multiple steps.

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 step may 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 identifying advantageous couplings. For example, if simulation and reduction run concurrently on different node subsets, we may expect the overall execution time to be the maximum of the simulation and reduction time on those node subsets, plus time for coupling communication. If they run consecutively on the same nodes, we may expect the overall execution time to be the maximum, across all nodes, of

the sum of the simulation and reduction times. Based on such 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 a 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 or refining performance estimates or models for individual components.

6. Co-design technologies

Online data analysis and reduction 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 performance implications of co-design choices (e.g., by quantifying the numerical properties of a data reduction method or by collecting and analyzing computational performance data), and running sets of co-design experiments. We describe here technologies that we have developed for these purposes.

6.1. Expose 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 trade-offs 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.

6.1.1. 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 processors 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, no equivalent standardization of process mapping constructs exists. 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.

6.1.2. Manage communication methods. Modern HPC systems support a wide variety of interprocess communication mechanisms with different performance characteristics, including MPI, remote direct memory access (RDMA), transmission control protocol/internet protocol, 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 (Godoy 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 the application code, to access a file system or to communicate over the network via

```
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']
shared_node_layout.gpu[3] = ['sim:10', 'sim:11']

# 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. 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 10 cores of each socket and use two of the six GPUs, while analysis tasks are placed only on four cores of the second socket.

different mechanisms. ADIOS2 libraries implement these different behaviors, for example, via the Sustainable Staging Transport (SST) engine, which uses RDMA mechanisms, and Strong Staging Coupler engine, which uses MPI methods for communications.

Application programmers who want 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 (Diffenderfer et al., 2019), and MGARD (Ainsworth et al., 2019), each packaged to expose tunable parameters such as error tolerance.

As noted earlier, multicomponent 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. Co-design center for Online Data Analysis and Reduction 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.

6.1.3. Programming models and libraries. High-performance computing programmers often use the MPI 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 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).

6.2. Run co-design experiments

A common need in co-design experiments is to run the same application multiple times while varying parameters such as compression method, compression parameters, computing resources, and mapping of application components to computing resources. This use case is essentially a 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 4.

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.

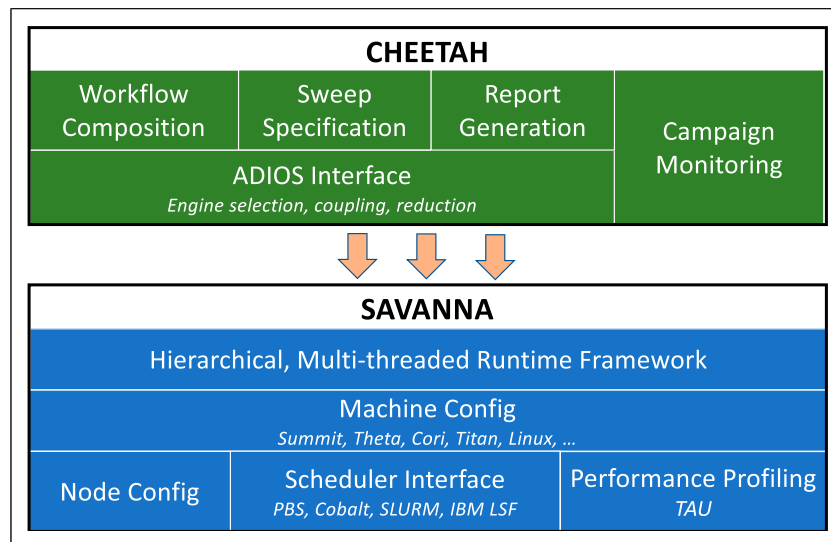


Figure 4. Co-design center for Online Data Analysis and Reduction co-design campaign framework. Cheetah supports the specification of a campaign; Savanna executes a campaign on a target system. From Mehta et al. (2019).

```

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',
                        [ {'SST': {} } ], # 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: for example, the same number of nodes. A SweepGroup represents a batch job on the underlying system; associated properties specify allocation size, wall-time limit, experiment timeout, and so forth.

6.3. 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.

6.3.1. 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 compression speed and

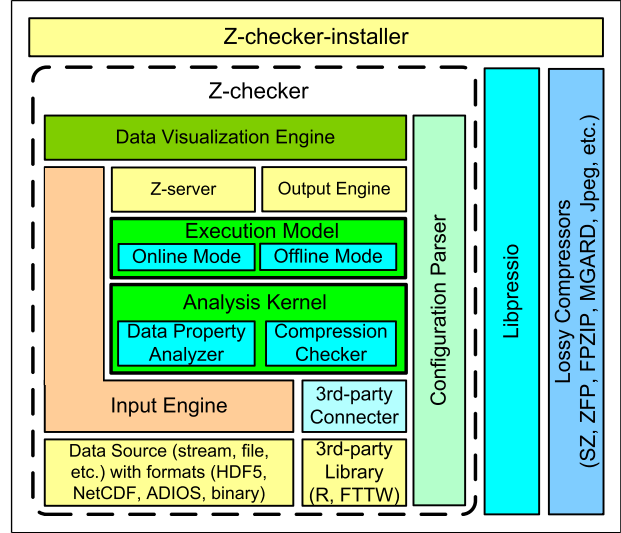


Figure 5. Modular and extensible Z-Checker architecture.

compression ratio but also (depending on the application) measures such as entropy, error distribution, power spectrum, and autocorrelation. To enable the systematic and convenient assessment of such factors, CODAR provides the Z-Checker tool (Tao et al., 2019), which can be used both off-line, to produce a detailed report concerning the performance of a specific compressor on specific dataset, and online.

The Z-Checker architecture is shown in Figure 5. Support for I/O libraries such as HDF5, NetCDF, and ADIOS2 makes 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.

6.3.2. 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

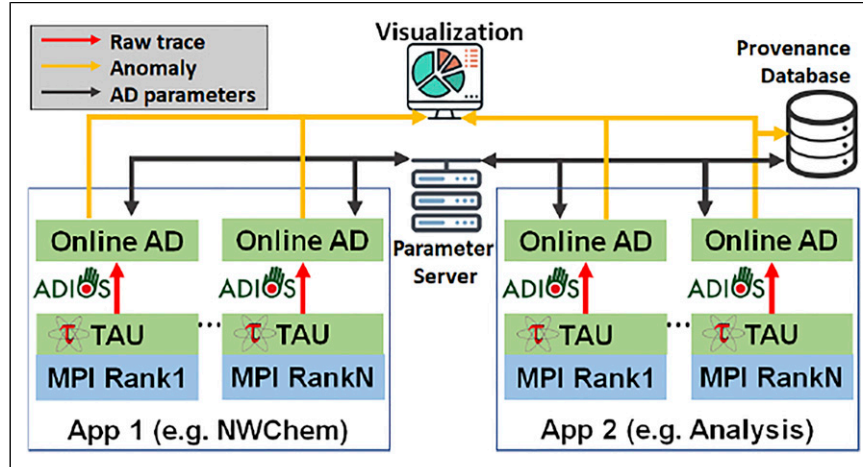


Figure 6. Chimbuko distributed performance analysis and anomaly detection (AD) framework, with TAU profiling, and ADIOS communication.

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 are 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 1 million cores we need first to process 1 billion events per second and then to identify and communicate anomalies in this event 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 6. Chimbuko can reduce the overall cost of performance analysis by allowing developers to 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).

7. Co-design case studies

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

7.1. Online data reduction and analysis: Gray-Scott

Our first study uses the Gray-Scott reaction–diffusion benchmark code (Sims 2020) to determine the best lossy compressor that preserves a domain-specific quantity of

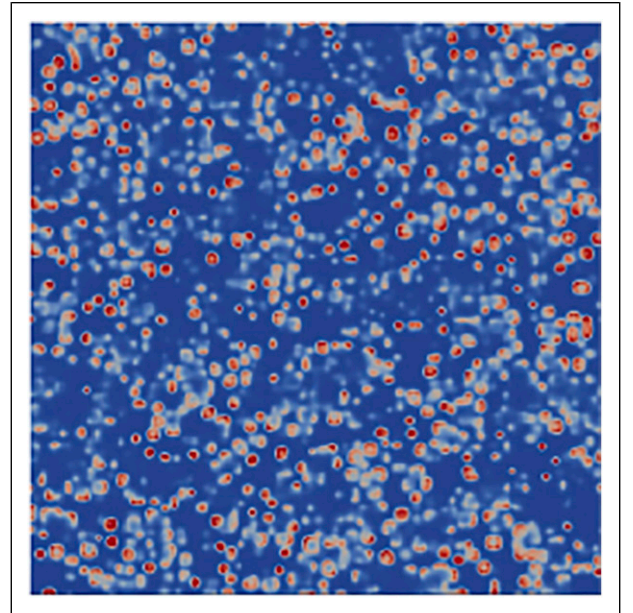


Figure 7. Gray-Scott V variable after 10,000 steps: 2D slice at $X = 128$, Experiment I. From Yakushin et al. (2020).

interest. This situation mimics the requirements of application scientists to meet their science objectives when using lossy compressors. Depending on the science to be performed with the compressed data, the scientist might choose a compressor in part based on its ability to smooth the data, rather than add artifacts.

The Gray-Scott case study was configured as in Yakushin et al. (2020), and our goal is to compress the simulation output, while preserving local maxima extracted using the Feature Tracking Kit (FTK) (Guo et al. 2020; Guo et al. 2021). Figure 7 shows the V variable after 10,000 timesteps

for one configuration and shows the extreme variability we need to preserve in the compressed data.

The general workflow, shown in Figure 8, comprises multiple components: output from Simulate is passed both to FTK (FTK1) for feature extraction and to a compressor (Comp); the compressed data are passed both to the store and to a decompressor (Decomp), which passes the decompressed data to a second FTK (FTK2); a Check component then compares the features obtained from the original and decompressed data. With lossless compression, the features obtained would be identical. In this co-design study, the Check component's output is written to disk. 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 (as shown by the dashed line) may be required in order to achieve the necessary output quality at each timestep.

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. Application-specific compressors, such as a physics-based filter and an algorithm that regenerates the simulation state, could also be used. Figure 9 shows 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, the number of features detected increases. In some regimes, MGARD introduces artifacts (negative error), while in other regimes, it smooths the data (positive error). SZ consistently acts as a smoother on this dataset and quality metric, while ZFP introduces artifacts. The relationship between compressor, compression ratio, and nature of the compressor (smoothing and/or artifact introducing), however, is not apparent without a co-design study that cannot be performed at scale without using the ODAR motifs.

7.2. Multiphysics code coupling: WDMApp

The development of a whole-device model (WDM) is critical for the science of magnetically confined fusion plasmas (Aymar et al., 2002). The WDMApp application being developed in ECP couples multiple physics codes implementing different models and approximations (XGC, GENE, and GEM), a separate coupler, and data analysis and reduction components. We report on two co-design studies, one aimed at understanding the appropriate fraction of available nodes to allocate to each code and alternative code placements, and the second investigating approaches to

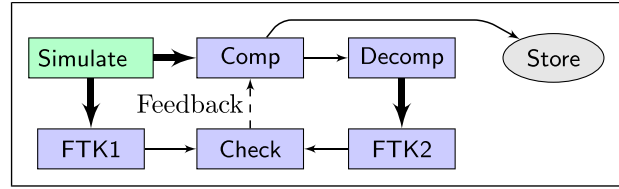


Figure 8. Components coupled to evaluate compressors for Gray-Scott reaction-diffusion application.

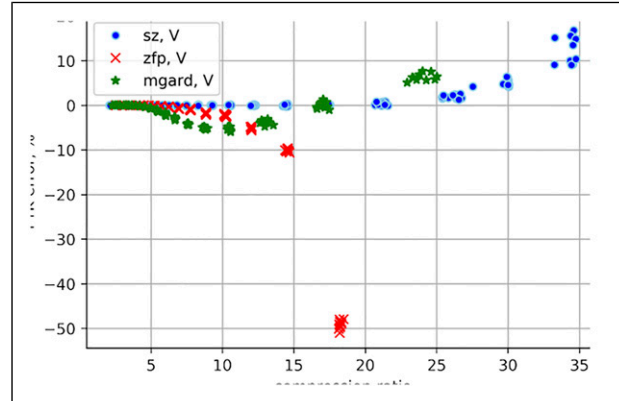


Figure 9. FTK error versus compression ratio, Gray-Scott, experiment 1, V, step = 10,000. From Yakushin et al. (2020).

coupling an analysis with the XGC code. We also discuss data reduction issues.

7.2.1. Coupling of simulation codes. In this first study, we explored the optimal allocation of resources for each component in a coupled simulation, placement of components on resources, and intercomponent data exchange mechanisms. We use a miniapp composed of two instances of the XGC code, one configured to simulate the edge of the fusion reactor and the other to simulate the core. (The primary differences between this miniapp and WDMApp are that WDMApp can use other core codes and implement the coupler as a separate process.) We use ADIOS for communication and can switch coupling approaches to exchange data via the GPFS storage system, node-local NVME when simulations run on the same node, or ADIOS's SST, which uses RDMA for communication.

We have used the miniapp to investigate a wide variety of resource allocation options. In one study, we investigated the impact of task placement on performance. When using fluid-based coupling (Dominski et al., 2018), 3D charge density and potential fluctuations data are exchanged in the overlap region of the codes, once per Runge-Kutta step (four times per simulation timestep in our implementation). We determine that the coupled code can be over 20% faster when

task-based graph embedding is employed to optimize which MPI ranks are used by each code (Choi et al. 2017, 2018, 2018).

We also investigated optimal resource allocations for each component for different science problems. In Figure 10, we demonstrate an edge-core coupling case on 32 Summit nodes in which 14 GB are exchanged at each iteration. The average iteration time varies depending on resource allocation (separate or shared) and ADIOS coupling methods (GPFS or SST). More details are in Choi et al. (2019).

7.2.2. Analysis and visualization. Whole-device model scientists commonly want to couple analysis codes to a simulation to obtain physics results in near-real time—for example, to detect instabilities that should cause a run to be halted. We consider here the case of tracer particle analysis, which enables the detailed understanding of energy flux focused on the edge, including through the X-point region. Scaling studies showed that this analysis could account for more than 30% of total runtime on 64 Summit nodes, which motivated the co-design question of whether alternative task

placements could reduce that cost—or, alternatively, whether new analysis methods were needed.

To investigate this question, we used CODAR tools to adapt the application so that the XGC and tracer particle analysis components could run on the same or different nodes, and investigated the performance achieved with different configurations. We show some performance results in Figure 11. We determined that when running XGC on 64 nodes, the best configuration for the entire application placed the tracer analysis on an additional 4 dedicated nodes. Total analysis time changed little on 4 vs 64 nodes, due to high interprocess communication costs within the analysis routine, and thus, as the analysis can run concurrently with XGC, the net effect of this reconfiguration is to reduce the analysis contribution to total cost from 31% to 6%. Furthermore, the benefits of this approach increase as XGC is scaled to yet more nodes.

7.2.3. Data reduction. XGC can produce 100 PB in a weeklong simulation on pre-exascale computers, with output taking about 4% of total runtime. Both numbers will

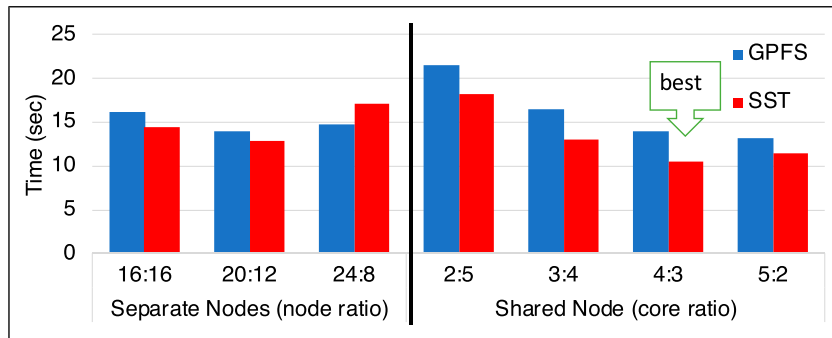


Figure 10. Whole-device model miniapp performance, demonstrating XGC edge-core coupling, on 32 Summit nodes with different node layouts (shared nodes and separate nodes) and ADIOS methods (GPFS and SST). The x-axis shows different node or core ratios assigned to the edge and the core application.

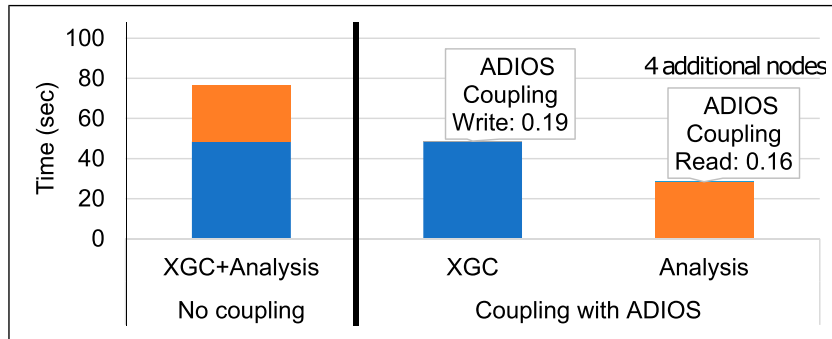


Figure 11. XGC code contains an analysis routine that advects tracer particles in the plasma to understand the flow around the different sections of the fusion reactor. This co-design study on Summit showed that moving the analysis from the 64 XGC nodes to four additional nodes reduced overall cost by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

increase considerably on exascale systems. Compression can reduce I/O volumes but introduces compression costs and has accuracy implications for common analysis routines run by WDMApp scientists. As a first step in a comprehensive co-design study, we have used MGARD to reduce the five dimensional f data to achieve a pointwise L_∞ error of 10^{-5} with a compression ratio of 13.

7.3. Online data analysis: NWChem

One focus of the NWChemEx ECP project is enabling MD simulations of million-atom dynamic biomolecular processes. A single simulation over microsecond timescales can involve 1B or more timesteps. Outputting all trajectory data for 1M atoms over 1B timesteps would generate 32 PB of output. Therefore, MD implementations have traditionally reduced at a reduced frequency, such as at every 100th timestep. However, that frequency would still generate 320 TB of output and may miss key phenomena. Thus, online data analysis and more sophisticated reduction methods are of interest.

We describe here three ODAR co-design studies in NWChemEx. The first is concerned with accelerating an online analysis that includes sorting and principal component analysis (PCA). The second uses an online adaptive sampling method to identify moments in an MD trajectory where the structure of the molecule undergoes significant changes. The third demonstrates a reduction in performance trace data achieved by identifying and storing only anomalous events. Since NWChemEx development was in progress, all studies used NWChem (Valiev et al., 2010) as a proxy.

7.3.1. Online data analysis. Sorting and PCA. Our first study investigated methods for outputting and analyzing MD simulation state. NWChem can be asked to write periodically to disk, in a table format with one row per atom, the spatial coordinates for every atom in a molecule. In order to track individual atoms over time, the table must be sorted by atom ID, which we can think of as a simple analysis. NWChem accomplished this task by performing an all-to-one MPI collective to move all atom data to rank zero, where a simple linear-time sort was applied and the sorted data were stored. A scientist later ingested the sorted data and performed analyses, such as PCA. We describe here how performance was improved by performing both the sorting and PCA online and concurrently.

A first co-design question is how best to organize the sorting and output step. We have two components (simulation and sort/output) coupled via the atomic coordinates produced by the simulation and consumed by sort/output. Since the sort/output can run concurrently while NWChem is computing, we identify as free parameters the choice of whether to run the simulation and sort/output on the same or different nodes and how many nodes (or cores) to allocate to

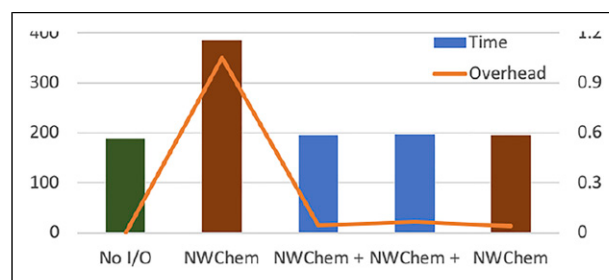


Figure 12. NWChem times for 1000 steps on a 36,536-atom problem. From left: original NWChem on 224 cores with no I/O, no sorting; original NWChem with sort/output every step; NWChem on 224 cores with two additional cores dedicated for sort/output; NWChem on 224 cores with two additional cores dedicated for sort/output and one additional core for running principal component analysis every 10 steps; original NWChem with sort/output every 25 steps.

each. Figure 12 shows relevant results. The first column gives the time for a 1000-step 36,536-atom simulation without sort and output: 188 s on 224 cores (14 nodes) of Rhea, a cluster at the Oak Ridge Leadership Computing Facility with 16-core, InfiniBand-connected x86 nodes. Total output is 2.3 MB of atom state per step (2.3 GB over 1000 steps). The second column shows that total time increases to 386 s when sorting and output are enabled on the same 224 cores.

Using Cheetah to perform runs in which we varied the number of cores dedicated to sorting, 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 total runtime to 194.7 s, 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 while only increasing total computational costs from $188 \times 224 = 42,112$ core-seconds with no output to $194.7 \times 226 = 44,002$ core-seconds.

Next we incorporated PCA, an analysis that is commonly applied to atom trajectory data to discover interesting movements of a molecule. To this end, we developed a motion-adjusted PCA analysis routine within the pbdR package (Schmidt et al., 2017) to run PCA repeatedly on a fixed-size window of 10 steps. As shown in the fourth column of Figure 12, PCA can run on a single additional core without slowing the overall computation significantly, for a total computational cost of $197.6 \times 227 = 44,851$ core-seconds.

Thus, we determined that by efficient configuration of online analysis, we can run NWChem with both sort/output and PCA with just a 6% overhead relative to running NWChem without these components.

7.3.2. 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. 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 measure such changes stably despite 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 only when the low-dimensional embeddings show significant changes. As shown by Yoo et al. (2016), a conventional nonstreaming 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, that is, it is linear in the number of atoms.

Figure 13 shows how this method can greatly reduce output while retaining key molecular conformations. 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 . Even fewer “interesting” conformations may be identified in longer simulations. For example, in a 10,000-step simulation, the adaptive online analysis identified 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.

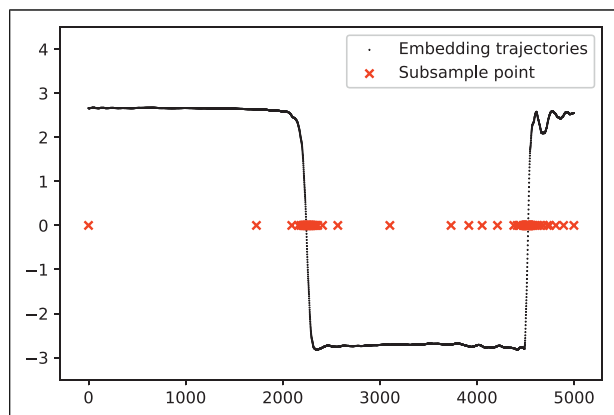


Figure 13. One-dimensional embedding values via PCA for a 5000-timestep trajectory where a protein first collides (first change) and then binds (second change) with DNA. Also shown are the 32 configurations selected for output by online sampling; for these, the y values have no significance.

7.3.3. Performance data reduction. We used the Chimbuko online performance analysis tool 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. A 1.2 M atom simulation on 2560 MPI ranks (128 nodes) of Summit generated 117.5 GB of performance trace events during a ~ 140 s 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.

7.4. 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., 2018b). CANDLE applications often involve many quasi-independent learning and inferencing tasks, each 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 hyperparameter 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 various search methods. Each NN configuration is characterized by quality measures such as validation error on a test set, and the network weights produced during training.

A large parallel computer can easily generate and evaluate NN configurations at tens of gigabytes per second on today’s computers and terabytes per second on exascale systems. Application scientists would strongly prefer to avoid writing these data to disk only to have to read them later to identify interesting configurations. But while reduction is desirable, it is 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 nontrivial. 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

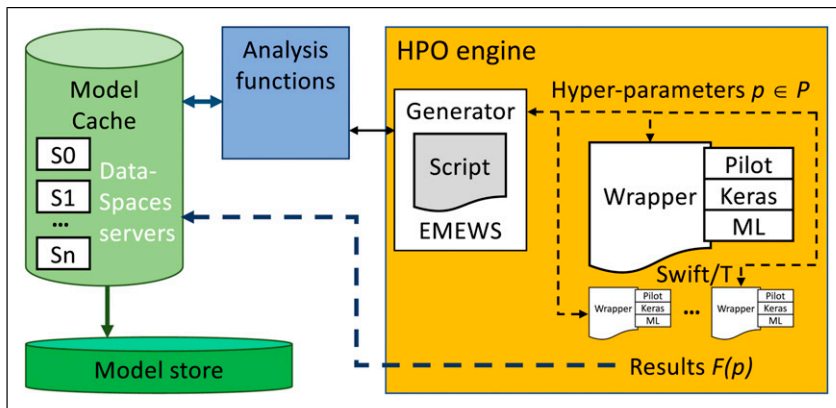


Figure 14. CANDLE hyperparameter optimization architecture with Model Cache, into which workers place results and with which analysis functions interact to filter results. Swift/T (Wozniak et al., 2013) is used to coordinate task execution and information flow; EMEWS (Ozik et al., 2016) is used to generate tasks.

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 architecture can be adapted to meet application needs. For example, Figure 15 shows results from an experiment with a synthetic workload 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

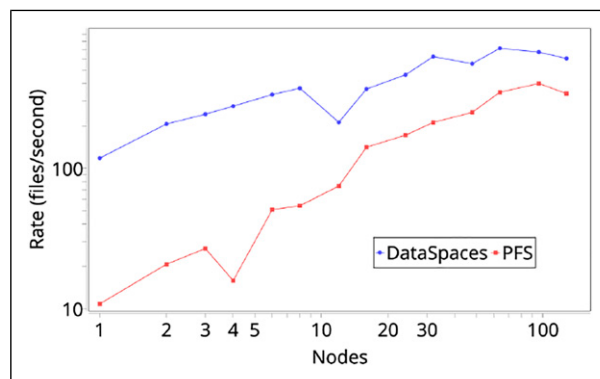


Figure 15. Small file (1 MB) access rate for CANDLE Model Store on NERSC Cori.

Cache and PFS implementations seems to converge, suggesting a need for multiple Model Cache servers. Co-design studies are needed 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.

8. Lessons learned

Our ECP co-design experiences motivate the following 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, since 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 such as 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 multicomponent ODAR system because of the need to account for interactions and interference among components. Exhaustive search of all possible configurations is rarely feasible. 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.

9. Conclusions

The ODAR motif is becoming increasingly important in HPC applications because of changes in both systems and applications. Efficient implementation of applications that implement this motif can raise challenging co-design questions due to trade-offs 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 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 of application components to processes, different communication structures, and different reduction methods) and to evaluate the data quality and computational performance achieved by these alternatives.

The co-design studies considered in this study have focused primarily on how to configure existing applications for efficient online data analysis and reduction, with few changes to the applications themselves. As components are coupled ever more tightly to enable rapid information exchange and feedback, it will likely become desirable also to reconfigure aspects of application internals, such as data layouts. This may motivate more extensive use of data layout abstractions.

The case studies presented here illustrate some of the ways in which ODAR methods and tools are being applied to accelerate ECP applications. Online data analysis and reduction tools can also be used in other ways. For example, the fusion whole-device modeling project's EFFIS coupling framework (Suchyta et al., 2021) uses Savanna to place components on nodes and processors; Z-Checker is used to evaluate compression methods in many contexts; Chimbuko is used to study the performance of ECP applications; and scientists are using ODAR methods to link ML with HPC to accelerate MD simulations (Brace et al., 2021). The co-design of ODAR workflows remains a fruitful activity that will benefit applications far into the future.

Acknowledgments

We thank ECP leadership and the many ECP colleagues with whom we have interacted in performing this work, and the referees and associate editor for their thoughtful comments that helped improve this article.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: 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.

ORCID iD

Ian Foster  <https://orcid.org/0000-0003-2129-5269>

References

- Abramson D, Sosic R, Giddy J, et al. (1995) Nimrod: a tool for performing parametrised simulations using distributed workstations. In: 4th IEEE international symposium on high performance distributed computing, Washington, DC, USA, 2–4 August 1995, pp. 112–121. IEEE.
- Ainsworth M, Tugluk O, Whitney B, et al. (2020) Multilevel techniques for compression and reduction of scientific data—The unstructured case. *SIAM Journal on Scientific Computing* 42(2): A1402–A1427.
- Ainsworth M, Tugluk O, Whitney B, et al. (2019) Multilevel techniques for compression and reduction of scientific data—The multivariate case. *SIAM Journal on Scientific Computing* 41(2): A1278–A1303.
- Alexander F, Almgren A, Bell J, et al. (2020a) Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378(2166): 20190056.
- Alexander FJ, Ang J, Bilbrey JA, et al. (2020b) Co-design center for exascale machine learning technologies (ExaLearn). *International Journal of High-Performance Computing Applications* in press.
- Asanovic K, Bodik R, Catanzaro BC, et al. (2006) *The Landscape of parallel Computing Research: A View from Berkeley*. Berkeley: University of California, Technical Report UCB/EECS-2006-183.
- Ayachit U, Bauer A, Duque EP, et al. (2016) Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In: SC'16: proceedings of the international conference for high performance computing, networking, storage and analysis, Salt Lake City, UT, 13–18 November 2016, pp. 921–932. IEEE.
- Aymar R, Barabaschi P and Shimomura Y (2002) The ITER design. *Plasma Physics and Controlled Fusion* 44(5): 519–565. DOI: [10.1088/0741-3335/44/5/304](https://doi.org/10.1088/0741-3335/44/5/304).
- Balaprakash P, Dongarra J, Gamblin T, et al. (2018) Autotuning in high-performance computing applications. *Proceedings of the IEEE* 106(11): 2068–2083.
- Barrett RF, Borkar S, Dosanjh SS, et al. (2013) On the role of co-design in high performance computing. In: (eds) *Transition of HPC towards Exascale Computing*. IOS Press, 141.
- Bauer AC, Abbasi H, Ahrens J, et al. (2016) In situ methods, infrastructures, and applications on high performance computing platforms. *Computer Graphics Forum* 35: 577–597.
- Beazley DM and Lomdahl PS (1996) Lightweight computational steering of very large scale molecular dynamics simulations. In: SC'96 computer graphics forum, Pittsburgh, PA, 1 January 1996, p. 50. IEEE.
- Brace A, Lee H, Ma H, et al. (2021) Achieving 100X faster simulations of complex biological phenomena by coupling ML to HPC ensembles. arXiv preprint arXiv:2104.04797.
- Carriero N and Gelernter D (1989) Linda in context. *Communications of the ACM* 32(4): 444–458.
- Childs H, Ahern SD, Ahrens J, et al. (2020) A terminology for in situ visualization and analysis systems. *International Journal of High Performance Computing Applications* 34.
- Choi JY, Chang CS, Dominski J, et al. (2018) Coupling exascale multiphysics applications: Methods and lessons learned. In: 14th international conference on e-Science, Amsterdam, The Netherlands, 29 October–1 November 2018, pp. 442–452. IEEE.
- Choi JY, Logan J, Mehta K, et al. (2019) A co-design study of fusion whole device modeling using code coupling. In: IEEE/ACM 5th international workshop on data analysis and reduction for big scientific data, Denver, CO, 17 November 2019, pp. 35–41. IEEE.
- Choi JY, Logan J, Wolf M, et al. (2017) TGE: machine learning based task graph embedding for large-scale topology mapping. In: International conference on cluster computing, Honolulu, HI, 5–8 September 2017, pp. 587–591. IEEE.
- Curtis B, Krasner H and Iscoe N (1988) A field study of the software design process for large systems. *Communications of the ACM* 31(11): 1268–1287.
- De Michell G and Gupta RK (1997) Hardware/software co-design. *Proceedings of the IEEE* 85(3): 349–365.
- Di S and Cappello F (2016) Fast error-bounded lossy HPC data compression with SZ. In: IEEE international parallel and distributed processing symposium, Chicago, IL, 23–27 May 2016, pp. 730–739. IEEE.
- Diffenderfer J, Fox AL, Hittinger JA, et al. (2019) Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing* 41(3): A1867–A1898.
- Docan C, Parashar M and Klasky S (2012) DataSpaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing* 15(2): 163–181.
- Dominski J, Ku S, Chang C-S, et al. (2018) A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes. *Physics of Plasmas* 25(7): 072308. DOI: [10.1063/1.5044707](https://doi.org/10.1063/1.5044707).
- Duplyakin D, Brown J and Ricci R (2016) Active learning in performance analysis. In: IEEE conference on cluster computing (CLUSTER), Taipei, Taiwan, 12–16 September 2016, pp. 182–191. IEEE.

- Efraimidis PS and Spirakis PG (2006) Weighted random sampling with a reservoir. *Information Processing Letters* 97(5): 181–185.
- Foster I (1994) *Designing and Building Parallel Programs*. Addison-Wesley.
- Foster I, Ainsworth M, Allen B, et al. (2017) Computing just what you need: online data analysis and reduction at extreme scales. In: European conference on parallel processing, pp. 3–19.
- Foster I, Insley J, von Laszewski G, et al. (1999) Distance visualization: data exploration on the grid. *Computer* 32(12): 36–43.
- Foster I and Kesselman C (2006) Scaling system-level science: scientific exploration and IT implications. *Computer* 39(11): 31–39.
- Godoy WF, Podhorszki N, Wang R, et al. (2020) ADIOS 2: the adaptable input output system. A framework for high-performance data management. *SoftwareX* 12: 100561.
- Gray J and Shenoy P (2000) Rules of thumb in data engineering. In: 16th international conference on data engineering, San Diego, CA, 29 February–3 March 2000, pp. 3–10. IEEE.
- Guo H, Lenz D, Xu J, et al. (2021) FTK: a simplicial spacetime meshing framework for robust and scalable feature tracking. *IEEE Transactions on Visualization & Computer Graphics*.
- Guo H, Xu J, Liang X, et al. (2020) FTK: the feature tracking kit. Available at: <http://www.github.com/CODARcode/ftk> (accessed 1 September 2020).
- Ha S, Jeong W, Matyasfalvi G, et al. (2020) Chimbuko: a workflow-level scalable performance trace analysis tool. arXiv preprint arXiv:2008.13742
- He W, Wang J, Guo H, et al. (2019) InSituNet: deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization & Computer Graphics* 26(1): 23–33.
- Hoefler T, Gropp W, Kramer W, et al. (2011) Performance modeling for systematic performance tuning. In: SC'11: proceedings of 2011 international conference for high performance computing, networking, storage and analysis, Seattle, WA, 12–18 November 2011, pp. 1–12. IEEE.
- Insley JA, Papka ME, Dong S, et al. (2007) Runtime visualization of the human arterial tree. *IEEE Transactions on Visualization and Computer Graphics* 13(4): 810–821.
- Johnson C, Parker SG, Hansen C, et al. (1999) Interactive simulation and visualization. *Computer* 32(12): 59–65.
- Klasky S, Abbasi H, Logan J, et al. (2011) In situ data processing for extreme-scale computing. *Proceedings of SciDAC*: 1–16.
- Larsen M, Ahrens J, Ayachit U, et al. (2017) The alpine in situ infrastructure: ascending from the ashes of strawman. In: Workshop on in situ infrastructures on enabling extreme-scale analysis and visualization, pp. 42–46.
- Lee H, Turilli M, Jha S, et al. (2019) DeepDriveMD: deep-learning driven adaptive molecular simulations for protein folding. In: 3rd workshop on deep learning on supercomputers, Denver, CO, 17 November 2019, pp. 12–19. IEEE.
- Liang X, Di S, Tao D, et al. (2018) Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In: International conference on big data, Seattle, WA, 10–13 December 2018, pp. 438–447. IEEE.
- Ma KL, Wang C, Yu H, et al. (2007) In-situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series* 78: 012043.
- Malakar P, Munson T, Knight C, et al. (2018) Topology-aware space-shared co-analysis of large-scale molecular dynamics simulations. In: SC'18: international conference for high performance computing, networking, storage and analysis, Dallas, TX, 11–16 November 2018, pp. 305–319. IEEE.
- Malakar P, Vishwanath V, Knight C, et al. (2016) Optimal execution of co-analysis for large-scale molecular dynamics simulations. In: SC'16: proceedings of the international conference for high performance computing, networking, storage and analysis, Salt Lake City, UT, 13–18 November 2016, pp. 702–715. IEEE.
- Malakar P, Vishwanath V, Munson T, et al. (2015) Optimal scheduling of in-situ analysis for large-scale scientific simulations. In: SC'15: proceedings of the international conference for high performance computing, networking, storage and analysis, pp. 1–11. IEEE.
- Mehta K, Allen B, Wolf M, et al. (2019) A codesign framework for online data analysis and reduction. In: IEEE/ACM workflows in support of large-scale science, Denver, CO, 17 November 2019, pp. 11–20. IEEE.
- Ousterhout J (2018) *A Philosophy of Software Design*. Yaknyam Press.
- Ozik J, Collier N, Wozniak JM, et al. (2016) From desktop to large-scale model exploration with Swift/T. In: Winter simulation conference, Washington, DC, USA, 11–14 December 2016, pp. 206–220. IEEE.
- Parnas DL (1972) On the criteria to be used in decomposing systems into modules. In: (eds) *Pioneers and Their Contributions to Software Engineering*. Springer, 479–498.
- Pouchard L, Huck K, Matyasfalvi G, et al. (2018) Prescriptive provenance for streaming analysis of workflows at scale. In: New York scientific data summit, NY, USA, 6–8 August 2018, pp. 1–6. IEEE.
- Ratanaworabhan P, Ke J and Burtscher M (2006) Fast lossless compression of scientific floating-point data. In: Data compression conference, Snowbird, UT, 28–30 March 2006, pp. 133–142. IEEE.
- Schmidt D, Chen W-C, Matheson MA, et al. (2017) Programming with big data in R: scaling analytics from one to thousands of nodes. *Big Data Research* 8: 1–11.
- Shende SS and Malony AD (2006) The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 20(2): 287–311.
- Shu T, Guo Y, Wozniak J, et al. (2020) In-situ workflow auto-tuning via combining performance models of component applications. arXiv preprint arXiv:2008.06991
- Sims K (2020) *Reaction-Diffusion Tutorial*. Available at: www.karlsims.com/rd.html.

- Subhlok J, Stichnoth JM, O'Hallaron DR, et al. (1993) Exploiting task and data parallelism on a multicomputer. In: ACM SIGPLAN notices 4th symposium on principles and practice of parallel programming, 28, pp. 13–22. ACM.
- Suchyta E, Klasky S, Podhoseski N, et al. (2021) The exascale framework for high fidelity coupled simulations (EFFIS): enabling whole device modeling in fusion science. *International Journal of High Performance Computing Applications*.
- Tao D, Di S, Chen Z, et al. (2017) Significantly improving lossy compression for scientific data sets based on multi-dimensional prediction and error-controlled quantization. In: IEEE international parallel and distributed processing symposium, Orlando, FL, 29 May–2 June 2017, pp. 1129–1139. IEEE.
- Tao D, Di S, Guo H, et al. (2019) Z-Checker: a framework for assessing lossy compression of scientific data. *The International Journal of High Performance Computing Applications* 33(2): 285–303.
- Underwood R, Di S and Cappello F (2020) CODARcode/libpressio. Available at: <http://www.github.com/CODARcode/libpressio>.
- Valiev M, Bylaska EJ, Govind N, et al. (2010) NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 181(9): 1477–1489.
- Wozniak JM, Armstrong TG, Wilde M, et al. (2013) Swift/T: scalable data flow programming for distributed-memory task-parallel applications. In: 13th international conference on cluster, cloud and grid computing, Delft, The Netherlands, 13–16 May 2013, pp. 95–102. IEEE.
- Wozniak JM, Davis PE, Shu T, et al. (2018a) Scaling deep learning for cancer with advanced workflow storage integration. In: IEEE/ACM machine learning in HPC environments, Dallas, TX, 12 November 2018, pp. 114–123. IEEE.
- Wozniak JM, Dorier M, Ross R, et al. (2019) MPI jobs within MPI jobs: a practical way of enabling task-level fault-tolerance in HPC workflows. *Future Generation Computer Systems* 101: 576–589.
- Wozniak JM, Jain R, Balaprakash P, et al. (2018b) CANDLE/supervisor: a workflow framework for machine learning applied to cancer research. *BMC Bioinformatics* 19(18): 491.
- Xie C, Xu W and Mueller K (2019) A visual analytics framework for the detection of anomalous call stack trees in high performance computing applications. *IEEE Transactions on Visualization and Computer Graphics* 25(1): 215–224.
- Yakushin I, Mehta K, Chen J, et al. (2020) Feature-preserving lossy compression for in situ data analysis. In: 49th international conference on parallel processing: workshops, pp. 1–9. ACM.
- Yoo S, Huang H and Kasiviswanathan SP (2016) Streaming spectral clustering. In: 32nd international conference on data engineering, Helsinki, Finland, 16–20 March 2016, pp. 637–648. IEEE.
- Zhang X, Huang H, Mueller K, et al. (2018) Streaming classical multidimensional scaling. In: New York scientific data summit, NY, USA, 6–8 August 2018, pp. 1–2. IEEE.
- Zhao K, Di S, Liang X, et al. (2020) Significantly improving lossy compression for HPC datasets with second-order prediction and parameter optimization. In: 29th international symposium on high-performance parallel and distributed computing, pp. 89–100. IEEE.

Author biographies

Ian Foster is a 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 computer scientist in the 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 in the Mathematics and Computer Science Division at Argonne National Laboratory.

Hanqi Guo is an assistant computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory.

Kevin A. Huck is a faculty research associate at the University of Oregon.

Chris Kelly is a scientist in the Computational Science Initiative at Brookhaven National Laboratory.

Scott Klasky is a distinguished scientist and group leader for Workflow Systems 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 an assistant professor in computer science at Missouri University of Science and Technology.

Kshitij Mehta is a computer scientist 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 is an assistant professor at Southern Illinois University Carbondale.

Ozan Tugluk is a postdoctoral research associate at the Division of Applied Mathematics, Brown University.

Hubertus Van Dam is an application architect in the Computational Science Initiative at Brookhaven National Laboratory.

Lipeng Wang is a computer scientist in the Computer Science and Mathematics Division at Oak Ridge 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.

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.