Contents

Project Narrative

1 Introduction

Motivation. Exponential increases in data size and the need to distill enormous amounts of information into usable knowledge are pushing the limits of data processing in many disciplines. A popular model to address the data avalanche is to process (transform, analyze, visualize) data closer to the source by linking processing with computation in a stream, pipeline, or network. Today such coupling or linking is performed either through storage or through memory/interconnect. In this proposal we refer to the former as *loose* coupling and the latter as *tight* coupling. Both modes have pros and cons. Loose coupling has favorable data translation and fault tolerance properties: the file is a common data representation language, and persistent storage provides data replication and a firewall against cascading errors. Despite these properties, the long latency, I/O bandwidth pressure, and the energy cost of moving data into and out of storage make loose coupling inappropriate for extreme-scale scientific data analysis, encouraging the community to turn to tight coupling instead. Tight coupling, while avoiding the I/O bottleneck and data movement energy costs, juxtaposes dissimilar components with different programming, data, performance, and fault models. A fragile system that practicing scientists often avoid, tight coupling is limited in practice to a simple direct stream custom-tailored to the endpoints. In most cases this means very specific data models for the specific producer/consumer pair with rigid communication lacking flexible flow control to buffer data rate mismatches, no consistent fault tolerance strategy, and no reuse for other combinations of endpoints.

Objective. The crux of the problem is the dichotomy between loose and tight coupling when actually neither is optimal for extreme-scale science. We propose loosening the grip of tight coupling, in essence decoupling tightly coupled data flows (called *Decaf* in this proposal) while keeping their favorable high performance and low power characteristics. For example, Figure 1 shows a workflow consisting of tight parallel tasks. We will expand each of those links with a miniature dataflow graph made of reusable primitives. The presence of optional short- and long-term storage in the dataflow gives us the best of both worlds: tight coupling whenever possible but loose coupling when usage patterns require persistent data. Our dataflow nodes may be heterogeneous and reside anywhere in the system; we will optimize their number, location, and organization to satisfy objectives such as data movement or time to solution. We will augment the dataflow with essential data operators—pipelining, selection, and aggregation—in order to achieve space-time data permutations within the dataflow, and we will execute the dataflow over various transport layers in current and future HPC architectures.

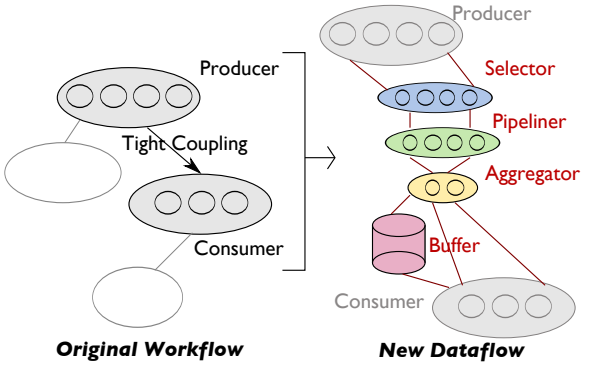


Figure 1: From a workflow to a dataflow, we will automatically expand links with a small set of primitives composed in an optimal way to satisfy user requirements.

Novel contributions and deliverables. Our proposed research addresses themes 2, 3, and 5 of the FOA: we target in situ methods and workflows, and we will evaluate our research in full and proxy applications developed in this project and by other groups in order to *improve performance, reduce power, add fault tolerance, and enhance usability*. Our proposal will result in (1) a library of dataflow primitives, akin to "BLAS for dataflow," (2) a method for automatically constructing broadly applicable dataflows from the same set of primitives, and (3) a generic and reusable solution that other workflow and coupling tools can use. We will not sacrifice performance or increase power: on the contrary, intelligently loosening rigid flows with a small set of data operators will enable connectivity of a simulation to analysis tools that would

otherwise default to expensive postprocessing. *Automatic pipelining* will support shallow copy adapters that transform data models as needed, a chunk of data at a time, rather than in its entirety, relieving pressure at all levels of memory. *Automatic aggregation* will reduce interconnect congestion. The ability to *select and permute* time-varying data in situ will reduce the need to read entire datasets back into memory for this single operation. Similarly, the ability to *handle faults* during coupling using optimal buffering of partial results, stateful operators, and criticality of data can avoid expensive entire restarts in case of errors. *Automatic buffering* allows bursty data production and consumption rates to be managed and faults to be contained. Because storage is just another producer or consumer in our design, we can augment it with the same data primitives and operate on data where they reside, further reducing postprocessing data round trips.

Research areas and project organization. We will integrate three levels of abstraction and software shown in Figure 2. A *high-level data description (led by Peterka)* (Section 3.1) will capture both the intrinsic data properties (sizes, types) and the extrinsic properties that depend on producer or consumer endpoints (data rates, chunk sizes). The *transport layer (led by Kimpe)* (Section 3.2) will move data, providing pipelining, and control flow with short- and long-term buffering using four coupling primitives (selector, pipeliner, aggregator, buffer). The data descriptions and coupling primitives will guide the synthesis of a *dataflow model (led by Lofstead)* (Section 3.3) that optimizes the performance of each link between producer and consumer. The data description layer will also define criticality of data that will be used to devise low-overhead *resilience strategies (led by Cappello)* (Section 3.4) combining appropriate detection and recovery techniques for fault tolerance at the two lower layers. The computing system is described by the other cross-cutting piece of our project, where we will tie into existing OS/R research to leverage capabilities such as global control backplanes and couple applications across enclaves or partitions (Section 4.2).

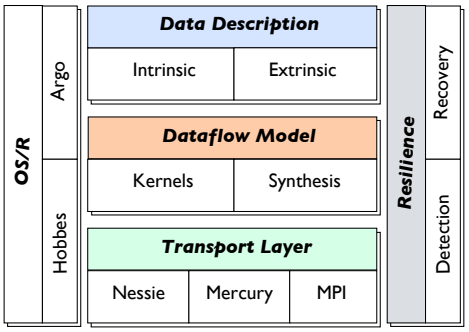


Figure 2: Overall project organization.

Use cases and impact on DOE mission. Three common use cases—data streams, data pipelines, and data networks—will drive our research. We will decouple data streaming in the in situ analysis of DOE simulations to trace particles in nuclear reactor Navier-Stokes computations. We will study data pipelining with several independent steps of in situ analysis to identify features in Ginzburg-Landau superconductivity simulations and then to track those features as they evolve over time. We will decouple dataflow networks in a workflow that transforms particles, meshes, and grids in Vlasov-Poisson N-body cosmology codes. Our proposal is designed to integrate with and contribute to four other efforts in the ASCR portfolio: Argo, Hobbes, SDAV, and CESAR. Argo and Hobbes are OS/R efforts in extreme-scale runtimes and operating systems. While Argo and Hobbes partition systems by enclaves and support information flow within and across enclaves, they do not provide any API or runtime support specifically for data processing and data coupling: our proposal addresses this gap. CESAR, one of three co-design centers, targets multiphysics coupling among other research foci. The coupling proxy app being developed in CESAR addresses accuracy and performance of simulation-simulation coupling but does not address simulation-analysis coupling, nor does it optimize coupling of complex workflows. Again, we target this gap. SDAV is the SciDAC-3 data management, analysis, and visualization effort to place SDMAV tools in the hands of computational scientists. Coupling of simulation and analysis in SDAV exists as custom and predefined point solutions for specific science use cases, but no general-purpose dataflow library is covered in SDAV or elsewhere. The solution that we propose will be able to be deployed in out-years through production-oriented funding mechanisms such as SciDAC.

2 Background: Applications, Architectures, and Existing Tools

2.1 Use Cases and Representative Science Applications

Three use cases—data stream, data pipeline, and data network—are represented by three science applications with which our team works on a regular basis: Navier-Stokes CFD [16, 25, 45] Ginzburg-Landau superconductivity [53, 27], and Vlasov-Poisson N-body cosmology[32, 6]. As a motivating example, we explore how the most complex case, a data network, is categorized for cosmology data analysis whereby particles are converted to a continuous mesh tessellation that is used to estimate particle density distribution onto a regular grid. The other use cases are derived similarly.

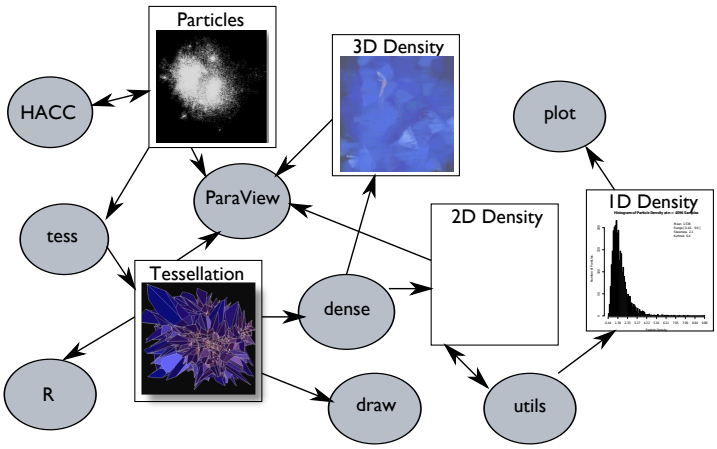


Figure 3: Dataflow network for data analysis of particle density from an N-body cosmological simulation.

Figure 3 is a **dataflow network** for a subset of the science workflow for the computation and analysis of dark matter tracer particles in the HACC [31, 30] N-body cosmology code running on the largest supercomputers in the world. The example focuses on the estimation of particle density; circles in the dataflow network indicate producers or consumers of data, and rectangles indicate the data products (data models). For example, HACC produces raw particles that can be viewed directly with ParaView [63] or particles can be converted to a mesh tessellation through the tess [49] parallel library, which can be visualized or further resampled onto a regular grid with the parallel tool dense.[1]Other serial utilities can operate offline on the tessellation and density fields.

Table 1: Dataflow Matrix Derived from Dataflow Network. Legend: P = particles, T = tessellation, ρ^3 = 3D density field, ρ^2 = 2D density field, ρ^1 = 1D density

Producers	**Consumers**							
	HACC	R	Tess	ParaView	Dense	Draw	Utils	Plot
HACC	P		P	P				
Tess		T		T	T	T		
Dense				ρ^3, ρ^2			ρ^2	
Utils							ρ^2	ρ^1

To resolve ambiguities such as cycles and invalid paths in the network, we convert the graph to a **dataflow matrix** shown in Table 1. Producers are listed in the left column; consumers are listed in the top row; and zero, one, or more data models appear in the body of the matrix for each producer-consumer pair. From the dataflow matrix, we derive two classes of **Data Properties**: those that are independent of producer and consumer (intrinsic) and those that depend on the particular endpoint (extrinsic). The intrinsic properties for the tessellation data model and the extrinsic properties for the tess and ParaView applications are listed in Table 2. Other data properties for the above example are defined similarly.

[1]Presented in an invited talk at CoDA'14 March5 2014 in Santa Fe, NM by Tom Peterka

2.2 Architectures for Scientific Data Analysis

Our proposed solutions are heavily influenced by HPC leadership-class computational hardware and system software. Simply put, big data live on big machines; and proposed solutions *must* scale to the largest supercomputers today and to architectures projected for the next 3-5 years. Through our integration with leadership computing facilities, we constantly are developing parallel analysis applications and systems software for the newest machines; and through PI Peterka's participation in the DOE exascale co-design initiative, [2] we are influencing the design of these machines. In this proposal, we focus our efforts on petascale HPC machines while planning for next-generation architectural challenges (Table 3) [60] in order to develop scalable technologies with greater longevity. As the International Exascale Software Project Roadmap [21] predicts, future machines will have high concurrency, multiple levels of parallelism, heterogeneity, constrained power consumption, and limited data bandwidth. These findings are confirmed by other groups, including the ASCAC subcommittee on exascale computing [22].

Table 2: Data Properties

Type	Property	Value
Intrinsic	Name	T
	Size	850 B/particle
	Decomp	DIY block
Extrinsic	Endpoint	Tess
	Role	Producer
	Data Format	dblock_t
	Granularity	Entire
	Duration	End
	Distribution	Multiblock Round Robin
Extrinsic	Endpoint	ParaView
	Role	Consumer
	Data Format	VTK
	Granularity	Entire
	Duration	Start
	Distribution	1 Block per Process

Our team has access to supercomputing resources at the Argonne and Oak Ridge leadership computing facilities and at the National Energy Research Scientific Computing Center (ALCF, OLCF, NERSC). *Mira* is the 10-petaflop IBM Blue Gene/Q supercomputer at the ALCF, with 48K 16-core compute nodes and 768 terabytes of memory. It exemplifies homogeneous multicore supercomputing architectures. *Titan* is the 20-petaflop hybrid Cray XK6 next-generation supercomputer at the OLCF. Unlike Mira, Titan couples a multicore CPU with a GPGPU processor; and it exemplifies heterogeneous, multithreaded supercomputing architectures. *Edison* is a 2-petaflop Cray XC30 at NERSC that has larger memory and higher bandwidth (memory, network, and disk) for high-throughput data analytics. Additionally, our team has access to *Cielo*, a 1.3-petaflop Cray XE6 at LANL with 8,944 16-core nodes.

2.3 Existing Data Flow Tools

Instead of proposing a connected component technology such as CCA [4, 5], we focus our efforts on transformations for data movement, not on building a workflow by discovering and connecting ports nor on a specification for building components or their interfaces. The CCA model requires defining the interface to the components through the procedural caller-callee relationship between modules. Decaf, on the other hand, is defined in terms of data properties, not producer/consumer computation properties. The CCA runtime primarily features in-memory tight component coupling of a single application. [3] When CCA components are parallel, their connection is independent 1:1 or MxN for arrays. Decaf, in comparison, features numerous truly parallel modes for data coupling such as buffering, pipelining, MxN, aggregation, and predefined as well as custom data transformations.

[2] https://cesar.mcs.anl.gov/research/projects

[3] http://www.cs.uoregon.edu/research/paracomp/papers/cca_cpe04/html/paper.html

Table 3: Projected Exascale Computer Specifications

Parameter	Projected Value
Total cores	10^9
Peak compute rate	1 EFLOP/s
Total power	20 MW
Total memory	64 PB
Total storage	500 PB
Aggregate storage bandwidth	60 TB/s
Mean time to interrupt	1 day

The major challenge that online workflows face is how to interface the various components. Workflow systems today rely on custom-built data manipulation functions to adjust the data from one workflow component to the next. This proposal seeks to reduce the requirement for these one-off data translation functions through standard interfaces with general, but usable functionality. Rather than competing with workflow tools such as Swift [59, 64], Condor [7], Kepler [43], Galaxy, [4] Yawl [61], Pegasus [14], and DAGMan [44], we designed Decaf to be a small library of standardized reusable dataflow tools sitting below workflows in the software stack so that Decaf will assist workflows in their work.

Decaf is also not a data staging/in transit framework. We envision that frameworks such as ADIOS [42] will be able to use our library to assist in coupling activities flexibly, reliably, and over a uniform data interface, all lacking today. While ADIOS does have a standard API for I/O [40], everything beneath that is custom and requires manual data conversion for each scenario. The first such data staging/in transit method [48] offloaded FFT processing on seismic data. Since then, these techniques have generally moved behind more standard I/O interfaces, and ADIOS was designed to provide a simple API with pluggable transport methods capable of either writing directly to disk or integrating with an online workflow. Demonstrations using production applications with useful data processing operations have shown in transit compression, filtering, indexing, LOD encoding, layout optimization, and array permutations. The PreDatA [65] project demonstrated the importance of selecting proper placement for data-processing operators and the conditions under which advantages can be gained. Additional processing operators have also been demonstrated [28]. Supported networks include Gemini, Cray Portals, and InfiniBand; transport mechanisms are MPI, DataTap/EVPath [2], and Nessie [38].

All these solutions aim to eliminate persistent storage for intermediate data by tightly coupling components. This accelerates the connection between workflow components, but it requires special interfaces tailored to the endpoints and does not address resilience in a portable way. We believe that these data transport tools, like the workflow tools, can benefit from our general dataflow abstraction. XCHANGE was an early effort in mapping input to output data format through a fixed set of operators [39]. DataSpaces [33] works through a custom interface to push data or metadata into a shared storage area but does not provide methods for deep storage or for controlling task triggering or blocking for data availability [17]. DataSpaces integrates DART as an asynchronous data transport and DIMES for point-to-point data movement into a unified publish/subscribe messaging/events substrate. Flexpath [12] is a point-to-point messaging system; COD (C on Demand) [1] provides dynamic code deployment at the most approporirate location though just-in-time compilation of a nearly complete subset of C. Leveraging Sandia's NNTI library provides cache-efficient, on-node shared-memory transport. I/O Containers [13] demonstrated a management framework that can restart failed components or redeploy operations based on system conditions and redirect I/O to disk should a component permanently fail or resources are insufficient to handle the dataflow bandwidth. Our proposed resilience research in Section 3.4 goes beyond restarts to detecting and correcting hard and soft errors, including silent data corruption, at various levels of accuracy customized for data analysis.

[4] https://usegalaxy.org

3 Project Objectives

Our main technical objectives align with the three horizontal software layers and the right-hand vertical cross-cutting column in Figure 2. Section 3.1 describes the data description; Section 3.2 describes the transport layer; Section 3.3 describes the dataflow model, and Section 3.4 describes cross-cutting resilience aspects. The discussion of the other vertical cross cutting column is deferred until Section 4.2.

3.1 Objective 1: Data Description

In Section 2 we derived properties of data based on use cases found in data analysis of DOE science at extreme scale; our objective now is to formalize that data description. We adopt an hourglass model: only the minimal description of the data, producer, and consumer (the middle of the hourglass) that is needed to synthesize a dataflow is described by the user. We do not want to redefine entire data dwarfs or force the description of data details not needed for Decaf to execute because, in our experience, scientists will not rewrite the interfaces to their codes. At most, we will annotate existing data models and adapt one data model to another at run time. Based on the previous analysis, we derive the following common properties of the data models. In a prototype version, these properties will be entered by the user in an XML file. In a production version, data producers and consumers would write these properties automatically or semi-automatically into a common system location, for example as additional fields in an ADIOS [41] XML configuration file.

```
struct dblock_t *d = (struct dblock_t *)dblock;
struct map_block_t map[] = {

  { DIY_FLOAT, OFST, 3,
    offsetof(struct dblock_t, mins)             },
  { DIY_FLOAT, ADDR, d->num_particles * 3,
    DIY_Addr(d->particles)                      },
  { ttype,     ADDR, d->num_tets,
    DIY_Addr(d->tets)                           },
  { rtype,     ADDR, d->num_rem_tet_verts,
    DIY_Addr(d->rem_tet_verts)                  },
  { DIY_INT,   ADDR, d->num_particles,
    DIY_Addr(d->vert_to_tet)                    },
  { DIY_FLOAT, OFST, 3,
    offsetof(struct dblock_t, maxs)             },

};

DIY_Create_struct_datatype(DIY_Addr(dblock), 6, map, dtype);
```

Figure 4: DIY datatypes are a compact language for marshaling complex datatypes consisting of nested structures and dynamically allocated buffers.

Intrinsic properties are independent of producer and consumer. They consist of the tuple *(data model name, total size, complete data type, chunk data type, decomposition type)*. Some properties are self-explanatory; but complete data type, chunk data type, and decomposition type warrant further discussion. The *complete and chunk data types* define how to marshal data for moving the entire data model and chunks for pipelining and how to correctly reconstruct the data model at the consumer. For prototyping, we will use the DIY datatype [51] data model language as a compact method for describing data layout in memory of general data structures that can include vectors, subarrays, plain and nested structures, and pointers to heap-allocated data. An example is shown in Figure 4 for marshaling an unstructured tetrahedral mesh; we have defined completely custom data types for Morse-Smale Complexes [29], data distributions [11], Lagrangian particle traces [52, 46], and unstructured polyhedral meshes [49] this way, and the same technique allows us to marshal a subset of the entire data model for the chunk data type.

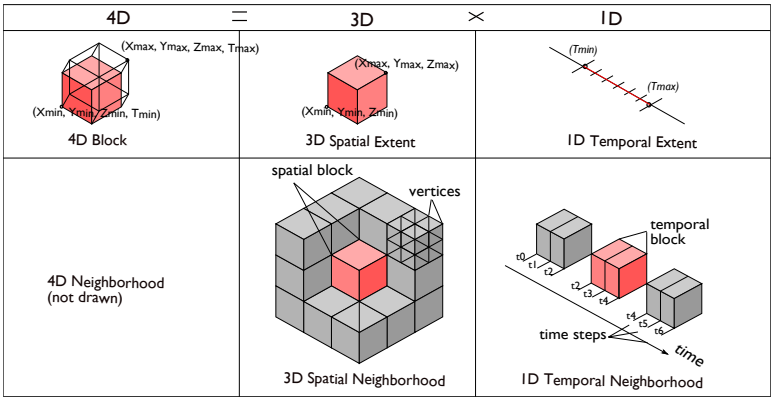


Figure 5: Hybrid 3D-4D block structure

The data decomposition consists of a set of subdomains called *blocks* and their assignment to tasks (threads,

processes). Blocks are simply the complete data types defined above; tasks may have multiple blocks that may be replicated. Blocks have 4D space-time extent and 4D neighborhoods. When convenient, 4D space-time is separated into 3D space and 1D time, as in Figure 5. The ordering of blocks can be regular (space-filling curves, row-major or column-major coordinate ordering, DFS and BFS tree or graph walks) or irregular, and their task mapping can likewise be regular (round-robin, contiguous) or irregular. The regular cases will be handled with predefined arguments (z-order + round-robin, for example), and the irregular cases will be explicitly defined with a lookup table of *(key = blocks, value = tasks)*. Because the table can be large, it will be maintained as a distributed array, where the key is a regular round-robin computation implicit to all tasks.

```
int DF_Add_data(char *Data_model_name);
int DF_Add_producer(int producer_id, float tot_period,
                    float burst_period, int tot_data_size,
                    int burst_data_size);
int DF_Add_consumer(int capacity, float input_bw);
int DF_Get(int producer_id, int consumer_id,
           void(*sel_type_func)(DF_Datatype *),
           void(*pipe_type_func)(DF_Datatype *),
           void(*aggr_type_func)(DF_Datatype *),
           int get_freq, int get_strict,
           int num_aggr, int resilience_level,
           void (*fault_check)(DF_Datatype *));
```

Figure 6: Data description API

Our earlier uses of DIY were limited to single, dedicated analysis algorithms. Using our existing tools for marshaling and decomposing data, we will investigate how to generalize those ideas for multiple nodes of a workflow when producer and consumer are not known in advance. One possible API for data properties is shown in Figure 6. The data model of previously-defined intrinsic data properties are added to Decaf with the `DF_Add_data` function; the remaining properties defined in Figure 6 are **extrinsic properties** that depend on a particular producer or consumer. They consist of the tuple *(endpoint name, endpoint role, data format, access granularity, access duration, distribution)*, and they are defined by the rates at which they produce or consume data. A periodic burst model is used in `DF_Add_producer`. Consumers are defined according to their ingest properties in `DF_Add_consumer`. The function `DF_Get` initiates a dataflow driven by the consumer using a pull model. In addition to specifying the producer and consumer, it defines a data ingest task. To do so, it provides usage information about how to select, pipeline, and aggregate data via custom functions and previously defined data types for those operations. The frequency of ingest (every frame, first, last) and the strictness to which that frequency must be adhered (strict / best effort) are also given. Optional hints such as the number of aggregators may also be specified.

The level of resiliency (none, weak, strong) is specified, and we use the fault check function to handle weak resiliency for silent data corruption by defining a custom check on the consumed data. Analogous to a checksum, it is a custom function computed by the producer and consumer for checking a set of values (for example, a distribution) against either a ground truth to some level of certainty or computed by an approximate method that can be compared with the full analysis. For example, temperature values need to follow an expected distribution, and particle densities ought to approximate the results of an inexpensive sampling technique. More information can be found in Section 3.4.

Deliverables: Year 1: Develop schema and API for intrinsic and extrinsic properties, and demonstrate data description of simulation data. Year 2: Integrate data description with dataflow layers and transport layers, and demonstrate decoupled dataflow driven by data descriptions. Year 3: Simulate silent data corruption, and use resilience information in data description to detect and correct errors.

3.2 Objective 2: Transport Layer

The Decaf transport layer handles all data communication between producers and consumers and intermediate nodes of the dataflow described in Section 3.3. The internal communication within a single producer

or consumer (such as a parallel numerical simulation application or visualization tool) does not need to be altered, an important practical consideration for successful adoption by the community. Unlike its predecessors, the Decaf transport layer provides much more than simple point-to-point or group data transfer. It provides a higher abstraction level and provides communication patterns and synchronization primitives for coupling applications. To do so, a number of challenges need to be addressed.

The Decaf transport layer needs to bridge multiple applications and their respective internal communication methods. While communication within a single application is well understood and provided by many mature software projects (DMAP [8], PAMI [37]) or as part of the programming model (MPI [55], UPC [10], CAF [47]), communication *between* multiple independently developed applications is unsolved. Currently, tightly coupled applications typically use MPI, which provides good portability, performance and support for structured communication. However, while MPI supports multiple-program multiple-data (MPMD) programming, it was primarily designed for a single-program multiple-data (SPMD) environment. For a tightly coupled workflow, in which both producer and consumer are specifically adapted and are aware of each other, using MPI is fairly straightforward. In a decoupled dynamic environment for which the overall data flow is not known a priori, however, adopting a SPMD-style, even by extending some concepts [15], is exceedingly difficult. The Decaf transport layer takes a dataflow-driven approach described below.

Decaf requires the transport layer to take on additional responsibilities. For example, buffering in Decaf is part of the transport. Another difference between the Decaf transport layer and most existing communication solutions is that the Decaf transport layer autonomously decides where to route the data; no explicit destinations or sources ever need to be mentioned. Instead, this information is derived from the dataflow and current system state. Others have investigated similar publish-subscribe models [19, 20]; Decaf extends these principles beyond point-to-point and adds support for MxN topologies, an essential feature for parallel workflows.

A third challenge is related to resilience. Although the transport layer connects all of the producers and consumers, a failing or misbehaving entity should not stall overall progress or endanger stability, and the transport layer should contain faults to the entity where they originated. This places unusual demands on the transport layer, such as the ability to replace a single entity without having to restart the overall dataflow and the ability to bypass failed components. While resilience has been an active research area for a number of years, the focus has been on intra-application resilience; Section 3.4 provides some solutions to *inter*application resilience.

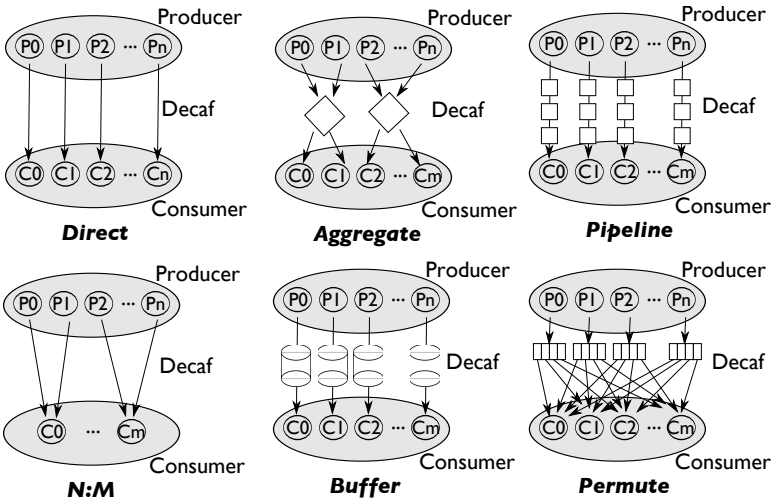


Figure 7: Major Decaf coupling modes.

We will leverage the following tools that we have developed in the past to rapidly prototype and develop the concepts above: data description and marshalling from DIY [51, 50], interapplication communication from software such as Mercury, Nessie and IOFSL [57, 38, 3], in-system buffering [35] and resilience [9]. We will reuse and adapt these existing solutions to ensure quick and efficient development of this critical transport layer.

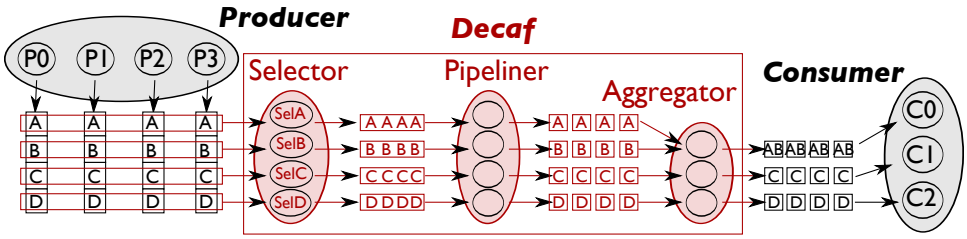


Figure 8: Coupling example with a data permutation and pipeline-able consumer. The producer generates columns of A,B,C,D while the consumer requires rows of A, rows of B, and so forth as soon as they are available.

3.2.1 Data Flow Primitives

Figure 7 shows a number of common communication patterns between producer and consumer. The patterns shown are not exclusive and can be combined to form new patterns. For example, an N:M coupling can include aggregation, pipelining, buffering, and even an optional data permutation. All these modes and their combinations can be implemented by using just four operators that make up Decaf's coupling primitives: **pipeliner, selector, aggregator, and buffer**. For example, the direct and pipeline coupling modes are provided by the pipeliner; direct coupling is treated as a special case of pipelining in which the basic pipeline unit is the full dataset. The N:M and aggregate modes are performed by the aggregator. The buffer primitive handles rate mismatch between producer and consumer and can provide persistence. Permutations are performed by using the selector and aggregator. The choice of primitives, their parameters, and their combination in a coupling dataflow are determined automatically by Decaf (see Section 3.3) using the data definition described in Section 3.1.

3.2.2 Usage

To see how this works in practice, consider Figure 8, motivated by the enstrophy computation of a 2D CFD vector field. For simplicity, we abstracted the data as A, B, C, D, but (A, B) could stand for velocity components (v_x, v_y), and C, D could represent pressure and temperature. As an abstraction, this model can represent many other analyses that consist of an N:M coupling of producer and consumer with a pipelined data permutation consisting of a selection and aggregation. The producer generates columns of A, B, C, D, while the consumer requires rows of A, rows of B, and so forth. Moreover, the consumer requires A and B to be combined (enstrophy is a vector quantity). Given the description of the data generated by the producer and the data expected by the consumer (Section 3.1), the Decaf dataflow model (Section 3.3) derives an optimal dataflow (including buffering where needed) as shown in red in Figure 8, in essence inserting intermediate producers and consumers from the primitives listed above. From this blueprint for data movement, at execution time each producer repeatedly provides the dataflow layer with data, and each consumer calls into the dataflow layer to receive data, *without the need to specify source or destination*. The latter is important because it effectively decouples each producer and consumer from the overall flow and enables them to be oblivious to faults in other components or to changes in the dataflow graph.

Deliverables: Year 1: Design the basic API for four primitives and implement the pipeliner and buffer. Year 2: Develop aggregator and selector and evaluate all four primitives in proxy applications. Year 3: Test and integrate resilience in the transport layer.

3.3 Objective 3: Dataflow Model

Now that the data model has been described and the transport primitives are defined, the next step is to synthesize a completed dataflow model for the desired coupling. Usually, more than one realization of a dataflow

is possible. For example, in Figure 8, the order of the pipeliner and selector tasks could be exchanged, and the number of nodes in the aggregator could vary. We will study different deployment strategies of coupling primitives to satisfy the data description and how to select the best dataflow.

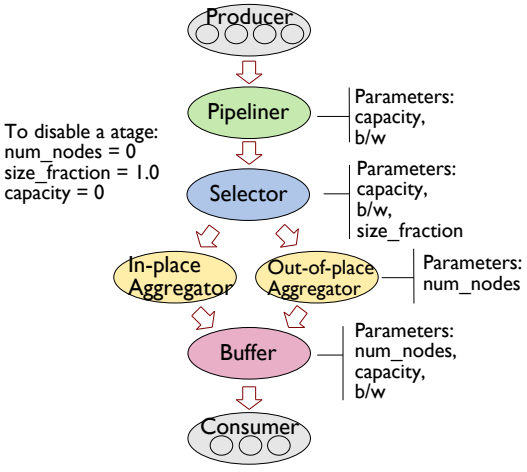
3.3.1 Dataflow Synthesis

We will research methods to optimize the dataflow and search for the best coupling that satisfies some set of constraints. Example constraints we must satisfy are the following: (1) the amount of data resident at any point in time cannot exceed the node capacity; (2) the amount of data moved into or out from a node should not exceed the node bandwidth; (3) the location of certain data may be prescribed by the input workflow; for example, initial input comes from disk and final output goes to disk; (4) the rate of data computation is limited by the producer's bandwidth profile; and (5) the bandwidth and capacity may need to accommodate a weak resilience check (see Section 3.4.2) that could increase the data size. These constraints are determined by the input workflow and data description. Additionally, certain heuristics may either be generally desirable or depend on the actual input data. Examples of such heuristics include whether the pipeliner precedes most other tasks (pipeline early), whether the aggregator should precede other tasks (aggregate early), and whether aggregation should be in-place or at external nodes depending on the actual data size.

3.3.2 Parameterized Time-Space Graphs

The synthesis of a dataflow composed of primitives may be framed as an optimization problem. Optimization research is not our primary concern, but we do want to understand whether and how the dataflow can be posed as an optimization problem so that existing tools can be used to solve it. We will not change the number of nodes in the producer or consumer; the input workflow dictates those parameters. But we will (a) vary the number of nodes in the Decaf internal dataflow, including allocating more nodes than in the original producer / consumer, (b) optimize for transmission time, (c) consider buffering (including deep storage), and (d) explore different resilience strategies.

One of the formulations we are investigating is a *parameterized time-space graph* (left side of Figure 9) that represents most of the possible combinations of coupling an M-node producer with an N-node consumer. As a motivating example, we are analyzing simulations of superconductivity for temporal events such as pinning/depinning of magnetic field vortices at inclusions in a superconducting material [53]. Such temporal



Minimize:
t, where
$t = t\_pipe + t\_sel + t\_aggr + t\_buf$
$t\_sel = sel\_size\_frac \times producer\_data\_size / selector\_bw$
$t\_aggr = f(num\_aggr\_nodes, sel\_size\_frac \times producer\_data\_size)$
$t\_buf = enable \times sel\_size\_frac \times producer\_data\_size / buf\_bw$

Subject to:
$pipeliner\_unit\_size \leq pipeliner\_capacity$
$pipeliner\_unit\_size \times sel\_size\_frac \leq selector\_capacity$
$pipeliner\_unit\_size \times sel\_size\_frac / num\_aggr\_nodes \leq aggr\_node\_capacity$
$pipeliner\_unit\_size \times sel\_size\_frac \leq writer\_capacity$

Figure 9: Left:Various combinations of M:N producer-consumer coupling represented in a single parameterized time-space graph. Right: Linear optimization of parameterized time-space graph.

activity summaries require most of the nodes in the parameterized time-space graph, including out-of-place aggregation and buffering. The placement of various intermediate nodes in the dataflow (pipeliner, selector, in-place and out-of-place aggregators, buffer) can be parameterized via the posing of the optimization problem (right side of Figure 9). In this example the wall-clock time t is being minimized, but other objective functions such as power or bytes moved could similarly be minimized. We will experiment with formulating different objective functions and their effect on extreme-scale characteristics such as power and time to solution.

To fill in some of the gaps in our understanding of the components of these formulas, we will leverage existing work and extend it to support the needs of our system. The I/O Containers [13] project demonstrated a management infrastructure for monitoring online workflows with capabilities for adjusting the resource allocations. A complicating factor not considered thus far is the impact of data transmission times. While previous work has identified the importance of considering this overhead [65], it did not enumerate all the considerations of placement for dataflow components. Rather, the intent was to describe an effective management infrastructure for maintaining a deployment that has been determined elsewhere. Our proposal, in contrast, seeks to combine these two considerations into a unified whole that addresses not just deciding where to place operators but also how to monitor and maintain minimum resource usage and maximum throughput by balancing use of network bandwith against available compute resources. Doing so in a set of reusable standardized components is another novel contribution, but it also raises research challenges.

The key to our dataflow is to consider not just the data producers but also the consumers. Unlike other projects that take a passive view of data consumers in deployment decisions, we propose incorporating these consumers into the deployment decision. A straightforward approach for doing so requires an understanding of the throughput of a component, that is, the rate at which it can expect and successfully process data, given by the data description layer. By balancing the output volume and frequency against these throughput measures, we can not only balance the resource deployments but also predetermine whether we should incorporate additional resources or just redirect late stages of the dataflow to persistent storage. Since the I/O bandwidth, at least within a reasonable approximation, is knowable, this factor can be incorporated into the equations as well.

In addition to accommodating short-term data rate mismatches, buffering serves two other purposes: aggregation (for example, to permute time-varying data) and resilience (for example, to localize faults). Including various resilience strategies to minimize resources, energy, and interruption in our optimization problem is another topic we will study and a unique feature of our proposal. Each node of our dataflow may use different fault tolerance techniques, and overall resilience depends on the combination of these techniques in every node. We will include models of different fault tolerance techniques in our dataflow and explore heuristics to find efficient configurations. Furthermore, we will investigate how to encode "desired or optional" resources in our dataflow model. Given sufficient resources, alternative optimizations can be performed; for example, we can consider allocating additional aggregator or buffer nodes to improve fault tolerance and performance.

Deliverables Year 1: Develop an effective deployment calculation and deployment strategy system that takes into account the various factors described above. Application workflows will be used to refine the model and demonstrate efficacy. Year 2: Implement the deployment calculation into a prototype system, and demonstrate that it can successfully generate appropriate job scripts for executing the workflows. Year 3: Incorporate resilience features into the calculations and generated job scripts.

3.4 Cross-cutting: Resilience

The expected hardware and software fault rates of future HPC platforms [9, 18, 26, 56] mandate the incorporation of resilience techniques in HPC software, and extreme-scale data analysis is certainly no exception

in this regard. While resilience is not the primary focus of the proposed research, it pervades nearly every aspect of it because processes may fail during a dataflow execution, and/or data may be silently corrupted. From specifying the level of protection in the data description layer to using buffering as a fault containment mechanism in the transport layer and optimizing placement of resilience measures in the dataflow layer, resilience must be addressed everywhere in order for our solution to be usable in future extreme-scale settings.

3.4.1 Resilience to Process Failure

In this proposal, we expanded original workflow links into additional nodes and edges in the transport and dataflow layers; these additional entities also need to be tolerant to process failure. The objective of fault tolerance is to guarantee progress, and our efforts will focus on identifying the best replacement and restart approaches for each node that optimize the dataflow. The naive approach, restarting the full workflow from the beginning [36], is prohibitive in time, resources, and energy. Two other approaches are possible. The first is that all nodes of the augmented dataflow are replicated [34]; replication allows continuation of service in presence of failures, but its main drawback is its cost. A second approach is node restart: nodes may restart from the beginning or from a checkpoint [36, 62]. As shown in Figure 10, if the producer (node 1) fails, the producer may regenerate the same data during the restart and the rest of the dataflow will need to filter the data until new data are generated. A failure internal to the Decaf dataflow (node 4) may require stopping and resuming messaging using flow control. The main advantage of node restart over replication is the significant reduction in resource and energy needs; however, restart cannot guarantee uninterrupted execution. Additionally, the user may define desired mitigation approaches for node failures. The user may simply accept a gap in the flow of data (for example, one frame of a movie) or specify a function to execute in the producer or consumer to fill the gap (for example, interpolation between two adjacent frames). We will explore approaches to skip a faulty node and approximate the missing data.

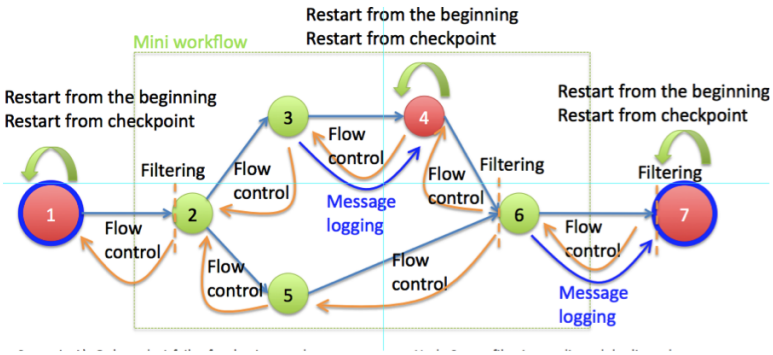


Figure 10: Basic resilience techniques for our dataflow. Scenario A: Node 1 fails, and node 2 uses filtering to discard duplicated messages. Scenario B: Node 4 fails, and node 3 may use flow control for message generation.

3.4.2 Resilience to Silent Data Corruption

We can tune the fault tolerance technique based on the user description of data in Section 3.1 to three different degrees of criticality: “None,” “Strong,” and “Weak.” The “None” mode simply means that the data are not critical, and the system selects the appropriate responses to faults. In the “Strong” mode, the user requires maximum data integrity, so our dataflow will use the fault tolerance techniques with the least number of false negatives. Replication is currently the most effective approach for detecting data corruption, and data corruption detection techniques based on replication are fairly well understood [54, 23, 24, 58].

We focus our attention on the “Weak” mode, where the user specifies custom corruption detection methods to be used. We envision several approaches: (1) checking preservation of global invariants in the data (total energy, total mass, checksum of particle positions), (2) comparing analyses with low-cost approximate methods, and (3) comparing output data distribution with expected data distributions. To illustrate (2), we

consider an example from cosmology in which particle density is estimated onto a regular grid. In preliminary research, we compared the advantages of using Voronoi tessellations over lower-cost kernels for density estimation. Figure 11 shows how tessellations better represent sparse regions of density compared with cloud-in-cell (CIC) kernels. Tessellations are too large for full replication, motivating a weak resilience strategy. Fortunately, CIC methods are cheap to compute and can be done on the fly, motivating their use as a low-cost resilience check of the high-quality analysis. We will consider several data analytics workflows in real application scenarios and will explore, in addition to using approximate low-cost analyses such as CIC, the conservation of global data invariants and expected data distributions to verify correctness.

Deliverables: Year 1: Optimize the combination of fail-over strategies for nodes in a dataflow to satisfy "Strong," "Weak," and "None" levels of data criticality. Year 2: Develop weak resilience to silent data corruption based on global invariants, approximate solutions, and expected results. Year 3: Evaluate Year 1 and 2 solutions in proxy and actual applications.

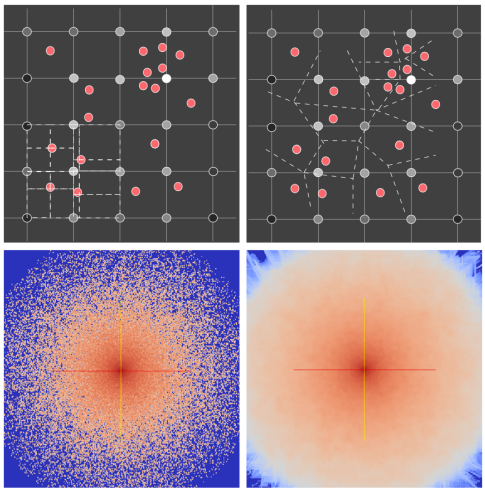


Figure 11: Density estimation using (left) an inexpensive CIC kernel and (right) an expensive but accurate tessellation kernel. We can use the CIC as a quick weak resilience check of the tessellation.

4 Management and Personnel

This effort brings together primarily early-career researchers with complementary strengths: each member brings a specific area of expertise to the project. Peterka has worked closely with science and visualization applications to demonstrate extreme-scale in situ analysis algorithms with DIY. Lofstead helped develop coupling technologies in ADIOS and Nessie. Kimpe provides storage and I/O forwarding skills with projects such as IOFSL and Mercury. Adding expertise to the team, Cappello is a renowned expert in resilience for HPC and is interested in applying those techniques to data analysis. Peterka will lead the overall management of the project by overseeing the collaboration and communication between the team members, and he will report to the DOE on progress and milestones. His technical focus will be the data description layer and interfacing with science codes. Lofstead will design the dataflow layer. He will also evaluate the integration of the transport and dataflow layers to synthesize dataflows from the primitive kernels of the transport layer. Kimpe will be responsible for interfacing with the low-level transport system software and for implementing the dataflow primitives; he will work with Lofstead to integrate them into the dataflow. Cappello will coordinate the resilience integration in all three layers, and he will also oversee the alignment of our strategies with emerging extreme-scale software and hardware computing environment.

The team will meet monthly via teleconference and annually in person to discuss the current state of the effort as well as to plan and prioritize the next steps. We will set up an online presence for the project consisting of a website, mailing list, code repository, documentation, and publications related to the work. All source code developed as part of this project will be open and available to the public through this site.

4.1 Evaluation Plan and Project Timetable

Three common use cases will drive our research. (1) We will decouple data streaming in the in situ analysis of DOE simulations, for example to trace particles in nuclear reactor Navier-Stokes computations. (2) We will study data pipelining with several independent steps of in situ analysis, for example to identify features

in Ginzburg-Landau superconductivity simulations and then to track those features as they evolve over time. (3) We will decouple dataflow networks in a workflow that transforms particles, meshes, and grids in Vlasov-Poisson N-body cosmology codes. We will complete the milestones in Figure 12 as part of our proposed effort, and we will package our work in a prototype library called Decaf that will be able to be deployed in out-years through production-oriented funding mechanisms such as SciDAC.

4.2 Collaboration with Related Projects and Community

Our project is designed to integrate with and contribute to four other efforts in the ASCR portfolio: CESAR, SDAV, Argo, and Hobbes. In the CESAR co-design center, the focus is simulation-simulation coupling, not simulation-analysis coupling in complex analysis workflows. We will target this gap. SDAV, the SciDAC-3 data effort, deploys coupling of custom and predefined simulation and analysis for specific science use cases, but not in a general reusable toolkit such as ours.

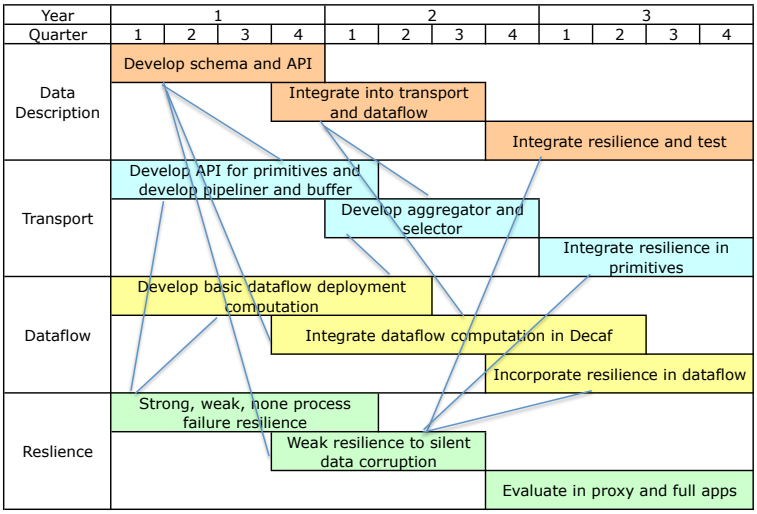


Figure 12: Project deliverable timeline. Lines indicate related or dependent tasks where PIs will work together. For clarity, only a subset of connections are shown, but in reality the project is highly interconnected.

Argo and Hobbes are OS/R efforts in extreme-scale runtimes and operating systems that partition systems by enclaves. They do not provide any API or runtime support specifically for data processing and data coupling: our proposal addresses this gap. It is projected that new OS/R research will offer useful software capabilities for Decaf, and we will design for these future capabilities such as global control backplanes and global view optimization. Our work will also help steer such projects toward designing system software to improve data-intensive tasks. Through our close collaborations with these projects, we can provide specific instances of data needed, for example, in the backplane to support weak resilience in dataflows.

Our team members are part of a broader community of researchers dedicated to extreme-scale data management and analysis. In fact, three of our team members participated or contributed to the recent BDEC workshop in Fukuoka, Japan. [5] We all participate regularly in PI meetings to update the community on ongoing projects, and we are active in technical program committees of many international conferences including IEEE/ACM SC, IEEE ISC, ACM ICS, IEEE IPDPS, and IEEE Vis. All the team members have a long track record of publishing in these conferences as well, and we anticipate that Decaf will result in several publications per year. Furthermore, all PIs advise several Ph.D. students each year. Argonne's Mathematics and Computer Science Division sponsors 8-10 interns for 10 weeks each summer, as does Sandia. Students will work closely with scientists, postdocs, and software developers on this project in assisting with research, design, deployment, and maintenance of production-quality software. All software produced will be open source and made publicly available for download. We plan to release under a BSD-style license, as approved by the Open Source Inititative. [6]

[5] http://www.exascale.org/bdec/

[6] http://www.opensource.org/licenses/bsd-license.html

4.3 Personnel

Tom Peterka is an assistant computer scientist at Argonne National Laboratory, fellow at the Computation Institute of the University of Chicago, and an adjunct assistant professor at the University of Illinois at Chicago. His research interests are in large-scale parallelism for in situ analysis of scientific datasets. His work has led to two best paper awards and publications in ACM SIGGRAPH, IEEE VR, IEEE TVCG, and ACM/IEEE SC, among others. Peterka received his Ph.D. in computer science from the University of Illinois at Chicago, and he currently works actively in several DOE ASCR projects including the CESAR co-design center and the SDAV SciDAC-3 institute.

Dries Kimpe is an assistant computer scientist at Argonne National Laboratory and visiting lecturer at the University of Illinois at Chicago. He is a fellow at the Computation Institute of the University of Chicago and an Institute Fellow at the Nortwestern-Argonne Institute for Science and Engineering at Northwestern University. He has been working on data analysis and I/O challenges for over a decade. He obtained his master's degree from the University of Ghent (Belgium) and his Ph.D. from the Catholic University of Leuven (Belgium). His current research interests include parallel I/O, distributed file systems, and parallel programming models.

Franck Cappello is an internationally renowned leader in resilience for HPC systems. He is member of the DoE Resilience Tech Council and has led the resilience roadmapping effort for IESP (International Exascale Software Project) and EESI (European Exascale Software Initiative). He has published many papers covering most of the aspects of resilience for HPC systems, including checkpoint/restart, fault-tolerant protocols, failure prediction, fault tolerance, performance modeling and optimization, fault injection, and SDC detection. Cappello is regularly invited as keynote speaker or invited speaker in major conferences and workshops (CCGRID, Cluster, Europar, etc.).

Jay Lofstead is a senior member of technical staff at Sandia National Laboratories in Albuquerque, NM in the Scalable System Software group. His primary responsibilities are to explore data management and workflow issues for supporting big science applications. Previously, he co-invented the 2013 R&D 100 award winning ADIOS IO API. Prior to graduate school he was the primary architect and developer for the McKesson Provider Technologies Portal tool that provides visual integration of multiple back-end healthcare systems with a user-cutomizable workflow. Earlier, he developed programming packages for Siemens Energy and Automation for their embedded industrial controls. Lofstead is a three-time Georgia Tech graduate, most recently in 2010 with his Ph.D. in computer science.

Appendix 3: Bibliography and References Cited

References

[1] Hasan Abbasi, Greg Eisenhauer, Matthew Wolf, Karsten Schwan, and Scott Klasky. Just in Time: Adding Value to the I/O Pipelines of High Performance Applications with JITStaging. In *Proceedings of the 20th international symposium on High performance distributed computing*, pages 27–36. ACM, 2011.

[2] Hasan Abbasi, Matthew Wolf, and Karsten Schwan. LIVE Data Workspace: A Flexible, Dynamic and Extensible Platform for Petascale Applications. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 341–348, Washington, DC, USA, 2007. IEEE Computer Society.

[3] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O Forwarding Framework for High-Performance Computing Systems. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009.

[4] Benjamin A Allan, Robert C Armstrong, Alicia P Wolfe, Jaideep Ray, David E Bernholdt, and James A Kohl. The CCA Core Specification in a Distributed Memory SPMD Framework. *Concurrency and Computation: Practice and Experience*, 14(5):323–345, 2002.

[5] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 115–124. IEEE, 1999.

[6] S. Bhattacharya, S. Habib, and K. Heitmann. Dark Matter Halo Profiles of Massive Clusters: Theory vs. Observations. *ArXiv e-prints*, December 2011.

[7] Allan Bricker, Michael Litzkow, and Miron Livny. Condor Technical Summary. URL `ftp://ftp.cs.wisc.edu/condor/DocOnly_4.1.3b.tar.Z`, October 1991.

[8] M Bruggencate and D Roweth. DMAPP: An API for One-Sided Programming Model on Baker Systems. *Cray Users Group (CUG)*, 2010.

[9] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward Exascale Resilience. *International Journal of High Performance Computing Applications*, 23(4):374–388, 2009.

[10] William W Carlson, Jesse M Draper, David E Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. *Introduction to UPC and Language Specification*. Center for Computing Sciences, Institute for Defense Analyses, 1999.

[11] Abon Chaudhuri, Teng-Yok Lee, Bo Zhou, Cong Wang, Tiantian Xu, Han-Wei Shen, Tom Peterka, and Yi-Jen Chiang. Scalable Computation of Distributions from Large Scale Data Sets. In *Proceedings of the 2012 IEEE Large Data Analysis and Visualization Symposium LDAV'12*, Seattle, WA, 2012.

[12] Jai Dayal, Drew Bratcher, Hasan Abbasi, Greg Eisenhauer, Scott Klasky, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics. In *Cluster, Cloud, and Grid*, CCGrid '14. IEEE, 2014.

[13] Jai Dayal, Jianting Cao, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Fang Zheng, Hasan Abbasi, Scott Klasky, Norbert Podhorszki, and Jay Lofstead. I/O Containers: Managing the Data Analytics and Visualization Pipelines of High End Codes. In *In Proceedings of International Workshop on High Performance Data Intensive Computing (HPDIC 2013) held in conjunction with IPDPS 2013*, Boston, MA, 2013. Best Paper Award.

[14] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, and John Good. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming*, 13(3):219–237, 2005.

[15] Erik D. Demaine, Ian Foster, Carl Kesselman, and Marc Snir. Generalized Communicators in the Message Passing Interface. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):610–616, June 2001.

[16] M. O. Deville, P. F. Fischer, and E. H. Mund. *High-Order Methods for Incompressible Fluid Flow*. August 2002.

[17] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. *HPDC '10: Proceedings of the 18th international symposium on High performance distributed computing*, 2010.

[18] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. The International Exascale Software Project Roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.

[19] Greg Eisenhauer, Karsten Schwan, and Fabián E Bustamante. Publish-Subscribe for High-Performance Computing. *Internet Computing, IEEE*, 10(1):40–47, 2006.

[20] Greg Eisenhauer, Matthew Wolf, Hasan Abbasi, and Karsten Schwan. Event-Based Systems: Opportunities and Challenges at Exascale. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 2. ACM, 2009.

[21] Jack Dongarra et al. International Exascale Software Project Roadmap. Technical report, 2012. www.exascale.org.

[22] Steve Ashby et al. The Opportunities and Challenges of Exascale Computing: Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Technical report, 2010.

[23] Kurt Ferreira, Rolf Riesen, Ron Oldfield, Jon Stearley, James Laros, Kevin Pedretti, and T Brightwell. rMPI: Increasing Fault Resiliency in a Message-Passing Environment. *no. SAND2011-2488, Sandia National Laboratories, Tech. Rep*, 2011.

[24] Kurt Ferreira, Jon Stearley, James H Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G Bridges, and Dorian Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2011.

[25] P. Fischer, J. Lottes, D. Pointer, and A. Siegel. Petascale Algorithms for Reactor Hydrodynamics. *Journal of Physics Conference Series*, 125(1):012076–+, July 2008.

[26] Al Geist, Bob Lucas, Marc Snir, Shekhar Borkar, Eric Roman, Mootaz Elnozahy, Bert Still, Andrew Chien, Robert Clay, John Wu, Christian Engelmann, Nathan DeBardeleben, Rob Ross, Larry Kaplan, Martin Schulz, Mike Heroux, Sriram Krishnamoorthy, Lucy Nowell, Abhinav Vishnu, and Lee-Ann Talley. U.S. Department of Energy Fault Management Workshop. http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/FaultManagement-wrkshpRpt-v4-final.pdf, 2012.

[27] Andreas Glatz, Igor Aranson, Valerii M Vinokur, Nikolay M Chtchelkatchev, and Tatyana I Baturina. Emergence of Superconducting Textures in Two Dimensions. *arXiv preprint arXiv:0910.0659*, 2009.

[28] Zhenhuan Gong, D.A. Boyuka, Xiaocheng Zou, Qing Liu, N. Podhorszki, S. Klasky, Xiaosong Ma, and N.F. Samatova. PARLO: PArallel Run-Time Layout Optimization for Scientific Data Explorations with Heterogeneous Access Patterns. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 343–351, May 2013.

[29] Attila Gyulassy, Tom Peterka, Valerio Pascucci, and Robert Ross. Characterizing the Parallel Computation of Morse-Smale Complexes. In *Proceedings of IPDPS '12*, Shanghai, China, 2012.

[30] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. HACC: Extreme Scaling and Performance Across Diverse Architectures. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 6:1–6:10, New York, NY, USA, 2013. ACM.

[31] Salman Habib, Vitaly Morozov, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Tom Peterka, Joe Insley, D. Daniel, Pat Fasel, Nick Frontiere, and Z. Lukic. The Universe at Extreme Scale: Multi-Petaflop Sky Simulation on the BG/Q. In *Proceedings of SC12*, Salt Lake City, UT, 2012.

[32] K. Heitmann, M. White, C. Wagner, S. Habib, and D. Higdon. The Coyote Universe. I. Precision Determination of the Nonlinear Matter Power Spectrum. *Astrophysical Journal*, 715:104–121, May 2010.

[33] Tong Jin, Fan Zhang, Qian Sun, Hoang Bui, Manish Parashar, Hongfeng Yu, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. Using Cross-Layer Adaptations for Dynamic Data Management in Large Scale Coupled Scientific Workflows. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 74:1–74:12, New York, NY, USA, 2013. ACM.

[34] Gopi Kandaswamy, Anirban Mandal, and Daniel A Reed. Fault Tolerance and Recovery of Scientific Workflows on Computational Grids. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 777–782. IEEE, 2008.

[35] Dries Kimpe, Kathryn Mohror, Adam Moody, Brian Van Essen, Maya Gokhale, Rob Ross, and Bronis R. de Supinski. Integrated In-System Storage Architecture for High Performance Computing. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, pages 4:1–4:6, New York, NY, USA, 2012. ACM.

[36] Sven Köhler, Sean Riddle, Daniel Zinn, Timothy McPhillips, and Bertram Ludäscher. Improving Workflow Fault Tolerance Through Provenance-Based Recovery. In *Scientific and Statistical Database Management*, pages 207–224. Springer, 2011.

[37] Sameer Kumar, Amith R Mamidala, Daniel A Faraj, Brian Smith, Michael Blocksome, Bob Cernohous, Douglas Miller, Jeff Parker, Joseph Ratterman, Philip Heidelberger, et al. PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 763–773. IEEE, 2012.

[38] Jay Lofstead, Ron Oldfield, Todd Kordenbrock, and Charles Reiss. Extending Scalability of Collective I/O Through Nessie and Staging. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 7–12. ACM, 2011.

[39] Jay Lofstead and Karsten Schwan. XCHANGE: High Performance Data Morphing in Distributed Applications. Technical Report GIT-CC-05-14, Georgia Institute of Technology, College of Computing, 2005.

[40] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Input/Output APIS and Data Organization for High Performance Scientific Computing. In *Petascale Data Storage Workshop, 2008. PDSW'08. 3rd*, pages 1–6. IEEE, 2008.

[41] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, Metadata Rich IO Methods for Portable High Performance IO. Rome, Italy, 2009.

[42] Jay F Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible I/O and Integration for Scientific Codes Through the Adaptable I/O System (ADIOS). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24. ACM, 2008.

[43] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific Workflow Management and the Kepler System: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[44] G. Malewicz, I. Foster, A.L. Rosenberg, and M. Wilde. A Tool for Prioritizing DAGMan Jobs and its Evaluation. *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 156–168, 0-0 2006.

[45] E. Merzari, W.D. Pointer, A. Obabko, and P. Fischer. On the Numerical Simulation of Thermal Striping in the Upper Plenum of a Fast Reactor. In *Proceedings of ICAPP 2010*, San Diego, CA, 2010.

[46] Boonthanome Nouanesengsy, Teng-Yok Lee, Kewei Lu, Han-Wei Shen, and Tom Peterka. Parallel Parrticle Advection and FTLE Computation for Time-Varying Flow Fields. In *Proceedings of SC12*, Salt Lake City, UT, 2012.

[47] Robert W Numrich and John Reid. Co-Array Fortran for Parallel Programming. In *ACM Sigplan Fortran Forum*, volume 17, pages 1–31. ACM, 1998.

[48] Ron A. Oldfield, David E. Womble, and Curtis C. Ober. Efficient Parallel I/O in Seismic Imaging. *The International Journal of High Performance Computing Applications*, 12(3):333–344, Fall 1998.

[49] Tom Peterka, Juliana Kwan, Adrian Pope, Hal Finkel, Katrin Heitmann, Salman Habib, Jingyuan Wang, and George Zagaris. Meshing the Universe: Integrating Analysis in Cosmological Simulations. In *Proceedings of the SC12 Ultrascale Visualization Workshop*, Salt Lake City, UT, 2012.

[50] Tom Peterka and Robert Ross. Versatile Communication Algorithms for Data Analysis. In *EuroMPI Special Session on Improving MPI User and Developer Interaction IMUDI'12*, Vienna, AT, 2012.

[51] Tom Peterka, Robert Ross, Wesley Kendall, Attila Gyulassy, Valerio Pascucci, Han-Wei Shen, Teng-Yok Lee, and Abon Chaudhuri. Scalable Parallel Building Blocks for Custom Data Analysis. In *Proceedings of the 2011 IEEE Large Data Analysis and Visualization Symposium LDAV'11*, Providence, RI, 2011.

[52] Tom Peterka, Robert Ross, Boonthanome Nouanesengsy, Teng-Yok Lee, Han-Wei Shen, Wesley Kendall, and Jian Huang. A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields. In *Proceedings of IPDPS 11*, Anchorage AK, 2011.

[53] Ivan Sadovskyy and Andreas Glatz. Time-Dependent Ginzburg-Landau Equations and Vortex Dynamics Simulations on GPUs. *Bulletin of the American Physical Society*, 2014.

[54] D.P. Siewiorek and R.S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, 1982.

[55] Marc Snir. *MPI–The Complete Reference: The MPI Core*, volume 1. MIT press, 1998.

[56] Marc Snir, Robert W. Wisniewski, Jacob A. Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji Jim Belak Pradip Bose Franck Cappello Bill Carlson, Andrew A. Chien, Paul Coteus, Nathan A. Debardeleben, Pedro Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. Addressing Failures in Exascale Computing. http://www.mcs.anl.gov/publication/addressing-failures-exascale-computing-0, 2013. to appear in International Journal of High Performance Computing.

[57] Jerome Soumagne, Dries Kimpe, Judicael Zounmevo, Mohamad Chaarawi, Quincey Koziol, Ahmad Afsahi, and Robert Ross. Mercury: Enabling Remote Procedure Call for High-Performance Computing. In *Proceedings of the IEEE Cluster 2013 Conference*, Indianapolis, IN, 2013.

[58] J. Stearley, K. Ferreira, D. Robinson, J. Laros, K. Pedretti, D. Arnold, P. Bridges, and R. Riesen. Does Partial Replication Pay Off? In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6, June 2012.

[59] T. Stef-Praun, B. Clifford, I. Foster, U. Hasson, M. Hategan, S. L. Small, M. Wilde, and Y. Zhao. Accelerating Medical Research Using the Swift Workflow System. *Studies in Health Technology and Informatics*, 126:207–216, 2007.

[60] Rick Stevens and Andy White. DOE Laboratory Plan for Providing Exascale Applications and Technologies for Critical DOE Mission Needs. Technical report, 2010. www.exascale.org.

[61] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. YAWL: Yet Another Workflow Language. *Information systems*, 30(4):245–275, 2005.

[62] Mingzhong Wang, Liehuang Zhu, and Jinjun Chen. Risk-Aware Checkpoint Selection in Cloud-Based Scientific Workflow. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 137–144, Nov 2012.

[63] Jonathan Woodring, Katrin Heitmann, James Ahrens, Patricia Fasel, Chung-Hsing Hsu, Salman Habib, and Adrian Pope. Analyzing and Visualizing Cosmological Simulations with ParaView. (arXiv:1010.6128. LA-UR-10-06301), Nov. 2010.

[64] Justin M Wozniak, Tom Peterka, Timothy G Armstrong, James Dinan, Ewing Lusk, Michael Wilde, and Ian Foster. Dataflow Coordination of Data-Parallel Tasks via MPI 3.0. 2013.

[65] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Scott Klasky, Qing Liu, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. PreDatA - Preparatory Data Analytics on Peta-Scale Machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.