SMARTBLOCK: An Approach to Standardizing In Situ Workflow Components

Alexis Champsaur*, Jay Lofstead†, Jai Dayal*, Matthew Wolf‡,
Greg Eisenhauer*, Patrick Widener†, Ada Gavrilovska*
*School of Computer Science, Georgia Tech, Atlanta, GA, USA
Email: {alexis.champsaur,dayalsoap}@gmail.com, {ada,eisen}@cc.gatech.edu
† Scalable System Software Group, Sandia National Laboratories, Albuquerque, NM, USA
Email: {gflofst,pwidene}@sandia.gov
‡ Scientific Data Group, Oak Ridge National Lab, Oak Ridge, TN, USA
Email: wolfmd@ornl.gov

***Abstract*—Multi-step scientific workflows have become prominent and powerful tools of data-driven scientific discovery. Runtime analytic techniques are now commonly used to mitigate the performance effects of using parallel file systems as staging areas during workflow execution. However, workflow construction and deployment for extreme-scale computing is still largely an *ad hoc* process with uneven support from existing tools. In this paper, we present SMARTBLOCK, an approach to designing generic, reusable components for end-to-end construction of workflows. Specifically, we demonstrate that a small set of SMARTBLOCK generic components can be reused to build a diverse set of workflows, using examples based on actual analytic processes with three well-known scientific codes. Our evaluation shows promising scaling properties as well as negligible overheads for using a modular approach over a custom, "all-in-one" solution. As extreme-scale systems incorporate data analytics on simulation data as it is generated at rates that far outstrip available I/O bandwidth, tools such as SMARTBLOCK will become increasingly valuable for defining and deploying flexible, efficient workflows.**

I. INTRODUCTION

Multi-step scientific simulation and engineering workflows have become increasingly prominent as extreme-scale computing environments evolve. Made up of collections of coupled analytic and/or data processing steps designed to refine data into information, such workflows are becoming powerful tools of data-driven scientific discovery. Performance issues caused by storage I/O between workflow stages become prohibitive as such workflows grow in size and complexity, a situation unlikely to improve given the relative future projections of storage system bandwidth and system compute capability. So-called *in situ* techniques, or *runtime data analysis*, in which analysis and visualization components perform data reductions and transformations on data in memory, are now being widely used to mitigate these issues.

While tools exist for facilitating *in situ* processing between data producers and consumers, large-scale workflows are still constructed *ad hoc*. Existing tools provide limited traction on the complexities of workflow implementation: widely varying data formats; impedance mismatches between the degrees of parallelism provided by various workflow components; and component (re-)deployment. However, even as the workflows themselves exhibit these differences, many of their component analytical routines remain similar.

In this work we introduce SMARTBLOCK, an approach to designing generic, reusable components for the construction of scientific workflows from start to finish. SMARTBLOCK allows for the assembly and launch of a variety of *in situ* -capable workflows "out of the box," without recompilation and requiring only one-time modifications to the output methods of the driving simulation codes. The SMARTBLOCK demonstrations leverage both self-describing data exchanged through the Adaptable I/O System (ADIOS) [1] interface as well as the stream-based, publish/subscribe interaction model implemented by the underlying transport system, FlexPath [2]. SMARTBLOCK consists of generic components that perform commonly used operations in workflows, such as multi-dimensional data filtering and histogram analysis. We present four such components in this paper. While this set is not exhaustive, it demonstrates the standardization benefits of the SMARTBLOCK approach by describing their use in the construction and deployment of three very different workflows.

Specifically, this paper makes the following contributions:

- A description of the SMARTBLOCK approach for the construction of in situ scientific workflows; and
- Details of a representative set of *in situ* generic, reusable analytic components designed for the deployment of full SMARTBLOCK workflows;

We demonstrate the effectiveness of the approach through a performance evaluation of SMARTBLOCK using three well-known scientific simulation drivers, showing its functionality as well as its strong and weak scaling characteristics. Since this modular approach aims to replace custom code, we provide a comparison of the performance of a representative SMARTBLOCK workflow with that of a fixed "all-in-one" code performing the same analyses.

The remainder of this paper is organized as follows. First is a survey of related work in §II. Second, §III presents the design approach. Next, §IV presents the demonstration implementation details. The evaluation is presented in §V followed by conclusions and future work in §VI.

II. Related Work

Existing workflow systems have typically only been able to offer generic, reusable components when the workflow system is for a particular niche with a fixed datatype and standardized interfaces. For example, enterprise document processing systems may all work against a single database with each user only seeing their current worklist. As documents are processed, they are moved to the next work queue, or completed state, according to hand-coded rules [3]. The Workflow Management Coalition [4] has developed standards to make enterprise process workflows more portable. These standards are not intended to make components reusable, but to make different workflow engines able to inter-operate or to port a workflow from one engine to another. The actual communication interfaces and data types are left to the components themselves.

More directly related to this work and the scientific community are workflow engines and frameworks custom-made for the parallel computing environment. Pegasus [5] and DAGMan [6] work together to offer an engine to execute a workflow and a front-end system to construct the workflow process itself. However, the focus in DAGMan is on specifying dependencies between the jobs involved in the workflow, so as to execute components only when required and provide resilience. In contrast, in SmartBlock, the entire workflow is executed at once, with FlexPath allowing readers and writers to block until the corresponding writer or reader is available for data exchange. Furthermore, the Pegasus/DAGMan engine does not provide generic, reusable components.

As an alternative to DAGMan, Swift [7] is a scripting language that allows ordinary applications to be composed into parallel scripts, and eventually into workflows, with dependencies specified in the script. However, these applications themselves must be written outside of the Swift script.

Kepler [8] offers a nice GUI for assembling different kinds of scientific workflows. Each component is an actor, with channels connecting actors and a director managing the execution. Complex workflows using Kepler have been assembled for many communities. However, the large collection of components that come with Kepler are mainly designed to work in a single Java Virtual Machine instance; using HPC-scale components requires significant effort both in coding the custom components and in allowing the I/O methods used to be understood by the higher-level Kepler engine.

To address much of the complexity of communicating between separate parallel components, inline approaches are being investigated. We use "inline" in the sense of "linked into the driving process." Catalyst [9] offers a way to integrate the ParaView [10] analysis and visualization system directly into the simulation executable through explicit calls from the host application into Catalyst routines with predictable data types on in-memory data structures. While this can work for limited kinds of data processing, it clearly cannot take advantage of additional resources available for off-site analysis, instead taking cycles away from the main scientific code.

Libsim [11] has a similar relationship to VisIt [12] as Catalyst has to ParaView. As with Catalyst, key limitations of Libsim coming from the fact that it runs on the same node as the simulation are that scaling limitations can prevent its use at extreme scale and that time series analysis and visualization can be difficult.

A middle path between in situ and offline processing was investigated in PreDatA [13]. This work demonstrates that the placement of the analytics can significantly affect the performance of workflows, and that this placement can be determined in part by the communication characteristics of the analytics components.

In the Big Data community, there exist several widely-used frameworks for the analysis of large data sets, and these can possibly operate on in-memory data and also be chained together over several analytical steps. However, these tools are largely based on the MapReduce model of data analysis, and the types of operations performed in scientific analyses generally cannot be well defined over a key-value view of the data and do not lend themselves well to this model.

Companion work to SmartBlock within this same project is Bredala [14]. This work presents an attempt to build a data model for in situ workflows. It has some similarity to FFS [15]. Unlike FFS, which is part of a much more complex infrastructure for typed messaging between distributed processes, Bredala strictly focuses on a data model that can preserve semantic integrity across redistributions.

Overall, while all of these efforts are addressing different portions of the online workflows puzzle, none of them address the idea of general, reusable data transformation and analysis components for the assembly of entire in situ HPC workflows out of the box.

III. Design

A. Design Goals

SmartBlock components are designed with the general goal of allowing for the assembly of a wide variety of in situ workflows. Below are some general guidelines for the design of such components which both guided the design of the existing SmartBlock components and which we also gained in retrospect.

1) To allow for the greatest variety of workflows, data manipulation primitives and data analysis components should be packaged in similar ways – that is, regardless of their individual complexity, the pieces that make up these workflows should export compatible interfaces as much as possible.

2) The ability to handle multi-dimensional data, along with the consistent labeling of dimensions and quantities as metadata, allows for components that are highly adaptable and simple to use. By designing components that can operate, as much as possible, on data having any number of dimensions, we can allow for a large variety of arrangements of components.

3) While different types of components understand varying levels of semantics, maintaining a high level of semantics (i.e., labeling quantities and dimensions as much as possible) early on and when passing through components that do not

necessarily require all of these labels allows for the most functionality downstream.

4) Because programming languages understand multi-dimensional data as being in a specific order in memory, there is a need for components that re-arrange data and re-label its dimensions without necessarily changing its size. Indeed, when data is stored in a database on disk, it is simple to gain a desired view of the data, for example by using SQL. However, in the middle of a real-time workflow, data must be presented to the components in a format that they expect and understand. This requires a specific ordering of data in memory. We expand on this topic in the description of the Dim-Reduce component later in this section.

These insights guide the design for the reusable workflow components presented in this paper. From a general perspective, designing a smaller number of components to assemble workflows with finer step decomposition allows for more general processing than designing more numerous components each having more complex functionality. And, as we show in §V, componentizing the analysis involves only minor changes in overall workflow performance.

B. Components Overview

SMARTBLOCK currently consists of four generic components that perform common operations in scientific workflows. These are Select, Magnitude, Dim-Reduce, and Histogram. Again, the choice of these components is not meant to encompass as wide a variety of workflow purposes as possible as-is. However, that we were able to deploy three very different workflows driven by different scientific codes using the existing SMARTBLOCK components demonstrates a promising approach.

Each component is a single MPI executable that uses the self-describing property of data exchanged through ADIOS to discover the dimensions and their sizes of the data it receives from its upstream component; in this way, the component is able to automatically partition the generally large dataset that it receives among its constituent processes. The component in question then operates on the data in a way that is specified by the user through command-line parameters. By specifying the names of the streams and the arrays that hold the data of interest through additional parameters, the user is able to specify an entire workflow as a series of applications launched together in a single job script. This procedure is detailed in §V.

Finally, the components are designed based on the assumption that the driving simulation outputs data at regular time steps. When it is done processing the data belonging to a particular time step for a stream, a component can request the data in the next time step. With FlexPath, this works as follows: readers block until the corresponding writers are ready to output the data; similarly, until readers are able to request the data for the next time step, corresponding writers store the data for this step in an internal queue.

C. Select

Given an input stream that includes an array with any number of dimensions, the Select component extracts certain rows (indices) from one of the dimensions. Thus, it outputs an array with the same number of dimensions, but with the dimension of interest having a smaller size. In order to select the quantities of interest, the component uses a header which must be passed by the previous component in the workflow as part of the meta-data that ADIOS can include in the stream. The header is a list of strings that name the quantities in the dimension of interest. This allows the user to specify the names of the quantities (rows) to select by name, rather than by index number, which is easier to do when preparing the launch script.

```
    aprun select input-stream-name input-array-name
        dimension-index output-stream-name
        output-array-name [arg1] [arg2] ...
```

Fig. 1. Select Component Usage

```
    aprun histogram input-stream-name
        input-array-name num-bins
```

Fig. 2. Histogram Component Usage

Figure 1 illustrates the usage of the Select component. The input and output stream and array names identify the stream and dataset on which Select operates, as well as how the component renames them in its output. All components use similar parameters; this lets the user assemble a workflow by using the same stream and array names for the output of an upstream component as for the input of its downstream component.

The parameter `dimension-index` identifies the dimension in which Select will filter certain rows. ADIOS maintains the number of dimensions as meta-data in the stream, and one can identify them by number. Finally, the `arg1` ... parameters identify the names of the rows to keep in the dimension of interest. Select is able to identify these rows by name using the header described previously.

D. Magnitude

The Magnitude component calculates the magnitudes of an array of vectors. That is, it operates on a two-dimensional array, where one dimension spans the data points, for example the particles that make up a simulation, and the other dimension spans any number of components of the same vector for each data point, for example the three-dimensional components of velocity, as we use it in one workflow we demonstrate later.

The output of Magnitude is a one-dimensional array holding the magnitudes of the input vectors. This SMARTBLOCK component only takes the names of the input and output streams as command-line parameters, since it always operates on a two-dimensional array.

E. Histogram

The processes that make up the Histogram component partition among themselves a one-dimensional array of data. They communicate to discover the global minimum and maximum values in the array, create a number of bins between these two extremes, and then communicate again to count the number

```
aprun dim−reduce input−stream−name
      input−array−name dim−to−remove dim−to−grow
      output−stream−name output−array−name
```

Fig. 3. Dim-Reduce Component Usage

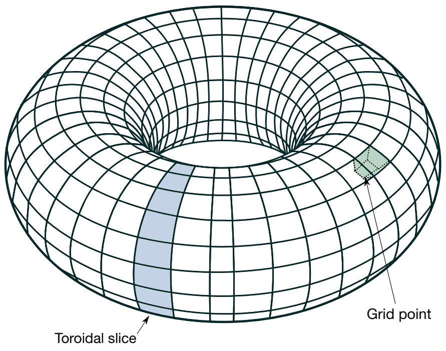

Fig. 4. Representation of GTCP Toroid, modification of [16]

of values in the globally partitioned array that fall in each bin. The number of bins to use must be passed to the component when it is launched.

In our current implementation, one of the processes of Histogram writes the output to a file on disk. We chose this approach because this component is often used as an endpoint in the workflow and because the output of this component is generally very small compared to its input and can be easily written by a single process.

Figure 2 illustrate the usage of the Histogram component. Because it is currently implemented as an endpoint component, Histogram only requires the names of the *input* stream and array names. In addition, the user specifies the desired number of bins.

F. Dim-Reduce

Dim-Reduce is a component that removes one dimension from its input array, "absorbing" it into another dimension without modifying the total size of the data.

This operation is necessary because certain analytical components in workflows expect data having a particular number of dimensions. Due to the fact that multi-dimensional data is arranged in a particular order in memory, this operation can require a re-arrangement of the data in the overall linear representation of the multi-dimensional array.

To better explain the need for Dim-Reduce, we turn to one workflow which we explain in more detail later. GTCP is a toroidal plasma simulator, which breaks up the plasma into toroidal slices and grid points, each having a number of associated physical quantities that describe it (such as the pressure inside it), as illustrated in Figure 4.

In the workflow that is driven by this simulation, we wish to obtain as a final result a histogram of all the pressures at all grid points in the entire toroid. However, the Histogram component expects one-dimensional data. As it is output from the GTCP simulation, the data is three-dimensional, with one dimension spanning the *toroidal slices* of the toroid, another dimension spanning the *grid points* inside of each such slice, and the last dimension spanning the 7 quantities that describe various *properties* of the plasma in each grid point.

To turn this three-dimensional data into a format that Histogram can operate on, the data must pass through two instances of Dim-Reduce. We describe the workflows in greater detail in §V.

IV. Implementation

Each component is an MPI executable written in C/C++, varying in length from 191 lines of code (Histogram) to 459 (Select). Their code is publicly available in [**?**]. The processes that make up a single component belong to the same MPI communicator once the component is launched. For each timestep, these processes communicate to determine how to partition the overall incoming dataset so that each process receives an approximately equal amount of data.

FlexPath allows for the exchange of data between different communicators living in different MPI executables, and this functionality is presented to the components through the ADIOS interface. Other implementation paths are possible here, requiring mainly a common communication mechanism and a typed payload. For example, the HDF5 Virtual Object Layer [17] and Mercury [18] + Boost serialization could be used.

ADIOS allows each process involved in the read operation to specify a bounding box for the multi-dimensional array portion it will receive. FlexPath carries out the actual MxN exchange of data, ensuring each reading process receives all the data that it specifes in its bounding box, even if this portion is itself partitioned among several writers.

More generally, FlexPath implements a publish/subscribe, asynchronous, *stream-based* data exchange abstracted to the components through the ADIOS interface. This facilitates the assembly of SmartBlock workflows in a number of ways:

1. Specifying the input and output stream names as command-line parameters to the SmartBlock components when they are launched allows the user to connect any number of components into a full workflow.

2. Workflow components can be launched in any order: downstream components will wait for data from upstream components and upstream components will buffer data up to a certain size until they are able to send it downstream.

3. Even if the number of processes used for one component is different from that used for the previous one in the workflow, each component can split the data (and therefore the computation) evenly among its processes.

4. A FlexPath stream is implemented as writer side internal data buffering until readers are ready to request the data, at which point a separate writer side thread carries out the data exchange. This asynchronous model allows a SmartBlock component to move on to the computation of a subsequent timestep when a downstream component is not yet ready to accept data, effectively overlapping computation and IO and offering high performance to a componentized workflow.

There is no need to re-compile SmartBlock components when using them in different workflows. Any configuration of a component for a particular workflow can be provided through run-time arguments.

To enable a scientific code to drive a workflow using our SmartBlock components, one needs to modify its output routines to use ADIOS. Specifically, ADIOS expects multi-dimensional arrays to be packed linearly, with the variables describing the dimensions specified in an XML configuration file that is read by ADIOS at run time. Roughly 70 lines of code were required to allow each of the three simulations in our evaluation, namely LAMMPS, GTCP, and GROMACS, to work with SmartBlock, along with an approximately 25-line XML file. This small, one-time modification to a simulation allows it to work with any SmartBlock workflow.

V. Evaluation

We designed and implemented three realistic in situ workflows based on scientific codes having large user bases: the LAMMPS Newtonian particle simulator [19], GTCP, a particle-in-cell Tokamak simulator [20], and GROMACS, a biomolecular dynamics code [21]. These are illustrated in Figure 5, Figure 6, and Figure 7.

The primary purpose of our evaluation is to demonstrate the successful assembly and deployment of SmartBlock workflows. In doing so, we also present useful weak and strong scaling properties exhibited by SmartBlock components during these runs, as well as a measurement-based validation of the componentized method to building workflows, which is inherent in the SmartBlock approach.

The large-scale evaluation is performed on Titan, the Cray XK7 machine at Oak Ridge National Laboratory. It consists of 18,688 nodes each with 1 16-core AMD Opteron CPU and 32 GB of RAM. The interconnect is a Gemini network.

The smaller-scale results are obtained from the Falcon cluster on the Georgia Tech campus; it is an 80-node cluster of Intel Xeon X5660 machines, with 12 cores and 24 GB of RAM per node.

A. Workflows

The three SmartBlock workflows used in this evaluation perform different kinds of runtime data analysis overall. However, there are similarities between them. Each driving scientific code is a simulation that operates on a large number of units of equal size — namely, atoms in the case of LAMMPS and GROMACS, and grid points in the case of GTCP (see Figure 4). Each simulation operates over these units with fine-grained time step granularity and outputs the states of these units at coarse-grained intervals. From this point on, we refer to these larger I/O intervals as "timesteps." These *states* correspond to various properties of these units, such as the positions of individual atoms or the temperatures of individual grid points.

All three SmartBlock workflows based on these simulations result in a per-timestep histogram that describes the spread of a certain quantity of interest over these units. It is often the case for in situ workflows that the resulting data set has a much smaller size than the raw data set output by the simulation. A histogram illustrates one such final step in a workflow that presents a human-readable reduction of data, and this explains our choice for Histogram as one of the SmartBlock components implemented for this work.

Still, how the workflows arrive at their results varies significantly. Histogram expects one-dimensional input data, and our choices for Select, Dim-Reduce, and Magnitude illustrate examples of various generic operations that tranform different data sets into a format that the allows one final, simple component such as Histogram to operate on.

All components of SmartBlock workflows, including the simulation, are launched simultaneously using a script such as that shown in Figure 8. The asynchronous property of FlexPath allows readers to block until the corresponding writers are ready to send their data, and vice versa. Providing the names of the input and output streams lets the user connect any number of components in any order. For example, the launch script tells Magnitude to look for an array named `lmpsel` in a FlexPath stream named `lmpselect.fp`, which are the names used to specify the output stream and array of Select. Thus, this script specifies that Magnitude is immediately downstream from Select in this workflow. Notice, also, the decreasing numbers of processes used in each component.

We configure LAMMPS to simulate a disruption (a "crack") in a thin layer of particles and output 5 numerical properties describing each particle in the simulation at regular timestep intervals. This corresponds to two-dimensional data (particles as one dimension and properties of interest as another) and among these properties are the three-dimensional components of the particles' velocities. Select filters out the ID and Type of the particles, keeping the velocities. Magnitude computes the magnitudes of these velocity vectors, outputting a one-dimension array of these magnitudes. Histogram outputs a human-readable distribution of the velocity magnitudes of all particles involved in this simulation.

GTCP, a code that simulates a toroidally confined plasma, splits the solid into toroidal slices, each made up of a number of grid points. For each of these grid points, it outputs 7 properties of the plasma such as pressure and energy flux. This division of the toroid is illustrated in Figure 4. The output of the simulation is therefore a three-dimensional array in which the dimensions span: (a) toroidal ranks (toroidal slice number), (b) grid point numbers, and (c) various properties that describe each grid point. Of these 7 properties, Select filters out all but the pressure in each gridpoint. However, the output of Select is still three-dimensional, and the data must go through two instances of Dim-Reduce to allow Histogram to operate on it, showing in the end a distribution of the pressures in the entire toroid.

Among other quantities, GROMACS outputs the three-dimensional coordinates of the atoms involved in the simulation at regular intervals. The data array itself is two-dimensional: 3D coordinates over all atoms. From these, we obtain a histogram of the distances of the atoms from the origin for each timestep, showing an evolution of the spread of the particles throughout the simulation.

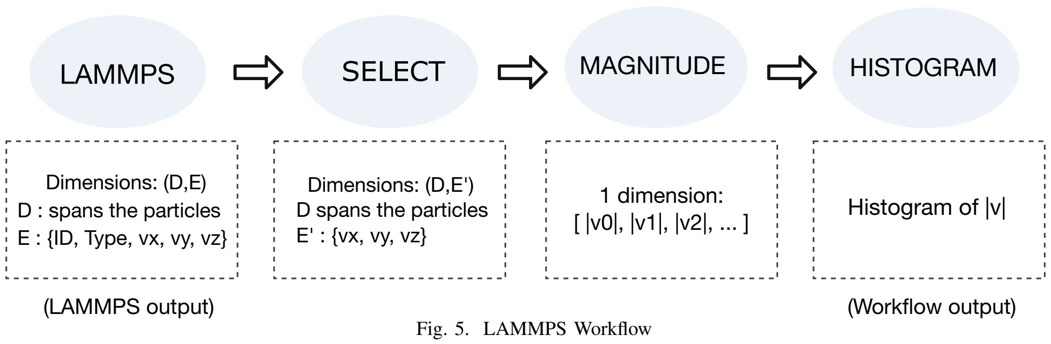

Fig. 5. LAMMPS Workflow

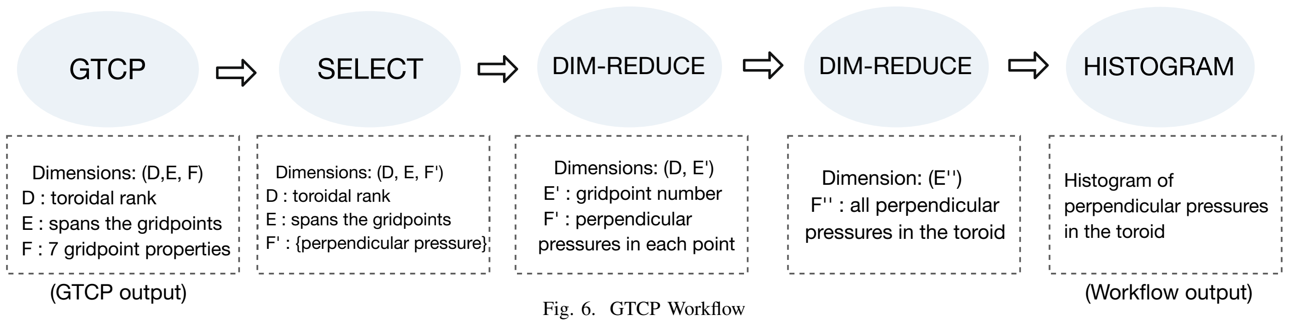

Fig. 6. GTCP Workflow

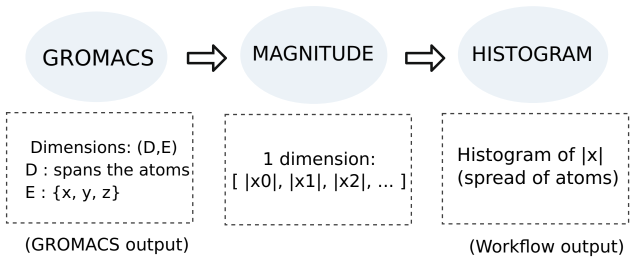

Fig. 7. GROMACS Workflow

```
aprun −n 64 histogram velos.fp 16 velocities &
aprun −n 256 magnitude lmpselect.fp lmpsel
      velos.fp velocities &
aprun −n 256 select dump.custom.fp atoms 1
      lmpselect.fp lmpsel vx vy vz &
aprun −n 1024 lammps < in.cracksm &
wait
```

Fig. 8. SMARTBLOCK example launch script, LAMMPS workflow

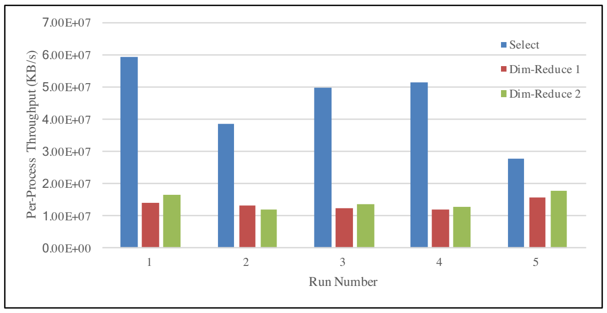

Fig. 9. GTCP workflow weak scaling experiment: Per-component, per-process throughputs in KB/s.

B. *Weak Scaling*

Different workflows have different requirements in terms of the data set sizes that they operate on and therefore on the scales of their components. Thus, in order to use *generic* components, it is important that the performance of the individual components as well as the end-to-end performance of the resulting workflow be reasonably predictable. To test this, we ran a weak scaling experiment whose setup is described in Table I. The table lists the process sizes and data set sizes used in the 5 runs of the GTCP workflow for this experiment. We measured the end-to-end times of these runs, as well as the completion time of individual timesteps, broken down by component, averaged over the component's communicator.

These measurements allow us to extract an approximate per-process throughput, both for the entire workflow, end-to-end, as well as on a per-component basis. The last column in Table I lists these end-to-end results, where the total data set size produced by the simulation is divided by the total number of processes used in the workflow and by the total time taken by the workflow. Figure 9 shows similar per-component, per-process throughputs for three of the components used in these runs, for a timestep taken arbitrarily in the workflow. We can see that while there is a certain variation in these throughputs across the runs, especially at the largest scale, where communication overhead is most significant, both SMARTBLOCK workflows and their individual components exhibit very good weak scaling properties, with a maximum throughput decrease of about 57% between the two extremes in scale.

C. *Comparing Generic and Ad Hoc Approaches*

It is expected that assembling a workflow using generic components involves the use of finer-grain components than

TABLE I
GTCP-SMARTBLOCK: WEAK SCALING EXPERIMENT SETUP, END-TO-END RESULTS

Run	GTCP Output (MB)	GTCP Procs	Select Procs	Dim-Red1 Procs	Dim-Red2 Procs	Histo Procs	End2End Time (s)	Throughput (KB/s)
1	918.3	64	10	6	6	2	92.72	112,541
2	1434.6	84	16	10	10	2	115.23	102,046
3	2065.6	156	18	14	14	4	97.27	103,089
4	2811.3	234	25	19	19	5	96.36	96,605
5	12905.4	1024	116	88	88	24	197.66	48,724

TABLE II
LAMMPS: SMARTBLOCK VS. ALL-IN-ONE COMPARISON

SIM output	AIO time (sec)	SMARTBLOCK time (sec)	LMP only (sec)
20 MB	115.26	116.51	115.03
80 MB	148.70	149.80	146.97
320 MB	154.65	157.65	153.69
1280 MB	155.32	157.98	152.48
5120 MB	167.39	168.79	165.22

when using ad-hoc analytical routines specifically coded for a workflow of interest. One problem that might be anticipated in more componentized workflows is a decrease in overall performance caused by (a) more stages that require the coordination of readers and writers and (b) more points of actual data transfer.

To investigate this potential problem, we wrote a custom, all-in-one (AIO) component that performs the same analytical procedure as all the components involved in the LAMMPS workflow outside of the simulation itself. We measured the start-to-end completion times of the two workflows at different scales.

Table II shows the start-to-end completion times at increasing scales, of (a) LAMMPS with the AIO component in the second column (b) LAMMPS with the full SMARTBLOCK workflow in the 3rd column and (c) the LAMMPS simulation only with the output routines removed from the code in the last column. The measurements in the last column are meant to give an idea of the portion of the workflow completion time taken up by the simulation computation only.

A weak scaling approach is used, with approximately the same per-process data size throughout the scaling. The time is measured from the start of the simulation to the point when the last histogram of the last timestep is written. For each SMARTBLOCK workflow run, the corresponding AIO workflow run allocates the same number of processes to the AIO component as the SMARTBLOCK workflow allocates to the Select component. In the SMARTBLOCK workflow, additional processes are allocated to the other two components (Magnitude and Histogram).

The results show only a small increase in workflow completion time for the SMARTBLOCK workflows (with a maximum increase of 1.9%). The componentized approach involves more data exchange points in the workflow, thus leading to overhead from increased MxN coordination and data exchange. However, the overlap of computation and I/O provided by FlexPath amortizes this overhead.

Granted, slightly more resources are allocated to the SMARTBLOCK workflow runs to allow the small number of final components to execute (1280 processes used in total for

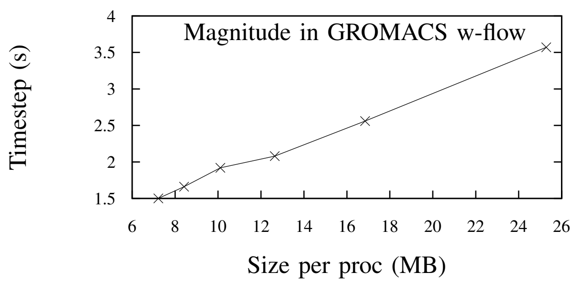

Fig. 10. Magnitude strong scaling in the GROMACS workflow.

the AIO workflow vs. 1600 for the SMARTBLOCK workflow at the largest scale). Also, much of the start-to-end time is spent on the simulation's computation. However, the measurements in the last column of Table II only give an idea of the proportion of overall time occupied by the simulation only, since when workflows are running, there is much overlap in the computation and I/O between the simulation and other components. All in all, these results show that the fine-grained component approach to assembling workflows is reasonable from a performance standpoint.

D. Strong Scaling

To determine appropriate resources to use in workflow runs, we run strong scaling experiments such as presented in Figure 10, where only one component's process size varies. What we generally observe is expected: a linear domain of scalability, followed by a turning point and eventual flattening of the scaling curve. Figure 10 shows the linear domain of the timestep completion time of the Magnitude component in the GROMACS workflow; the process size of Magnitude varies, with more processes used in the lower end of the chart, and the process sizes of GROMACS and Histogram are kept the same. Such experiments allow users to better determine how to allocate resources to SMARTBLOCK workflows. While limited space only allows us to present this sample result, numerous results we have obtained from other components and workflows show similar strong scaling characteristics.

VI. CONCLUSIONS AND FUTURE WORK

This paper presents SMARTBLOCK, a demonstration of making generic, reusable components for scientific workflows. By using a stream-based pipeline and decomposing the operations into small chunks, we achieve components that can be reused, without modification, for a variety of different workflows. These components handle data having any number of dimensions and operate on various streams and arrays passed to them at run time. Maintaining high level semantics upstream, by labeling dimensions and certain quantities inside of these dimensions, gives good data understanding to downstream components.

Through the demonstration of generating a velocity histogram for LAMMPS, a pressure histogram for GTCP, and a distribution of the spread of the atoms for a GROMACS experiment, we demonstrate reusing the same components over different data formats and application types.

While this work leverages ADIOS and the FlexPath transport, this is not the only approach for addressing reusable in situ components. Other, similar approaches can also work. However, in this case, the data annotation provided by this infrastructure simplifies creating reusable components.

The components presented here are limited to in situ workflows with all components running simultaneously. However, introducing new components that write and read from storage as part of a workflow can break that dependency and are a superficially simple addition. Future work will investigate these challanges.

Other future work involves expanding the generic components library to include a variety of other analytical operations. In particular, the SMARTBLOCK components presented in this paper result in an output dataset having either the same size or a smaller size as the input. Analytical procedures that lead to an increase in data size, such as all-pairs calculations, are common and can be implemented using the SMARTBLOCK approach.

To enrich SMARTBLOCK into a true Workflow Management System, we hope to leverage ADIOS' ability to have several "write groups" so as to allow for the development of a *Fork* component that would permit the creation of much richer workflows described by directed acyclic graphs, such as those used to evaluate [22]. And, to manage the execution of workflows over longer periods of time, we plan on investigating the incorporation of SMARTBLOCK into higher-level workflow management systems such as Kepler and DAGMan.

ACKNOWLEDGMENTS

The authors would like to thank Matthieu Dreher and Tom Peterka from Argonne National Lab for their valuable assistance with GROMACS and Pegasus.

This work was supported by the U.S. Department of Energy, under FWP 15-017577 and DE-SC0016313, program manager Lucy Nowell.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] J. Lofstead, F. Zheng *et al.*, "Adaptable, metadata rich IO methods for portable high performance IO," in *Proceedings of the International Parallel and Distributed Processing Symposium*, Rome, Italy, 2009.

[2] J. Dayal, D. Bratcher *et al.*, "Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics," in *Cluster, Cloud, and Grid*, ser. CCGrid '14. IEEE, 2014.

[3] McKesson, "Formfast," 2016, http://formfast.com/platform/integrate/ehr-integration/mckesson/.

[4] wfmc, "Workflow management coalition," 2016, http://wwww.wfmc.org/.

[5] S. J. Mullender, I. M. Leslie, and D. McAuley, "Operating-system support for distributed multimedia," in *Proceedings of the 1994 Summer USENIX Technical Conference*, 1994, pp. 209–219.

[6] G. Malewicz, I. Foster *et al.*, "A tool for prioritizing DAGMan jobs and its evaluation," *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pp. 156–168, 0-0 2006.

[7] M. Wilde, M. Hategan *et al.*, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[8] B. Ludäscher, I. Altintas *et al.*, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[9] H. Karimabadi, B. Loring *et al.*, "In-situ visualization for global hybrid simulations," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*. ACM, 2013, p. 57.

[10] K. Moreland, D. Lepage *et al.*, "Remote rendering for ultrascale data," *Journal of Physics: Conference Series*, vol. 125, no. 1, p. 012096, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012096

[11] B. Whitlock, J. M. Favre, and J. S. Meredith, "Parallel in situ coupling of simulation with a fully featured visualization system," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, ser. EGPGV '11. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2011, pp. 101–109. [Online]. Available: http://dx.doi.org/10.2312/EGPGV/EGPGV11/101-109

[12] M. Riedel, T. Eickermann *et al.*, "Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures," in *e-Science and Grid Computing, IEEE International Conference on*, dec. 2007, pp. 483 –490.

[13] F. Zheng, H. Abbasi *et al.*, "PreDatA - preparatory data analytics on Peta-Scale machines," in *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.

[14] M. Dreher and T. Peterka, "Bredala: Semantic data redistribution for in situ applications," in *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Ieee, 2016.

[15] G. Eisenhauer, M. Wolf *et al.*, "A type system for high performance communication and computation," in *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*. IEEE, 2011, pp. 183–190.

[16] Yassine Mrabet, "Simple torus," https://commons.wikimedia.org/wiki/File:Simple_Torus.svg, 2007, accessed: April 27, 2016.

[17] M. Folk, G. Heber *et al.*, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM, 2011, pp. 36–47.

[18] J. Soumagne, D. Kimpe *et al.*, "Mercury: Enabling remote procedure call for high-performance computing." in *CLUSTER*. IEEE, 2013, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/cluster/cluster2013.html#SoumagneKZCKAR13

[19] S. Plimpton, R. Pollock, and M. Stevens, "Particle-mesh ewald and rrespa for parallel molecular dynamics simulations," in *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997, March 14-17, 1997, Hyatt Regency Minneapolis on Nicollel Mall Hotel, Minneapolis, Minnesota, USA*. SIAM, 1997.

[20] Z. Lin, T. S. Hahm *et al.*, "Turbulent transport reduction by zonal flows: Massively parallel simulations," *Science*, vol. 281, no. 5384, pp. 1835–1837, September 1998.

[21] B. Hess, C. Kutzner *et al.*, "Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation," *Journal of chemical theory and computation*, vol. 4, no. 3, pp. 435–447, 2008.

[22] H. Sim, Y. Kim *et al.*, "Analyzethis: an analysis workflow-aware storage system," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 20.