# Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform

Fan Zhang, Ciprian Docan, Manish Parashar
Center for Autonomic Computing
Rutgers University, Piscataway NJ, USA
{zhangfan,docan,parashar}@cac.rutgers.edu

Scott Klasky, Norbert Podhorszki, Hasan Abbasi
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN, 37831, USA
{klasky,pnorbert,habbasi}@ornl.gov

***Abstract*—Emerging scientific application workflows are composed of heterogeneous coupled component applications that simulate different aspects of the physical phenomena being modeled, and that interact and exchange significant volumes of data at runtime. With the increasing performance gap between on-chip data sharing and off-chip data transfers in current systems based on multicore processors, moving large volumes of data using communication network fabric can significantly impact performance. As a result, minimizing the amount of inter-application data exchanges that are across compute nodes and use the network is critical to achieving overall application performance and system efficiency. In this paper, we investigate the in-situ execution of the coupled components of a scientific application workflow so as to maximize on-chip exchange of data. Specifically, we present a distributed data sharing and task execution framework that (1) employs data-centric task placement to map computations from the coupled applications onto processor cores so that a large portion of the data exchanges can be performed using the intra-node shared memory, (2) provides a shared space programming abstraction that supplements existing parallel programming models (e.g., message passing) with specialized one-sided asynchronous data access operators and can be used to express coordination and data exchanges between the coupled components. We also present the implementation of the framework and its experimental evaluation on the Jaguar Cray XT5 at Oak Ridge National Laboratory.**

***Keywords*-coupled simulations, data-intensive application workflows, data-centric task mapping, in-situ application execution**

## I. Introduction

High-performance computing (HPC) systems are increasingly being based on processor architectures that rely on large core counts and deeper memory hierarchies to achieve better performance and efficiency. For example, the current Jaguar system is composed of 12-core AMD Opteron processors, the upcoming Titan system will be based on 16-core AMD Opteron processors, and the forthcoming petascale supercomputers Mira and Sequoia will both employ the IBM Blue Gene/Q 18-core processor to exploit more on-chip parallelism. This architectural trend is increasing the performance gap between on-chip data sharing and off-chip data transfers, and moving large volumes of data using communication network fabric can significantly impact performance. As a result, minimizing the amount of data exchanges that are across compute nodes and use the network is critical to achieving overall application performance and system efficiency.

Meanwhile emerging scientific application workflows that target at high-end computing platforms are composed of heterogeneous coupled component applications that simulate different aspects of the physical phenomena being modeled, and that interact and exchange significant volumes of data at runtime. For example, in the Community Earth System Model (CESM) [1] application workflow, separate parallel applications are coupled as part of a multiphysics model that simulates the interaction of the earth's ocean, atmosphere, land surface and sea ice. Similarly, the coupled fusion simulation workflows [2] developed by the Fusion Simulation Project are composed of codes modeling kinetic pedestal buildup (XGC0), magnetic equilibrium reconstruction (M3D_OMP), linear stability-boundary check (Elite), nonlinear ELM crash (M3D_MPP), and diverter heat-load evaluation (XGC0). Furthermore, application workflows are increasingly being composed of end-to-end I/O pipelines, such as those enabled by the Adaptive IO System (ADIOS) [3], which extract and stream data being produced by the simulations to data staging nodes where parallel data analysis and/or transformation operations (e.g., redistribution, interpolation, reduction) are executed asynchronously and concurrently. The costs of moving the increasingly large volume of data associated with these interactions and couplings has become a dominant part of the overall application executions times and costs.

Clearly, in order to effectively utilize the potential of current and emerging HPC systems it is essential that such coupled data-intensive scientific workflows exploit data locality and core-level parallelism to the extent possible. However, achieving this can often be non-trivial and involves several challenging issues: (1) *Locality-aware mapping of tasks from separate coupled applications onto processor cores.* While existing research (e.g., [4], [5]) has focused on mapping frequently communicating computation tasks (i.e., MPI processes) within a single parallel application onto processing elements (PEs) that are physically "close", mapping computation tasks from multiple separate applications that part of a tightly coupled simulation workflow presents new challenge. (2) *Efficient support for coordination and data exchange between the coupled applications*. The coordination and data exchange patterns in application workflows can vary depending upon the type of data decomposition of the coupled application, the number of tasks in each application, the nature of the coupling, etc. For

example, code coupling typically requires data redistribution, i.e., the M×N problem where data from an application running on M processes is coupled with another application running on N processes [6]. Existing solutions to this M×N coupling problem have involved approaches such as using separate data coupling server or creating a single MPI meta-application. Appropriate abstractions and underlying mechanisms that are flexible, efficient and scalable, and exploit on-chip communications to the extent possible are required to support the inter-application coordinations and data exchanges.

In this paper, we investigate the in-situ execution of the coupled component applications of a scientific workflow, so as to maximize intra-node exchange of data. Specifically, we present a distributed data sharing and task execution framework that (1) employs data-centric task placement to map computations from the coupled applications onto processor cores so that a large portion of the data exchanges can be performed using the on-processor shared memory, (2) provides a shared space programming abstraction that supplements existing parallel programming model (e.g., message passing) with specialized one-sided asynchronous data access operators, and can be used to express coordination and data exchanges between the coupled applications and end-to-end applications workflows. The framework builds a scalable, semantically specialized virtual shared space that is distributed across processor cores on compute nodes of the HPC system, and provides simple abstractions for coordination, interaction and data-exchange.

We have implemented the data sharing and task execution framework on the Jaguar Cray XT5 system at Oak Ridge National Laboratory. The framework currently supports execution and data-centric task mapping for workflow that composed of data parallel applications with regular multidimensional data meshes and domain decompositions.

The rest of the paper is structured as follows. Section II describes the problem addressed and presents two motivating application workflow scenarios. Section III describes the architecture of the presented framework. It also describes the programming abstractions provided for supporting coupled application workflows. Section IV describe the implementation of the framework. Section V presents an experimental evaluation of the framework using sample workflows on the Jaguar Cray XT5 system at ORNL. Section VI presents related work. Section VII concludes the paper and outlines future research directions.

## II. Background

### A. Motivating Application Scenarios

This research is motivated by two application scenarios that are becoming increasingly important at the peta- and exascales:

**End-to-end application workflows:** Traditionally, scientific data analysis and visualization are performed offline as a post-processing step. For example, simulations write data to a file system, which is then read by analysis and visualization codes. However, given the increasing scale of long-running simulations and the costs associated with IO, end-to-end application workflows that integrate simulations with online data analysis and visualization are more attractive.

**Coupled multi-phase, multi-physics simulations:** Large-scale simulations model complex phenomena that involves multiple physics, phases, scales that are coupled together. For example, the CESM coupled climate modeling system has different parallel geophysical component models such as atmosphere, land and sea-ice. These models are tightly coupled and frequently exchange boundary data consisting of a large number of data fields. Furthermore, the order of execution of the models and coordination between them is defined by the science. For example, in a typical CESM configuration, during each simulation step, the land and sea-ice components run concurrently, and run after the atmosphere model has completed.

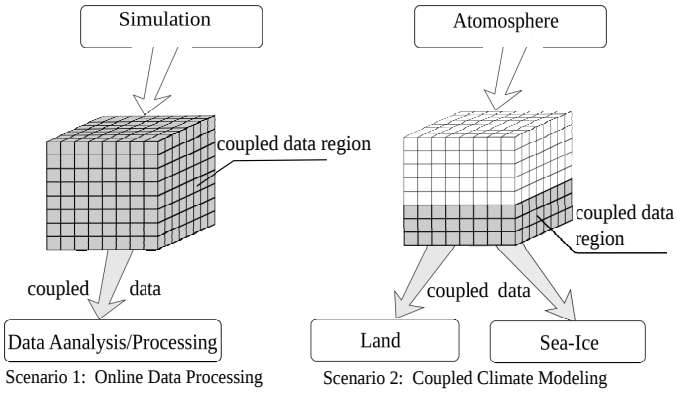


Fig. 1. Examples of coupled scientific workflow

Both motivating application scenarios described above can be represented as scientific workflows composed of interacting component applications, that are parallelized by decomposing and distributing their data domains across a set of computation tasks (i.e., processes in a MPI program). Furthermore, the coupling between these applications can be defined in terms of overlaps in their domains, i.e., applications regularly exchange data that associated with the overlapped regions. This is illustrated in Figure 1. The coupled data region (the shaded area in the figure) is the entire data domain in the end-to-end application workflow (i.e., online data processing) scenario, and is the interface region between the component models in case of the coupled multi-physics simulation (i.e., coupled climate modeling) scenario.

### B. In-situ Execution of Coupled Scientific Workflow

It is clear from Figure 1 and the discussion above that reducing the overheads of data movement during the interactions between the component applications can improve performance and efficiency. However, existing frameworks typically run the different applications of these tightly-coupled data-intensive workflows on separate sets of compute nodes, which results in a large amount of inter-application data movement over the communication network fabric impacting both, performance and costs.

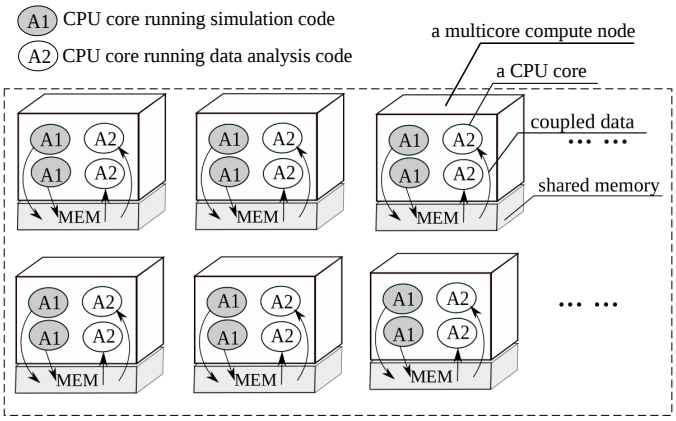


Fig. 2. In-situ execution of online data processing workflow

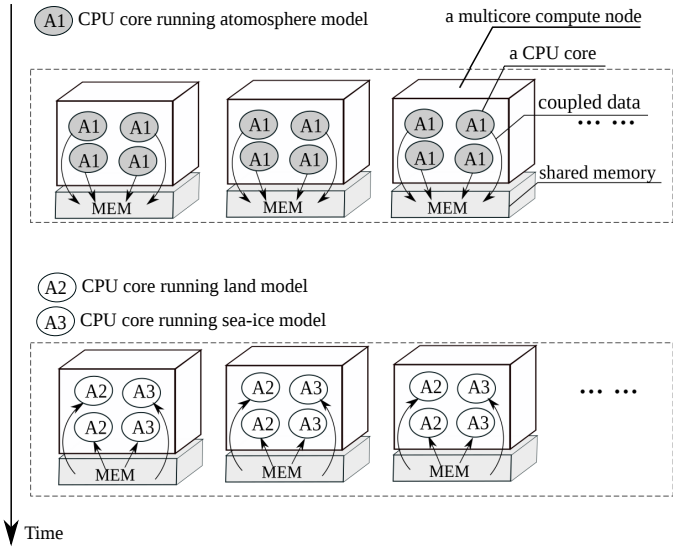


Fig. 3. In-situ execution of coupled climate modeling workflow

The expensive network-based data movement can be reduced through in-situ execution of the coupled application so as to maximize on-node data locality. The key idea of in-situ execution is a data-centric mapping of computation task associated with the coupled component applications onto processor cores so that a large portion of the data exchanges can be performed using the on-node shared memory. This is illustrated in Figures 2 and 3 for the two application scenarios. In Figure 2, the simulation and analysis components of the end-to-end application workflow run concurrently on the same compute nodes, but on different cores. As a result, intra-node data transfers between the applications can be performed using shared memory. Similarly, as shown in Figure 3, the atmosphere, land and sea-ice components of the coupled climate modeling workflow run on the same set of compute nodes, and data produced by atmosphere model is cached in memory, and then consumed in-situ by subsequent computation tasks of land and sea-ice models.

The rest of this paper focuses on a framework for enabling in-situ execution of coupled scientific workflows as illustrated above. Specifically, it focuses on addressing two key challenges: first, mapping the computation tasks of the workflow component applications onto distributed multicore processors in order to increase on-node locality and the amount of intra-node exchange, and second, abstraction and mechanisms for expressing and implementing efficient parallel data transfers between these applications.

## III. Architecture and Programming Interface

### A. System Architecture

The system architecture of the proposed framework consists of two main components as illustrated in Figure 4, the workflow management server and the execution client. The workflow management server acts as the rendezvous point to bootstrap execution clients and manages the execution of the DAG-based workflow, and the distributed execution clients run the computation tasks of the data parallel applications within the coupled scientific workflow. For example, in case of an MPI application, one MPI process is created per core on a multicore compute node, and each process runs as an execution client. Both components build on the HybridDART communication layer.

The *HybridDART Communication Layer* supports asynchronous data transport between computation tasks running on a multicore based system. It is based on DART [7], which builds on RDMA-enabled networks and provides an RPC-like abstraction and hides the complexities of the underlying communication systems such as buffer management. HybridDART exploits the available shared memory between processor cores on a compute node to achieve better performance, and dynamically select the appropriate data transfer mechanism, i.e., shared memory or RDMA-supported network transport, depending on the locations of the communicating tasks.

The execution clients build a co-located DataSpace (CoDS), which provides a virtual distributed shared-space abstraction that can be associatively accessed by the coupled applications using semantically specialized operators that are based on representation of the scientific application's data domain, for example, a grid or mesh. The CoDS *Data Lookup* service provides data locations to compute communication schedule and the *Data Sharing Service* implements a simple *put()*, *get()* data sharing API that can be used for inter-application coordination and data sharing. These data sharing operations are implemented using HybridDART. Coupled data generated by data producer applications in the sequential coupling scenario is first stored in the CoDS distributed memory space, and then shared with subsequently running data consumer applications. Concurrent coupling scenario directly transports coupled data between the concurrently running producer and consumer applications.

The workflow management server includes two major modules. The *Execution Client Management* module handles the registration/unregistration of the execution clients, and manages information such as the network address for each registered execution client. The *Workflow Engine* manages

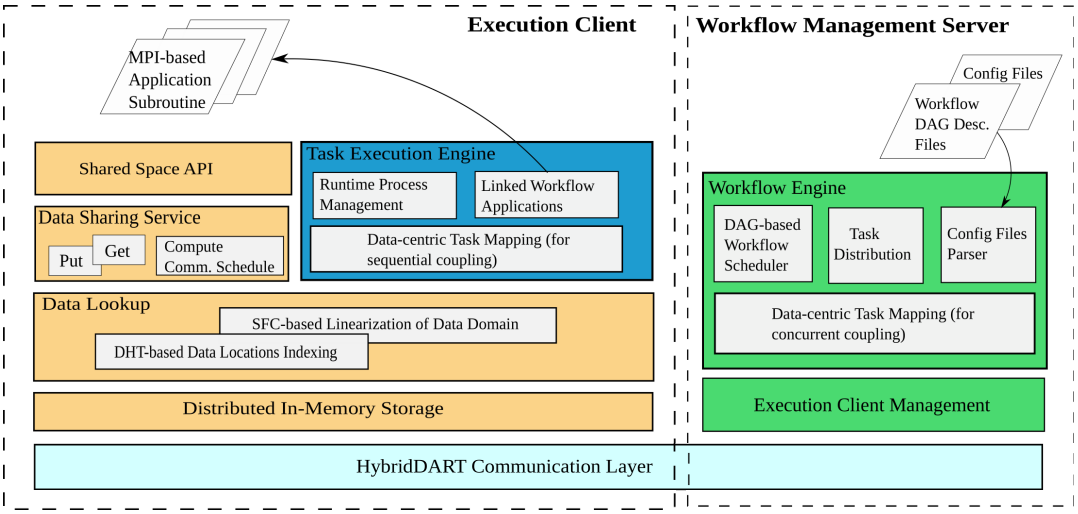


Fig. 4. A schematic overview of the system architecture.

the correct enactment and progress of DAG-based scientific workflows composed of parallel applications. *Workflow Engine* is also responsible for tracking the availability of registered execution clients, their allocation to the parallel component applications, and the initial distribution of computation tasks.

The overall goal of our framework is to enable the in-situ execution of the scientific workflow by using a data-centric and locality-aware task mapping that moves computation tasks closer to the data they require. As shown in Figure 4, the framework employs a combination of server side and client side task mappings strategies to support the two coupling patterns in the motivating application workflow scenarios, i.e., concurrent coupling in the online data processing workflow scenario and sequential coupling in the coupled climate modeling workflow scenario.

The server side mapping strategy is designed to place computation tasks from concurrently coupled applications so that data producer and consumer tasks run closer to each other, i.e., on cores of the same compute node. In this strategy, the framework first computes the inter-application communication graph offline for the workflow components based on the specified data decomposition of the component applications. The workflow management server then uses graph partitioning tools (e.g., METIS [8]) to group and map data-intensive communicating tasks onto the same compute node, in order to reduce the amount of network-based data transfer.

The client side mapping strategy is designed to place computation tasks of the data consumer applications closer to required data. In a sequentially coupled workflow scenario, the coupled applications run in a time sequential manner, so the coupled data generated by data producer applications would have been stored in CoDS when the data consumer applications are launched. This decentralized mapping strategy first distribute and assign a computation task of the data consumer applications to each execution client, then execution client queries the *Data Lookup* service to get locations of data required by the assigned computation task, and dispatch that task to compute node where all or large portion of coupled data can be directly retrieved from local memory.

Once task mapping is complete, the *Tasks Execution Engine* initiates the execution of the computation tasks of an application on the processor cores they are mapped.

### B. Programming Interface

Programming application workflows using our framework consists of three steps: (1) composing the coupled application components into a DAG, (2) exposing the data decomposition used by the applications, and (3) expressing coordination and data sharing between the coupled applications using the CoDS operators. These steps are described below.

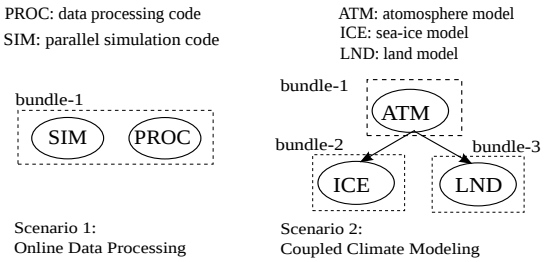


Fig. 5. Examples of workflow DAG representations.

Our framework supports coupled application workflows that can be expressed as a DAG, where each vertex in the DAG represents a parallel application. Our DAG representation extends traditional DAG representation such as DAGMan used in the workflow engine Pegasus [9], with the concept of a "bundle" which represents a group of parallel applications that need to be scheduled simultaneously, for example, concurrently coupled applications that exchange data at runtime. The edges of the DAG represent data dependencies between sequentially couopled applications.

The DAG as well as the bundles are explicitly defined by users. Figure 5 presents the DAG representation for the two

coupled scientific workflow scenarios, online data processing and climate modeling. Users produce a DAG description file for their workflow (see Listing 1), which is then parsed by the management server. Each parallel application in the DAG is identified by a unique *application id* in the description file.

```
# Online Data Processing Workflow
# Simulation code has appid=1
# Processing code has appid=2
# Bundle is specified by IDs of its applications
APP_ID  1
APP_ID  2
BUNDLE  1 2


# Climate Modeling Workflow
# Atmosphere model has appid=1
# Land model has appid=2, Sea-ice model has appid=3
APP_ID  1
APP_ID  2
APP_ID  3
PARENT_APPID 1   CHILD_APPID  2
PARENT_APPID 1   CHILD_APPID  3
BUNDLE  1
BUNDLE  2
BUNDLE  3
```

Listing 1. DAG representation for workflows for the Online Data Processing and Climate Modeling scenarios.

The proposed framework requires two piece of information to perform the data-centric task mapping:

(1) *Data required by a computation task*. The current implementation uses an application's data decomposition to infer the data regions that a computation task needs. As a result, users need to specify the decomposition of the applications data domain. We assume that the application are based on a regular multidimensional data domain and its decomposition can be expressed in terms of a domain size, process layout, data distribution type, and data block size. A *n*-tuple $(s_1, ..., s_n)$ and *n*-tuple $(p_1, ..., p_n)$ is used to specify the size and number of processes in each dimension of the coupled data domain. The framework currently supports three types of data distributions: standard blocked, cyclic and block-cyclic. A *n*-tuple $(b_1, ..., b_n)$ is used to specify the size in each dimension of the data block when block-cyclic distribution type is applied.

(2) *Data locations*. The location of data required by a task can be discovered in two ways. The first is querying the *Data Lookup* service, which keeps track of locations of the data that has been produced and stored in CoDS. The second is using the data decomposition used by the application as specified by users, which indicates how coupled data is to be partitioned among different computation tasks (i.e. processes in a MPI program). The former is used for sequential coupling while the latter is used for concurrent coupling scenarios.

Application can use a simple API (shown in Table I) to share the coupled data. There are two pairs of operators for concurrent and sequential data coupling respectively: *cods_put_con()* and *cods_get_con()* are used to set up direct data transfers between producer and consumer applications for a concurrent coupling scenario. *cods_put_seq()* and *cods_get_seq()* enable asynchronous data sharing using the CoDS distributed in-memory storage space and enable the sequential coupling scenario. These operators require users to specify the region of interest using a simple geometric descriptors, for example, a bounding box (i.e., $< 0, 0, 0; 10, 10, 20 >$). The inter-application data transfers associated with these operators are transparently managed at runtime by the framework.

TABLE I
PROGRAMMING INTERFACE FOR COUPLING.

| | |
|---|---|
| *cods_put_seq()* | Put data (specified by a geometric descriptor) into the virtual shared space. Used in sequential data coupling. |
| *cods_get_seq()* | Get data (specified by a geometric descriptor) from the virtual shared space. Used in sequential data coupling. |
| *cods_put_con()* | Put data (specified by a geometric descriptor) into the virtual shared space. Used in concurrent coupling. |
| *cods_get_con()* | Get data (specified by a geometric descriptor) from the virtual shared space. Used in concurrent coupling. |

## IV. IMPLEMENTATION OVERVIEW

### A. Data Sharing using Co-located DataSpaces (CoDS)

Coupled applications share data using the virtual shared space abstraction provided by the co-located data space. CoDS is based on DataSpaces [10] and essentially constructs a distributed hash table (DHT) that spans cores across all the compute nodes, which keeps track of locations of the coupled data and uses a semantically specialized indexing that is based on the scientific applications' representation of the data domain.

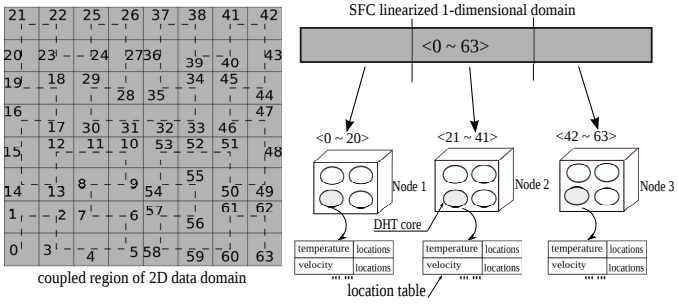


Fig. 6. SFC-based linearization of the application domain and DHT construction.

For example, in case of a Cartesian mesh, the framework applies Hilbert space-filling curve (SFC) to linearize the n-dimensional Cartesian coordinates to generate a 1-dimensional index space, which is then used to construct the DHT. Using this indexing, a continuous data region in the original Cartesian domain can be represented either by a geometric descriptor such as a bounding box, or a set of spans of the linearized index space.

The 1-dimensional index space is divided into intervals, which are assigned to DHT cores (each compute node has one DHT core). As a result, each DHT core is assigned a distinct

data region of the application data domain, and creates a table to record where data associated with that region is located. Figure 6 illustrates how a simple 2D 8×8 data domain is linearized, divided and indexed across the DHT cores. The figure also shows the location table at each core used to maintain the storage locations for shared variables (e.g., *temperature*, *velocity*) defined over the coupled data domain. When a data *put()* operation is invoked, the execution client translates the user-defined geometric descriptor into a linearized DHT query key, and routes the query to the appropriate DHT cores that are responsible for data regions specified by the key. The corresponding DHT cores then update their location tables to record the location of the newly inserted data.

A communication schedule represents the sequence of data transfers required to correctly move data between coupled applications. When the data consuming application invokes a *get()* operation with the appropriate geometric descriptor, the execution client first translates the geometric descriptor into the corresponding set of index spans and queries the DHT for locations of the data. After successfully retrieving data locations, the execution client can compute the communication schedule. To optimize subsequent operations, the execution client caches this communication schedule. As data coupling patterns are often repeated in iteration based scientific simulations, these schedules can be reused, which improves performance.

The framework uses a receiver-driven pull approach to implement the data transfers in parallel for both, the concurrent and sequential coupling scenarios. Once the communication schedules are computed, the receiving execution clients issue one or more data requests to the processor cores where data is being produced (for the concurrent coupling scenario) or processor cores where data has been stored (for the sequential coupling scenario). The actual data transfers use HybridDART, which creates remotely accessible data buffers using either shared memory segments or RDMA memory regions, depending on whether the end-points of the data transfer are on the same node or on different nodes. Selected execution clients can directly *get*/*put* data from/to these buffers. HybridDART automatically selects the appropriate transport methods, avoiding the use of network interfaces to transfer data when the communicating execution clients run on different processor cores of the same compute node, using shared memory instead.

### *B. Data-Centric Task Mapping*

Our framework employs locality-aware, data-centric task placement to map computations from the coupled applications onto processor cores so that a large portion of the data exchanges can be performed using the on-node shared memory.

Server side data-centric task mapping is used for a "bundle" of concurrently coupled applications. Implementation of this mapping consists of two steps. The first step is the generation of the inter-application communication graph. Currently this step is performed offline before the workflow starts running, and is based on the data decomposition descriptor specified by users for each application. Each vertex of the communication

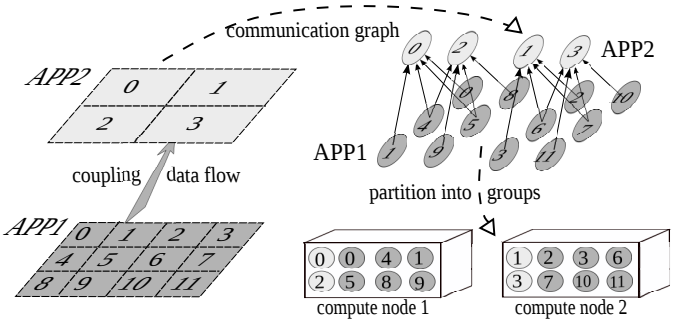


Fig. 7. Locality-aware, data-center partitioning of the inter-application communication graph for concurrently coupled applications.

graph represents a computation task of a parallel application in the "bundle", and each edge connects two communicating computation tasks from different applications that are concurrently coupled. The second step is performed by the workflow management server at runtime, which consists of partitioning the inter-application communication graph and the distribution of computation tasks. Before launching the applications within the "bundle", the workflow management server uses METIS [8] to partition the total *num_task* computation tasks into $num\_task/core\_count$ groups, where *core_count* represents the number of processor cores on a compute node. After the partitioning completes, each task group is mapped to a distinct compute node, and the associated computation tasks are distributed to the *core_count* processor cores in a round-robin fashion. Figure 7 shows the mapping of computation tasks from two concurrently coupled applications, APP1 and APP2 onto two 8-core compute nodes. In this simple scenario, APP1 runs 12 computation tasks and APP2 runs 4 computation tasks. The goal of the partitioning is to remove inter-application network-based data transfer links in the communication graph by grouping the communicating computation tasks onto the same multi-core compute node, so that data transfers are intra-node.

Decentralized client side data-centric task mapping is used to launch applications that are sequentially coupled with a preceding application in the workflow. As soon as the required computation resources are available for the pending applications, the workflow management server distributes computation tasks to the compute nodes in a round-robin fashion. After this initial task distribution, each execution client is assigned a computation task that has three attributes: application id, process rank, and its requested data region. The execution client then queries the *Data Lookup* service for the storage locations of the data region that demanded by the assigned computation task. The query result could contain one compute node where the entire demanded data region is stored, or multiple compute nodes that each stores a portion of the demanded data region. Each execution client selects only one compute node to map the assigned computation task, by maximizing the amount of coupled data that can be locally retrieved by the computation task.

### C. Dynamic Execution Clients Grouping and Application Launching

Our framework supports executing a parallel application on a set of dynamically selected processor cores, which enables running computation tasks of different applications on the same multi-core processor. Applications of a workflow are currently implemented as MPI-based subroutines that statically compiled and linked into the framework, so each execution client could dynamically select which application routine to run at runtime.

The execution client would get one assigned computation task when the mapping process is completed. Then each execution client is colored with the value of application id which is associated with the computation task and uniquely assigned by users. Execution clients with the same color form a processes group at runtime to execute a parallel application. A "bundle" that consists of $k$ concurrently coupled applications will divide the allocated execution clients into $k$ different groups. The execution clients then utilize *MPI_Comm_split* function to create a new communication domain or communicator for each processes group, and use the computation task's process rank value to control rank assignment within the group. Finally, execution client selects and runs the pre-linked MPI-based subroutine which matches the task's application id, and uses the newly created communicator for all subsequent intra-application communication.

Our prototype requires each application to be developed as a MPI-based subroutine of the framework, and statically linked at compile time. An ideal implementation would represent each application in the workflow as a separate MPI binary executable, schedule and execute these executables dynamically at runtime, which requires less source code modifications of user's legacy simulation programs. Dynamic process management features defined in MPI-2 standard, such as *MPI_Comm_spawn* and *MPI_Comm_spawn_multiple*, support runtime execution of MPI binaries. However, the targeted platform used by current implementation does not support these features.

## V. Experimental Evaluation

The prototype implementation of our framework was evaluated on the Jaguar Cray XT5 system at Oak Ridge National Laboratory. Jaguar XT5 has 18,688 compute nodes, and each compute node contains dual hex-core AMD Opteron processors and 12 cores, 16GB memory and a SeaStar2+ router that interconnects the nodes via a fast 3D torus network.

Our evaluation presented in this section consists of three parts. The first part evaluates the effectiveness of the framework's data-centric task mapping using different inter-application data transfer patterns. We also compared our data-centric task mapping strategy with the round-robin task mapping that employed by many MPI job launchers. The second part analyzes the framework's impact on various data communication costs within a coupled simulation workflow and explains the overall cost reduction that can be achieved by favoring the intra-node inter-application data transfers. The third part evaluates the scalability of the framework's data sharing substrate, i.e., CoDS.

The experiments used two testing workflow scenarios which are driven by the online data processing and coupled climate modeling examples. The first scenario concurrently couples two interacting applications which are referred to as **CAP1** and **CAP2**. CAP1 and CAP2 run concurrently and share data over a 3-dimensional data domain. The second scenario sequentially couples three applications which are referred to as **SAP1**, **SAP2** and **SAP3**. The execution of this sequential workflow first launches SAP1 which produces and stores data into the CoDS. When SAP1 completes, SAP2 and SAP3 would be launched to run on the same set of compute nodes used by SAP1, and retrieve coupled data from the CoDS. Similar to the first scenario, the three applications use a shared 3-dimensional common data domain.

### A. Effectiveness of the Data-centric Task Mapping

This section evaluates the effectiveness of our data-centric task mapping. More specifically, we ran experiments to measure the amount of inter-application data that transferred over the network, and the time used to transfer the coupled data, for both data-centric task mapping and the round-robin task mapping. In this case, we used a concurrent and a sequential testing scenario. Each computation task of the data producing applications (CAP1 and SAP1) was assigned a region of size $128\times128\times128$ from the global data domain, and inserts 32MB data into CoDS. For the concurrent coupling scenario, CAP1 and CAP2 separately ran on 512 and 64 cores, a total of 8GB inter-application coupled data was redistributed from CAP1 to CAP2. For the sequential coupling scenario, SAP1 first ran on 512 cores, then SAP2 and SAP3 separately ran on 128 and 384 cores. A total of 16GB data was redistributed from SAP1 to SAP2 and SAP3.

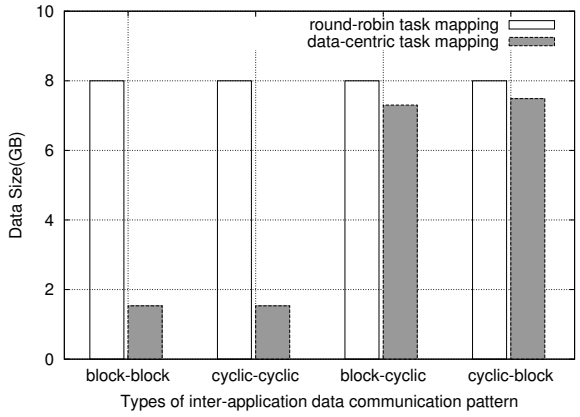


Fig. 8. Concurrent coupling scenario: Comparison of the amount of coupled data transferred over the network for the data-centric and round-robin task mapping cases, for different inter-application communication patterns.

Figure 8 and Figure 9 plot the amount of coupled data that is transferred over the communication fabric for the two coupling scenarios. The X axis of the figures represents the data decomposition pattern for the two coupled applications.

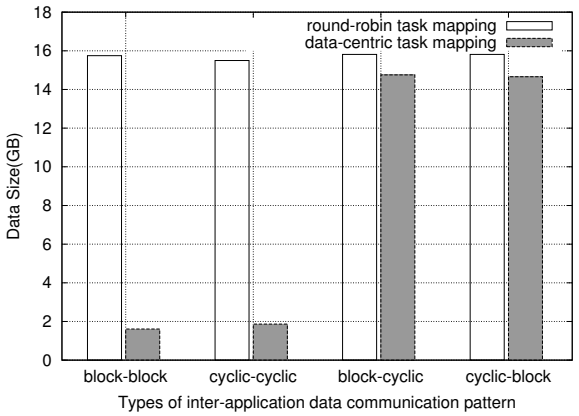


Fig. 9. Sequential coupling scenario: Comparison of the amount of coupled data transferred over the network for the data-centric and round-robin task mapping, for different inter-application communication patterns.

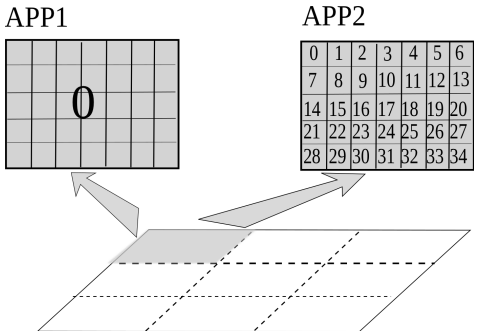


Fig. 10. Coupled data region when the two applications are decomposed differently.

As shown in the figures, the locality aware data-centric task mapping works effectively for both scenarios when the coupled applications have the same data distribution type. Compared to the round-robin task mapping,the data-centric task mapping transferred about 80% less data over the network by co-locating data producing and consuming computation tasks in CAP1 and CAP2, and transferred about 90% less data over the network by placing data consuming tasks in SAP2 and SAP3 closer to the data. Most of the data is retrieved in-situ using intra-node shared memory. However, our framework did not perform as well when the applications have different data distribution patterns. Figure 10 shows why it is harder for data-centric task mapping to be effective when two applications have different distributions. In Figure 10, the shaded region of the shared data domain is mapped to APP1 process 0 using a blocked distribution, and mapped to APP2 processes 0-34 using a block-cyclic distribution. If APP1 process 0 tries to get coupled data corresponding to the shaded region from APP2, it needs to retrieve data from each process of APP2, i.e., 0-34. As a result, a 1-to-N or even N-to-N inter-application communication patterns may occur, and the value of N can be much larger than the processor cores count when application runs at scale, which makes it harder to achieve in-situ workflow execution and intra-node inter-application data flow.

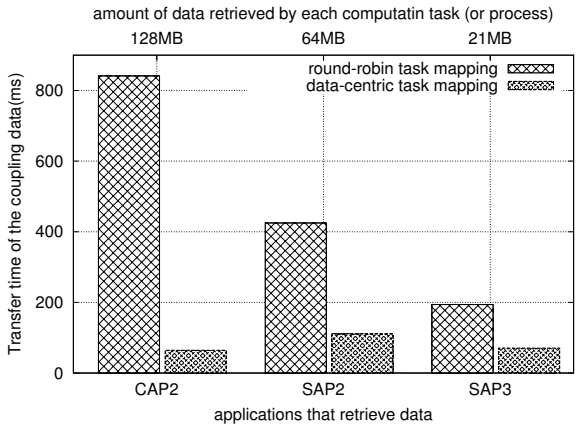


Fig. 11. Transfer time for the coupled data for the concurrent and sequential coupling scenarios. The bottom X axis indicates the application name, and the top X axis indicates the amount of data retrieved by one computation task in the corresponding application.

Figure 11 presents the time required to retrieve coupled data for applications CAP2, SAP2 and SAP3. For each application, the data transfer time decreased significantly for data-centric mapping when compared with the round-robin task mapping. This performance improvement was because most of the coupled data could be directly retrieved from the intra-node shared memory in case of data-centric task mapping. As shown in the figure, a;though SAP2 and SAP3 retrieve less data per computation task, the time to transfer data is longer than CAP2. The main reason is that in the sequential scenario, more data retrieve requests need to be processed because each of the 512 cores (in this case) requires certain data regions.

### *B. Impact on Intra-application Data Communication*

This experiment evaluates how our data centric task mapping affected the performance of the intra-application data communication. There are two primary types of data communications for applications within a coupled scientific workflow. The first is the inter-application data coupling in which the coupled data is transferred between the different component applications. The second is the intra-application data communication. This experiment used 2D or 3D stencil-like near-neighbor data exchanges to represent the cost of intra-application communication, which is common for the targeted class of data parallel scientific applications. The configurations used for these experiments for the two workflow scenarios, such as the number of processor cores used and dimensions of the data domain, were the same as those used in the previous experiment.

Figure 12 and Figure 13 illustrate the influence of data-centric task mapping on the cost of intra-application near-neighbors data exchange. As shown in the figures, data-centric task mapping almost doubled the amount of network transferred intra-application data exchanges, for application CAP2 in the concurrent workflow scenario, and SAP2 in the sequential workflow scenario. For other applications including

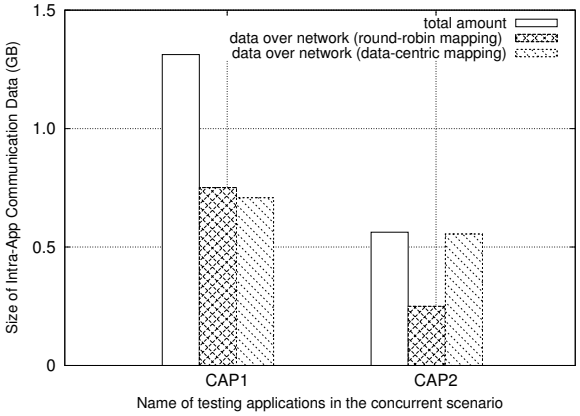


Fig. 12. Concurrent coupling scenario: Comparison of the amount of coupled data transferred over the network for the data-centric and round-robin task mapping cases, for different inter-application communication patterns.

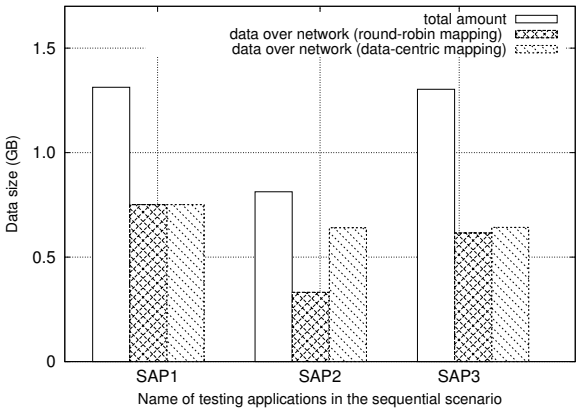


Fig. 13. Sequential coupling scenario: Comparison of intra-application data exchanges over the networks between round-robin and data-centric task mapping.

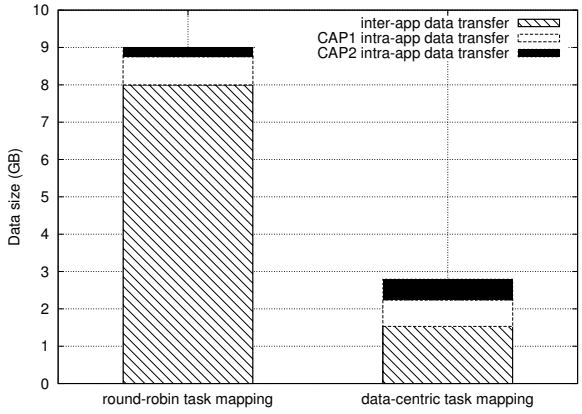


Fig. 14. Concurrent coupling scenario: Decomposition of the cost of data transfers over the network for round-robin and data-centric task mapping.

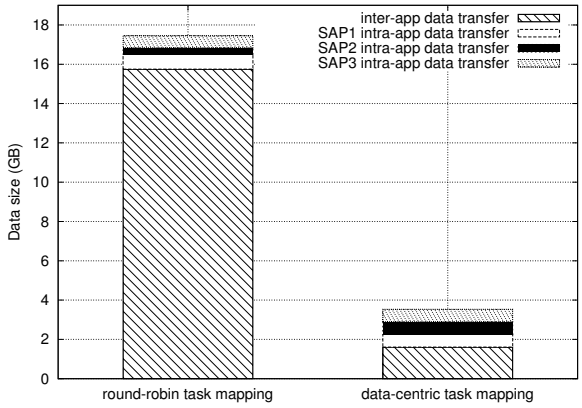


Fig. 15. Sequential coupling scenario: Decomposition of the cost of data transfers over the network for round-robin and data-centric task mapping.

CAP1, SAP1 and SAP3, the changes were very small. CAP2 and SAP2 share a common characteristic, i.e., both of them run on a smaller portion of the processor cores used for the coupled applications. For example, CAP2 runs on 64 cores out of 576, and SAP2 runs on 128 cores out of 512. Data-centric task mapping co-locates the data consuming application with the data producing application, or with the required data already produced and stored. By moving computation to data, the smaller number of computation tasks in CAP2 and SAP2 were more scattered across different compute nodes. As a result, while the amount of inter-application data transfers over the network greatly decreased, the intra-application data exchanges over the network in CAP2 and SAP2 increased.

Figure 14 and Figure 15 give a clearer view on the communication cost (measured as the amount of data transferred over network). In the two workflow scenarios used in the experiment, inter-application coupling requires redistribution of the volume of the entire shared data region, which resulted in larger transferred data size compared with the intra-application near-neighbor data exchanges. As shown by the figures, transferring the coupled data is the dominant cost when round-robin task mapping is applied. With data-centric task mapping, the significant decrease of inter-application data transfers over the network significantly reduces the overall communication cost. As a result, the effectiveness of the data-centric task mapping also depends on the ratio of inter-application data transfer size to intra-application data exchange size. As long as the large amount of data movement between applications is a concern for a coupled workflow, data-centric task mapping presents clear advantages.

### C. Scalability

This section evaluates the scalability of the framework. The experiment used weak scaling and varied the number of processor cores for the coupled applications and consequently the number of data retrieve queries. For the concurrent coupling workflow scenario, the number of processor cores for CAP1/CAP2 were varied from 512/64 to 8192/1024. For the sequential workflow scenario, the number of processor cores for SAP/(SAP2+SAP3) were varied from 512/(128+384) to 8192/(2048+6144). Each computation task of the data producing applications (i.e., CAP1 and SAP1) inserted 16MB of data into the space. Each computation task of CAP2 application retrieves 128MB data from CoDS, and each computation task

of SAP2 and SAP2 retrieves 64MB, 22MB data from CoDS respectively.

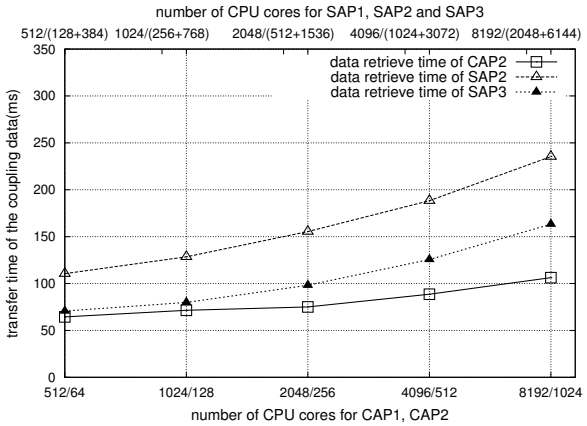


Fig. 16. Weak scaling of the time to retrieve coupled data with increasing numbers of cores. On the X axis, the bottom part represents the number of cores used in the concurrent coupling scenario, and the top part represents the number of cores used in the sequential coupling scenario.

The results in Figure 16 show good overall scalability with increasing number of application computation tasks and transferred data sizes. The total amount of transferred data was increased 16-fold between the small and large scale cases, from 8GB to 128GB for the concurrent coupling scenario, and 16GB to 256GB for the sequential coupling scenario. The data retrieve time for the three applications CAP2, SAP2 and SAP3 had only small increase, i.e., less than 150 ms. This increase in transfer time is mainly due to the contention on the shared network links, which is caused by the increasing number of concurrent data transfers at larger application scale. Furthermore, as shown in Figure 16, the rate of increase in the transfer time for SAP2 and SAP3 at larger scales is higher than CAP2. Although each single computation task of SAP2 and SAP3 gets less data than CAP2, the total number of concurrent data retrieve queries in the sequential coupling scenario is double of that in the concurrent coupling scenario, and SAP2 and SAP3 request data simultaneously in this case, which caused the observed behavior.

## VI. Related Work

This section summarizes previous research efforts related to our work.

***Scientific workflow management systems***: Scientific workflow engines such as Pegasus [9] and KEPLER [11] are used to automate resources (i.e. data, compute nodes) management and execution of the loosely coupled applications. Data sharing between the different component applications are usually performed by reading data files stored in the distributed file systems, thus the performance is affected by the system IO performance. Our framework targets at more tight coupling workflow scenario, employs the direct memory-to-memory approach to transfer data between concurrently coupled applications, and implements a distributed in-memory storage to facilitate sequential data sharing. Compared to the file-based approach, our framework provides faster and more scalable data sharing service.

***Distributed tasks scheduling and execution frameworks***: Several recent projects provide runtime systems to improve performance of computation tasks scheduling and execution on emerging distributed multicore architecture. DAGuE [12] proposes a generic engine to express numerical algorithm as a DAG of tasks at a finer granularity, and dynamically schedule the tasks execution at runtime. StarPU [13] provides a unified execution model and runtime system to support execution of parallel tasks over heterogeneous hardware. These frameworks provides programming interface and runtime system for development of numerical computation kernels (i.e. BLAS routines, FFT) that exploit the heterogeneous multicore architecture. Task scheduling and execution in our framework focuses on mapping tasks (or processes) from multiple data parallel programs to multi-core processors to increase data locality and reuse between the coupled applications within a scientific workflow.

***Data coupling software tools***: The M×N working group in the Common Component Architecture (CCA) [14] forum provides a package of software tools to perform parallel data redistribution between coupled simulation codes, including Meta-Chaos [15], InterComm [16], MCT [17], Parallel Application Work Space (PAWS) [18]. CCA forum defines a set of standard interfaces to promote interoperability between tools developed by different organizations. These software libraries are often integrated into simulation code to perform inter-application data communications with other simulations. Our framework additionally provides interface to compose tightly coupled workflow and enables in-situ placement of applications tasks to decrease network-based data transfers.

***In-situ data analytics and visualization***: The increasing performance gap between computing and IO, and the cost of moving large volume of data to/from disks, motivates computational scientists to propose a new approach to perform scientific data processing - in-situ data analysis and visualization [19], [20], [21]. The key idea is to move data analytic operations or visualization computation to data where the simulation is running. Existing implementations of in-situ visualization [22] tightly integrate analysis or visualization libraries into simulation code. Our framework's capability to run in-situ concurrent coupling workflow, can also be applied to in-situ data analysis and visualization. Moreover, our work provides a generic framework to compose and execute in-situ workflow in a flexible and customized way.

***Staging area based data sharing and exchange***: The data staging area, a set of additional compute nodes allocated by users when launching the parallel simulations, and the application of staging area has been investigated to add values to simulation's IO pipeline [10], [23], [24], [25]. DataSpace [10] project builds a distributed in-memory storage in the staging area to support data sharing between coupled applications. During the execution of a workflow, scientific data is asynchronously extracted from one simulation, stored and indexed in the staging area, and then accessed by other

simulations. This approach requires coupled data to be shared indirectly through the staging area, which would result two data movements (i.e., data producing application to the space, then space to data consuming application) and cause extra cost for tightly coupled scientific workflow. The direct application-to-application in-memory data sharing in our framework provides a faster mechanism to move data between tightly coupled applications.

## VII. Conclusion and Future Work

This paper explored the in-situ execution of the coupled components of a scientific application workflow so as to maximize on-chip exchange of data. This work is motivated by the observation that the movement of large volumes data over the communication fabrics is significantly impacting the performance of coupled scientific application workflows. Specifically, we presented the design and implementation of a distributed data sharing and task execution framework that (1) employs data-centric task placement to map computations from the coupled application components onto processor cores so that a large portion of the data exchanges can be performed using the on-node shared memory, and (2) provides a shared space programming abstraction that supplements existing parallel programming models (e.g., message passing) with DAG-based workflow descriptions and specialized one-sided asynchronous data access operators and can be used to express coordination and data exchanges between the coupled components. The presented in-situ execution approach can be applied to a range of application scenarios such as online data analysis and visualization, multiphysics coupled simulations, and to a wide range of applications domains.

We also presented the implementation of the framework and its experimental evaluation on the Jaguar Cray XT5 at Oak Ridge National Laboratory. The evaluation used two coupled workflow scenarios motivated by real-world applications, and demonstrated the effectiveness and performance of the proposed data-centric task mapping. In addition, the paper analyzed the trade-off of using the framework, and evaluated the scalability of CoDS data sharing substrate.

Our directions for future work include extending the framework to enable task mapping and execution on emerging heterogeneous multicore platforms with accelerators (i.e., GPUs). While the current implementation targets at parallel applications developed with message passing programming model (e.g., MPI), we will also explore supporting other programming models such as Partitioned Global Address Space (PGAS) and MapReduce.

## Acknowledgment

The research presented in this paper is supported in part by National Science Foundation via grant numbers IIP 0758566 and DMS-0835436, by Department of Energy via the grant number DE-FG02-06ER54857, and by an IBM Faculty Award, and was conducted as part of the Center for Autonomic Computing at Rutgers University.

## References

[1] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. J. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S. McKenna, B. D. Santer, and R. D. Smith, "The Community Climate System Model Version 3 (CCSM3)," *Journal of Climate*, vol. 19, no. 11, pp. 2122–2143, 2006.

[2] J. Cummings, J. Lofstead, K. Schwan, A. Sim, A. Shoshani, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and R. Barreto, "EFFIS: An End-to-end Framework for Fusion Integrated Simulation," in *Proc. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'10)*, Feburary 2010.

[3] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, Metadata Rich IO Methods for Portable High Performance IO," in *Proc. 23th IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, May 2009.

[4] A. Bhatele and, G. Gupta, L. Kale and, and I.-H. Chung, "Automated Mapping of Regular Communication Graphs on Mesh Interconnects," in *Proc. International Conference on High Performance Computing (HiPC'10)*, December 2010.

[5] T. Agarwal, A. Sharma, A. Laxmikant, and L. Kale, "Topology-aware Task Mapping for Reducing Communication Contention on Large Parallel Machines," in *Proc. 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, April 2006.

[6] F. Bertrand, R. Bramley, A. Sussman, D. Bernholdt, J. Kohl, J. Larson, and K. Damevski, "Data Redistribution and Remote Method Invocation in Parallel Component Architectures," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, April 2005.

[7] C. Docan, M. Parashar, and S. Klasky, "DART: A Substrate for High Speed Asynchronous Data IO," in *Proc. of 17th International Symposium on High Performance Distributed Computing (HPDC'08)*, June 2008.

[8] G. Karypis and V. Kumar, "METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0," Dept. of Computer Science, Univ. of Minnesota, Tech. Rep., 1995.

[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Sci. Program.*, vol. 13, pp. 219–237, July 2005.

[10] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows," in *Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10)*, June 2010.

[11] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and The Lepler System: Research Articles," *Concurrency and Computation : Practical and Experience*, vol. 18, pp. 1039–1065, August 2006.

[12] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, H. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra, "Distributed-Memory Task Execution and Dependence Tracking within DAGuE and the DPLASMA Project," Innovative Computing Laboratory, University of Tennessee, Tech. Rep., 2010.

[13] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, 2011.

[14] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High-Performance Scientific Computing," in *Proc. of 8th International Symposium on High Performance Distributed Computing (HPDC'99)*, August 1999.

[15] G. Edjlali, A. Sussman, and J. H. Saltz, "Interoperability of Data Parallel Runtime Libraries," in *Proc. of the 11th International Symposium on Parallel Processing (IPPS'97)*, April 1997.

[16] J.-Y. Lee and A. Sussman, "High Performance Communication between Parallel Programs," in *Proc. of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, April 2005.

[17] J. Larson, R. Jacob, and E. Ong, "The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models," *International Journal of High Performance Computing Applications (IJHPCA)*, vol. 19, pp. 277–292, August 2005.

[18] P. Fasel and S. Mniszewski, "PAWS: Collective Interactions and Data Transfers," in *Proc. of the 10th International Symposium on High Performance Distributed Computing (HPDC'01)*, August 2001.

[19] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, "In Situ Visualization for Large-Scale Combustion Simulations," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010.

[20] K.-L. Ma, "In Situ Visualization at Extreme Scale: Challenges and Opportunities," *IEEE Computer Graphics and Applications*, vol. 29, no. 6, pp. 14–19, 2009.

[21] H. Childs, "Architectural Challenges and Solutions for Petascale Post-processing," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012012, 2007.

[22] J.-M. F. Brad Whitlock and J. S. Meredith, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[23] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA - preparatory data analytics on peta-scale machines," in *Proc. of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, April 2010.

[24] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky, "Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging," in *Proc. 20th International Symposium on High Performance Distributed Computing (HPDC'11)*, June 2011.

[25] C. Docan, M. Parashar, J. Cummings, and S. Klasky, "Moving the Code to the Data - Dynamic Code Deployment Using ActiveSpaces," in *Proc. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*, May 2011.