Enabling Efficient and Flexible Coupling of Parallel Scientific Applications *

Li Zhang and Manish Parashar

The Applied Software Systems Laboratory (TASSL)

Rutgers University

94 Brett Road, Piscataway, NJ 08854, USA

{emmalily, parashar}@caipclassic.rutgers.edu

Abstract

Emerging scientific and engineering simulations are presenting challenging requirements for coupling between multiple physics models and associated parallel codes that execute independently and in a distributed manner. Realizing coupled simulations requires an efficient, flexible and scalable coupling framework and simple programming abstractions. This paper presents a coupling framework that addresses these requirements. The framework is based on the Seine geometry-based interaction model. It enables efficient computation of communication schedules, supports low-overheads processor-to-processor data streaming, and provides high-level abstraction for application developers. The design, CCA-based implementation, and experimental evaluation of the Seine based coupling framework are presented.

1. Introduction

Scientific and engineering simulations are becoming increasingly sophisticated as strive to achieve more accurate solutions to realistic models of complex phenomena. A key aspect of these emerging simulations is the modeling of multiple interacting physical processes that comprise the phenomena being modeled, which leads to challenging requirements for coupling between multiple physical models and associated parallel codes that execute independently and in a distributed manner. For example, in plasma science, an integrated predictive plasma edge simulation couples an edge turbulence code with a core turbulence code through common grids at the spatial interface [17]. Similarly, in geosciences, multiple scales in the domain and multiple physics models are coupled via shared boundaries between neighboring entities [18]. These coupled systems provide the individual models with a more realistic simulation environment, allowing them to be interdependent on and interact with other physics models in the coupled system and to react to dynamically changing boundary conditions.

However, achieving efficient, flexible and scalable coupling of physics models and parallel application codes presents significant algorithmic, numerical and computational challenges. From the computational point of view, the coupled simulations, each typically running on a distinct parallel system or set of processors with independent (and possibly dynamic) distributions, need to periodically exchange information. Specifically, this requires that: (1) interaction/communication schedules between individual processors executing each of the coupled simulations need to be computed efficiently, locally, and on-the-fly, without requiring synchronizations or gathering global information, and without incurring significant overheads on the simulations themselves; and (2) data transfers should also be efficient and should happen directly between the individual processors of each simulation. Furthermore, specifying these coupling behaviors between the simulations codes using popular message-passing abstractions can be cumbersome and often inefficient, as these systems require matching sends and receives to be explicitly defined for each interaction. As the individual simulations become larger, more dynamic and heterogeneous and their couplings more complex, implementations using message passing abstractions can quickly become unmanageable. Clearly, realizing coupled simulations requires an efficient, flexible and scalable coupling framework and simple high-level programming abstractions.

This paper presents a coupling framework that addresses these requirements. The framework is based on the Seine geometry-based interaction model [12], which is motivated by two observations about the targeted applications: (a) formulations of these scientific and engineering applications are based on multi-dimensional geometric discretizations of

the problem domain (e.g., grid or mesh) and (b) couplings and interactions in these applications can be defined based on geometric relations in this discretization (e.g., intersecting or adjacent regions). Seine provides a geometry-based virtual shared space interaction abstraction. This abstraction derives from the tuple space model. However, instead of implementing a general and global interactions space (as in the tuple model), Seine presents an abstraction of transient geometry-based interaction spaces, each of which is localized to a sub-region of the overall geometric domain. This allows the abstraction to be efficiently and scalably implemented and allows interactions to be decoupled at the application level. A Seine interaction space is defined to cover a closed region of the application domain described by an interval of coordinates in each dimension, and can be identified by any set of coordinates contained in the region.

The Seine geometry-based coupling framework differs from existing approaches in several ways. First, it provides a simple but powerful abstraction for interaction and coupling in the form of the virtual geometry-based shared space. Processes register geometric regions of interest, and associatively read and write data associated with the registered region from/to the space in a decoupled manner. Second, it supports efficient local computation of communication schedules using lookups into directory implemented as a distributed hash table. The index space of the hash table is directly constructed from the geometry of the application using Hilbert space filling curves [13]. Processes register their regions of interest with the directory layer, and the directory layer automatically computes communications schedules based on overlaps between the registered geometric regions. Registering processes do not need to know of or explicitly synchronize with other processes during registration and the computation of communication schedules. Finally, it supports efficient and low-overhead processor-to-processor socket-based data streaming and adaptive buffer management. The Seine model and the Seine-based coupling framework is designed to complement existing parallel programming models and can work in tandem with systems such as MPI, PVM and OpenMP.

The design, implementation and experimental evaluation of the Seine based coupling framework are presented. The implementation is based on the DoE Common Component Architecture (CCA) [5] and enables coupling within and across CCA-based simulations. The experimental evaluation measures the performance of the framework for various data redistribution patterns and different data sizes. The results demonstrate the performance and overheads of the framework.

The rest of the paper is organized as follows. Section 2 presents some background and discusses related work. Section 3 introduces the Seine geometry-based interaction model and the design of the Seine-based coupling frame-

work. Section 4 presents the CCA-based implementation of the coupling framework and an experimental evaluation of its performance. Section 5 presents a conclusion and outlines future research directions.

2. Background and Related Work

Parallel data redistribution (also termed the MxN problem) is a key aspect of the coupling problem, since it addresses the problem of transferring data from a parallel program/model running on M processors to another parallel program/model running on N processors. Different aspects of this problem have been addressed by recent projects such as Model Coupling Toolkit [7], InterComm [4], PAWS [3], CUMULVS [2], DCA [9], SciRun2 [11], etc., with different foci and approaches. One approach, used by component-based systems such as CCA, encapsulates parallel data redistribution support into a standard component, which can then be composed with other components to realize different coupling scenarios. An alternate approach embeds the parallel data redistribution support into a Parallel Remote Method Invocation (PRMI) [10] mechanism. This PRMI-based approach addresses issues other than just data redistributions, such as remote method invocation semantic for non-uniformly distributed components. Existing projects based on these two approaches are summarized in Table 1.

Projects such as Model Coupling Toolkit (MCT), InterComm, PAWS (Parallel Application Workspace), CUMULVS (Collaborative User Migration, User Library for Visualization and Steering), and DDB (Distributed Data Broker) use the component-based approach. As presented in the table, some of these systems have only partial or implicit support for parallel data redistribution, or can support only a limited set of data redistribution patterns. Projects such as PAWS and InterComm fully address the parallel data redistribution problem. These systems can support random data redistribution patterns. PRMI-based projects include SciRun2, DCA and XCAT.

While the projects discussed above address aspects of parallel data redistribution and coupling problem, these systems differ in the approaches they use to compute communication schedules, the data redistribution patterns that they support, and the abstractions they provide to the application developer. Most of the existing systems gather distribution information from all the coupled models at each processor and then locally compute data redistribution schedules. This implies a global synchronization across all the coupled systems, which can be expensive and limit scalability. Further, abstractions provided by these systems are based on message passing, which require explicit matching sends and receives and synchronous data transfers. Moreover, expressing very general redistribution patterns using message passing type abstractions can be quite cumbersome.

Table 1. Parallel Data Redistribution Projects

Approach I: Component-based Parallel Data Redistributions		
Project Name	**Brief Overview**	**MxN Support**
MCT [7]	Facilitates model coupling between model components in the Earth System Modelling Framework (E-SMF) [8]. A flux coupler in ESMF uses MCT functionality to transfer data between physics simulation components.	Implicit
InterComm [4]	Provides the support for direct data transfer between different parallel programs. It achieves efficient communication in the presence of complex data distributions for multi-dimensional array data structures.	Full
PAWS [3]	Provides the ability to share data structures between parallel applications. Multi-dimensional arrays in PAWS can be partitioned and distributed completely generally. A central controller is used.	Full
CUMULVS [2]	Is a middleware library aimed to provide support for remote visualization and steering of parallel applications and sharing parallel data structures between programs. Array distribution pattern is restricted.	Mx1
DDB [14]	Handles distributed data exchanges between ESM (Earth Science Model) components. DDB is designed to avoid centralized coupling.	Implicit
Approach II: PRMI-based Parallel Data Redistribution		
SciRun2 [11]	Defines PRMI and data redistribution as extensions to the SIDL [6] language. It supports all-to-all or one-to-one process participation in the parallel RMI.	Partial
DCA [9]	Is a prototype distributed CCA framework built on top of MPI. It adopts many MPI concept in defining process participation, MxN data redistribution, and argument passing in Parallel RMI.	Partial
XCAT [15]	Is a distributed CCA framework based on Globus that uses RMI over XSOAP. It supports parallel data redistribution for the case of M=N.	Partial

The Seine geometry-based coupling framework supports the component-based approach. It provides a simple but powerful high-level abstraction, based on a virtual associative shared space, to the application developer. Communication schedules are computed locally and in a decentralized manner using a distributed directory layer. The directory layer automatically detects overlaps between registered regions and computes communications schedules. All interactions are completely decoupled and data transfer is socket-based and processor-to-processor, and can be synchronous or asynchronous. The Seine coupling framework is described in detail in the following section.

3. The Seine Geometry-based Coupling Framework

Seine is a dynamic geometry-based coupling/interaction framework for parallel scientific and engineering applications. It is derived from the tuple space model and provides the abstraction of a virtual shared space, allowing it to support decoupled and extremely dynamic communication and coordination patterns. It is based on the observations that (a) formulations of the targeted scientific applications are based on multi-dimensional geometric discretizations of the problem domain (e.g., grid or mesh) and (b) couplings and interactions in these applications can be defined based on geometric relations in this discretization (e.g., intersecting or adjacent regions), and allows virtual interaction/coupling spaces to be localized to specific regions of interaction in the discretized application domain. This enables efficient

and scalable implementations. Seine spaces can be dynamically created and destroyed. Finally, Seine complements existing parallel programming models and can co-exist with them during program execution. The Seine interaction/coupling model and framework are described in detail in this section.

3.1. The Seine Geometry-based Coupling Model

Conceptually, the Seine coupling/interaction model is based on the tuple space model where entities interact with each other by sharing objects in a logically shared space. However there are key differences between the Seine model and the general tuple space model. In the general tuple space model, the tuple space is global, spans the entire application domain, can be accessed by all the nodes in computing environments, and support a very generic tuple-matching scheme. These characteristics have presented several implementation challenges for the general tuple model. In contrast, Seine defines a virtual dynamic shared space that spans a geometric region, which is a subset of the entire problem domain, and is accessible to only the dynamic subset of nodes to which the geometric region is mapped. Further, objects in the Seine space are geometry-based, i.e. each object has geometric descriptor, which specifies the region in the application domain that the object is associated with. Applications use these geometric descriptors to associatively *put* and *get* objects to/from a Seine space. Interactions are naturally decoupled.

Seine provides a small set of very simple primitives,

Table 2. Primitives of the Seine geometry-based coupling/interaction framework.

Primitives	Description
init(bootstrap server IP)	Uses a bootstrap mechanism to initialize the Seine runtime system.
register(object geometric descriptor)	Registers a region with Seine.
put(object geometric descriptor, object)	Inserts a geometric object into Seine.
get(object geometric descriptor, object)	Retrieves and removes a geometric object from Seine. This call will block until a matching object is *put*.
rd(object geometric descriptor, object)	Copies a geometric object from Seine without removing it. Multiple *rd* can be simultaneously invoked on an object. It returns immediately if the object searched for does not exist.
deregister(object geometric descriptor)	De-registers a region from Seine.

which are listed in Table 2. The *register* operation allows a process to dynamically register a region of interest, which causes it to join an appropriate existing space or a new space to be created if one does not exist. The *put* operator is used to write an object into the space, while the *get* operator reads a matching object from the space, if one exists. The *get* operation is blocking, i.e., it blocks until a matching object is *put* into the space. *rd* copies a geometric object from Seine without removing it. It is non-blocking, i.e., it returns immediately with an appropriate return code if a matching object does not exist. The *deregister* operation allows a processor to de-register a previously registered region. The operation of Seine is described in more detail later in this section.

3.2. Design of the Seine Geometry-based Coupling Framework

A schematic of the Seine architecture is presented in Figure 1. The framework consists of three key components: (1) A distributed directory layer that enables the registration of spaces and the efficient lookup of objects using their geometry descriptors; (2) A storage layer consisting of local storage at each processor associated with a shared space and used to store its shared objects; (3) A communication layer that provides efficient data transfer between processors.

Directory layer: The Seine distributed directory layer is used to (1) detect geometric relationships between shared geometry-based objects, (2) manage the creation of shared spaces based on the geometric relationship detected so that all objects associated with intersecting geometric regions are part of a single shared space, (3) manage the operation
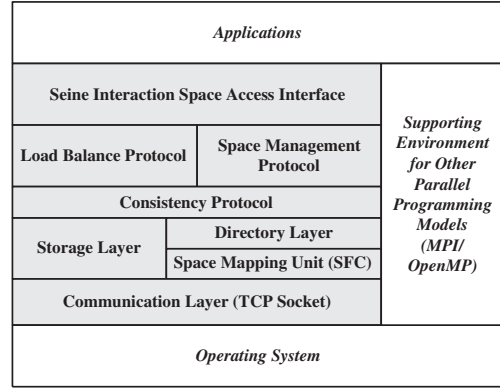

Figure 1. Architecture of the Seine geometry-based coupling/interaction framework.

of geometry-based shared spaces during their lifetimes including the merging of multiple spaces into a single space and the splitting of a space into multiple spaces, and (4) manage the destruction of a shared space when it is no longer needed.

The directory layer is essentially a distributed hash table where the index space of the table is directly constructed from the geometry of the discretized computational domain using the Hilbert space filling curve (SFC). Space-filling curves [13] are a class of locality preserving mappings from d-dimensional space to 1-dimensional space, i.e. $N^d \rightarrow N^1$, such that each point in N^d is mapped to a unique point or index in N^1. Using this mapping, a point in the N^d can be described by its spatial or d-dimensional coordinates, or by the length along the 1-dimensional index measured from one of its ends. The construction of SFCs is recursive and the mapping functions are computationally inexpensive and consist of bit level interleaving operations and logical manipulations of the coordinates of a point in multi-dimensional space. SFCs are locality preserving in that points that are close together in the 1-dimensional space are mapped from points that are close together in the d-dimensional space.

The Hilbert SFC is used to map the d-dimensional coordinate space of the computational domain to the 1-dimensional index space of the hash table. The index space is then partitioned and distributed to the processors in the system. As a result, each processor stores a span of the index space and is responsible for the corresponding region of the d-dimensional application domain. The processor manages the operation of the shared space in that region, including space creation, merges, splits, memberships and deletions. As mentioned above, object sharing in Seine is based on their geometric relationships. To share objects corresponding to a specific region in the domain, a processor must first register the region of interest with the Seine run-

time. A directory service daemon at each processor serves registration requests for regions that overlap with the geometric region and corresponding index span mapped to that processor. Note that the registered spaces may not be uniformly distributed in the domain and as a result, registration load must be balanced while mapping and possibly remapping index spans to processors.

To register a geometric region, the Seine runtime system first maps the region in the d-dimensional coordinate space to a set of intervals in the 1-dimensional index space using the Hilbert SFC. The index intervals are then used to locate the processor(s) to which they are mapped. The process of locating corresponding directory processors is efficient and only requires local computation. The directory service daemon at each processor maintains information about currently registered shared spaces and associated regions at the processor. Index intervals corresponding to registered spaces at a processor are maintained in an interval tree. A new registration request is directed to the appropriate directory service daemon(s). The request is compared with existing spaces using the interval tree. If overlapping regions exist, a union of these regions is computed and the existing shared spaces are updated to cover the union. Note that this might cause previously separate spaces to be merged. If no overlapping regions exist, a new space is created.

Storage layer: The Seine storage layer consists of the local storage associated with registered shared spaces. The storage for a shared space is maintained at each of the processors that have registered the space. Shared objects are stored at the processors that own them and are not replicated. When an object is written into the space, the update has to be reflected to all processors with objects whose geometric regions overlap with that of the object being inserted. This is achieved by propagating the object or possibly corresponding parts of the object (if the data associated with the region is decomposable based on regions, such as multidimensional arrays) to the processors that have registered overlapping geometric regions. Such an update propagation mechanism is used to maintain consistency of the shared space. As each shared space only spans a local communication region, it typically maps to a small number of processors and as a result update propagation does not result in significant overheads. Further, unique tags are used to enable multiple distinct objects to be associated with the same geometric region. Note that Seine does not impose any restrictions on the type of application data structures used. However, the current implementation is optimized for multi-dimensional arrays.

Communication layer: Since coupling and parallel data redistribution for scientific application typically involves communicating relatively large amounts of data, efficient communication and buffer management is critical. Further,

this communication has to be directly between individual processors. Currently Seine maintains the communication buffers at each processors as a queue, and multiple sends are overlapped to better utilize available bandwidth [16]. Adaptive buffer management strategies described in [16] are currently being integrated.

3.3. Coupling Parallel Scientific Applications using Seine

A simple parallel data redistribution scenario shown in Figure 2(a) is used to illustrate the operation of the Seine coupling framework. In this scenario, data associated with 2-dimensional computational domain of size 120*120 is coupled between two parallel simulations running on 4 and 9 processors respectively. The data decomposition for each simulation and the required parallel data redistribution are shown in the figure. Simulation M is decomposed into blocks M.1 - M.4 and simulation N is decomposed into blocks N.1 - N.9. The Seine coupling framework coexists with these simulations and is also distributed across 4 processors in this example. This is shown in the top portion of Figure 3. Note that these processors may or may not overlap with the simulation processors. Figure 3 also illustrates the steps involved in coupling and parallel data redistribution using Seine, which are described below.

The initialization of the Seine runtime using the *init* operator is shown in Figure 3 (a). During the initialization process, the directory structure is constructed by mapping the 2-dimensional coordinate space to a 1-dimensional index using the Hilbert SFC and distributing index intervals across the processors.

In Figure 3 (b), processor 1 registers an interaction region R1 (shown using a darker shade in the figure) in the center of the domain. Since this region maps to index intervals that spans all four processors, the registration request is sent to the directory service daemon at each of these processors. Each daemon services the request and records the relevant registered interval in its local interval tree. Once the registration is complete, a shared space corresponding to the registered region is created at processor 1 (shown as a cloud on the right in the figure).
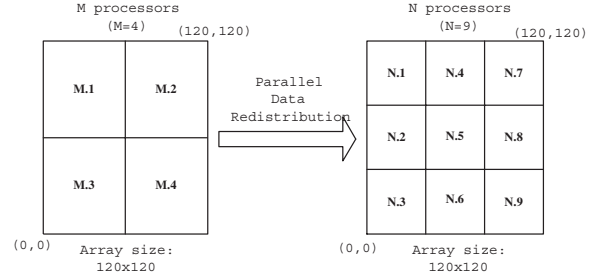
In Figure 3 (c), another processor, processor 0, registers region R2 (shown using a lighter shade in the figure). Once again, the region is translated into index intervals and corresponding registration request is forwarded to appropriate directory service daemons. Using the existing intervals in its local interval tree, the directory service daemons detect that the newly registered region overlaps with an existing space. As a result, processor 0 joins the existing space and the region associated with the space is updated to become the union of the two registered regions. The shared space also grows to span both processors. As more regions are

registered, the space is expanded if these regions overlap with the existing region, or new spaces are created if the regions do not overlap.
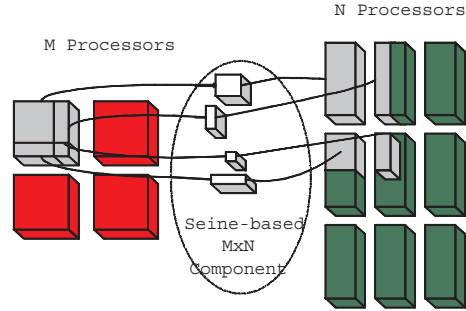
Once the shared space is created, processors can share geometry-based objects using the space. This is illustrated in Figures 3 (d) and (e). In Figure 3 (d), processor 0 uses the *put* operation to insert object 2 into the shared space. As there is an overlap between the regions registered by processors 0 and 1, the update to object 2 is propagated from processor 0 to processor 1. The propagated update may only consist of the data corresponding to the region of overlap, e.g., a sub-array if the object is an array. In Figure 3 (e), processor 1 retrieves object 1 using a local *get* operation.

Building coupled simulations using the Seine abstractions: Seine provides a virtual dynamic geometry-based shared space abstraction to the parallel scientific applications. Developing coupled simulations using this abstraction consists of the following steps. First, the coupled simulations register their geometric regions of interests with Seine. The registration phase detects geometric relationships between registered regions and results in the creation of a virtual shared space localized to the region and the derivation of associated communication schedules. Coupling data between simulations consists of one simulation writing the data into the space, along with a geometric descriptor describing the region that it belongs to; and the other simulation independently reading data from the space with an appropriate geometric descriptor. The communication schedule associated with the space and the Seine communication layer is used to set up parallel point-to-point communication channels for direct data transfer between source and destination processors. The associated parallel data redistribution is conceptually illustrated in Figure 2(b).

Computation of communication schedules: Communication schedules in the context of coupling and parallel data redistribution refer to the sequence of messages required to correctly move data among coupled processes [10]. As mentioned above, these schedules are computed in Seine during registration using the Hilbert SFC-based linearization of the multidimensional application domain coordinate space. When a region is registered, the Seine directory layer uses the distributed hash table to route the registration request to corresponding directory service node(s). The directory service node is responsible for detecting overlaps or geometric relationships between registered regions efficiently. This is done by detecting overlaps in corresponding 1-dimensional index intervals using the local interval tree. Note that all registration requests that are within a particular region of the application domain are directed to the same Seine directory service node(s), and as a result, the node(s) can correctly compute the required schedules. This is in contrast to most existing systems which require informa-

(a) An illustrative example

(b) Abstraction of Seine-based parallel data redistribution

Figure 2.

tion about the distributions of all the coupled processes to be gathered.

Data transfer: When an object is written into a space, Seine propagates the object (or possibly the appropriate part of the object, e.g., if the object is an array) to update remote objects based on the relationships between registered geometric regions, using the communication layer.

4 Prototype Implementation and Performance Evaluation

4.1 A CCA-based Prototype Implementation

The current prototype implementation of the Seine coupling framework is based on the DoE Common Component Architecture (CCA) [5] and enables coupling within and across CCA-based simulations. In CCA, components communicate with each other through *ports*. There are two basic types of *ports*, the *provides* port and the *uses* port. Connections between components are achieved by wiring between a *provides* port on one component and a *uses* port on the other component. The component can invoke methods on the *uses* port once it is connected to a *provides* port. CCA ports are specified using the Scientific Interface Definition Language (SIDL) [6]. CCA frameworks can be distributed
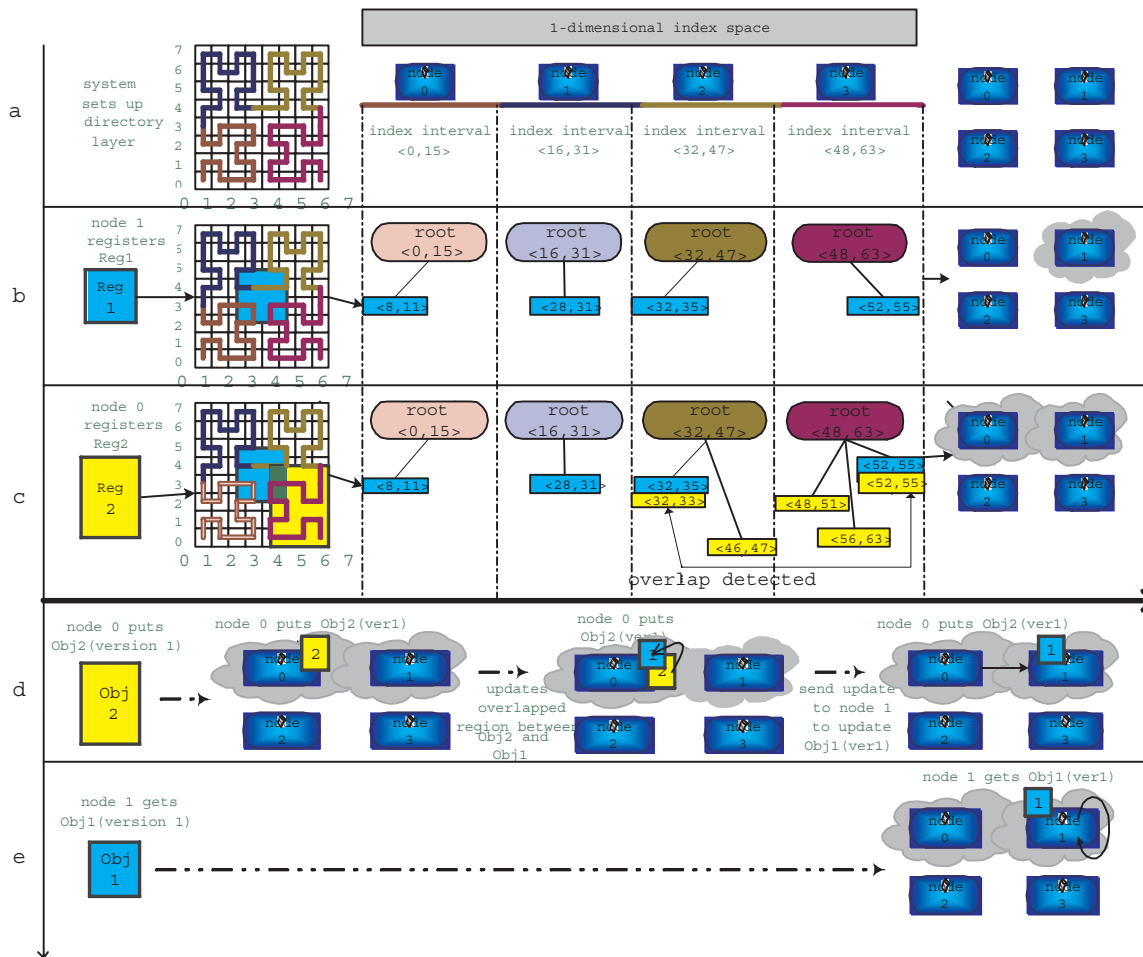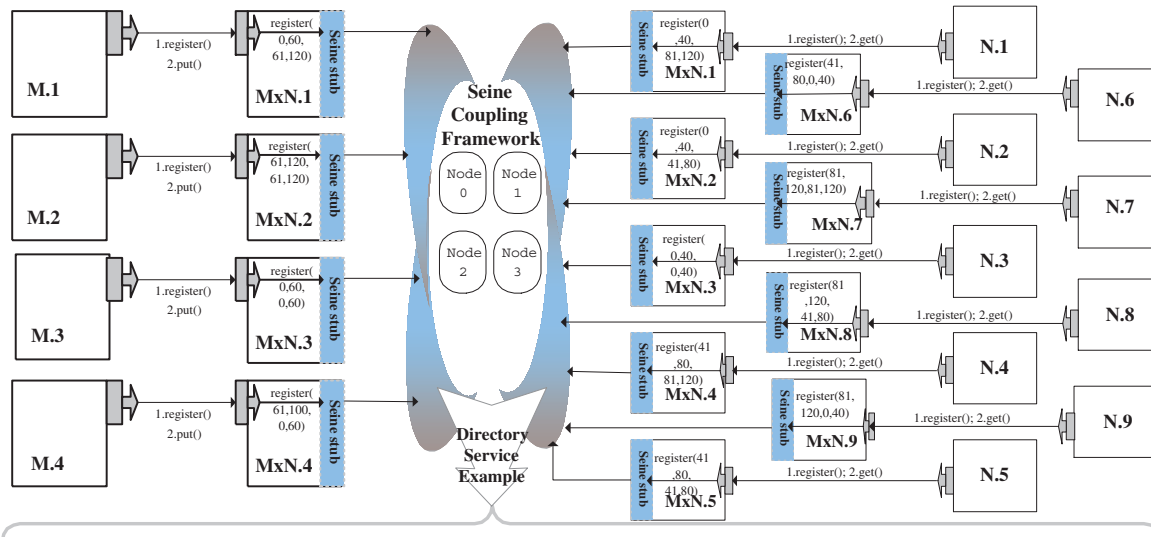
Figure 3. Operation of coupling and parallel data redistribution using Seine.
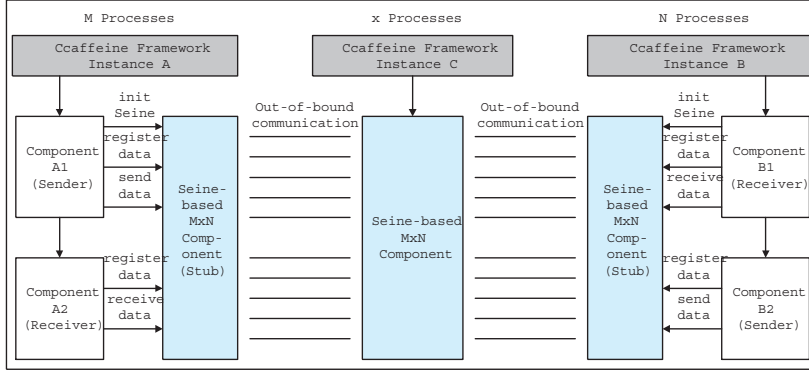
Figure 4. MxN data redistribution between CCA frameworks using the Seine coupling component.

or direct-connected. In a direct-connected framework all components in one process live in the same address space. Communication between components, or the port invocation, is local to the process.

The Seine coupling framework was encapsulated as a CCA compliant component within the CCAFFEINE direct-connected CCA framework, and is used to support coupling and parallel data redistribution between multiple instances of the framework, each executing as an independent simulation, as well as within a single framework executing in the multiple component-multiple data (MCMD) mode. The former setup is illustrated in Figure 4. In the figure, cohorts of component A1 need to redistribute data to cohorts of component B1; similarly, cohorts of component B2 need to redistribute data to cohorts of component A2. To enable data redistribution between framework instances A and B (executing on M and N processors respectively) a third framework instance, C, containing the Seine coupling component is first instantiated. This framework executes on X processors, which may or may not overlap with the M and N processors of frameworks A and B. The Seine coupling component handles registrations and maintains the Seine directory. Frameworks A and B initiate Seine stubs, which register with the Seine coupling component. The ports defined by the Seine coupling and stub components are *register*, *put* and *get*. The operation is as follows. To achieve A1 - B1 coupling, component A1 registers its region of interest using the *register* port of its stub and invokes the *put* port to write data. Similarly component B1 independently registers its region of interest using the *register* port of its stub and invokes the *get* port. The *register* request is forwarded to the Seine coupling component, and if there is an overlap between their registered regions, at the *put* request from A1, the data written by A1 are directly and appropriately forwarded to Seine stubs at B1 by Seine stubs at A1. At the *get* request from B1, the data received by Seine stubs at B1 are copied from Seine's buffer. Note that A1 does not have to be aware of B1 or its data distribution.

Experiment with different data redistribution scenarios: In this experiment, a 3-dimensional array of size 120*120*120 is redistributed to 27 processors from 2, 4, 8, and 16 processors respectively, i.e., M = 2, 4, 8 and 16, and N = 27. Data in the array are of type double. The distribution of the array is (Block, Block, Block) [1] on the x-, y-, z-axis respectively on both the sender and receiver ends, except that for case I and II, (Block, Block, Collapsed) [2] is used at the sender end. The registration, data transfer, and send (put) and receive (get) costs are listed in Table 3. The application components do not explicitly compute communication schedules when using Seine. However, from their perspective, the time spent on computing communication schedule is equivalent to the time it takes to register their region of interest, i.e., the cost of the *register* operation, which is a one time cost. Since this cost depends on the region and its overlap, it will be different for different components, and the cost listed in Table 3 is the average cost. As the table shows, for a given total data size to be redistributed, the average registration cost decreases as the number of processors involved increases. The reason for this decrease is that as the number of processors involved in registration increases, each processor is assigned a correspondingly smaller portion of the array. As a result, each processor registers a smaller region. Since processing a *register* request involves computing intersections with registered regions, a smaller region will result in lower registration cost. In this experiment, as the number of processors increases, i.e., 2+27, 4+27, 8+27, and 16+27, and the size of the overall array remains constant, the sizes of regions registered by each processor decrease and the average registration cost decreases correspondingly. This experiment also measured the cost in data send (put) and receive (get) operations re-

---

[1]Block distribution along each dimension. This notation for distribution patterns is borrowed from High Performance Fortran and is described in [1].

[2]The first two dimension are distributed using a Block distribution and the third dimension is not distributed.

Table 3. Cost in seconds of computing registration and data transfer for different data redistribution scenarios.

Array size: 120x120x120
Data type: double
Distribution type (x-y-z):
   M side: block-block-collapsed (Cases I & II)
      or block-block-block (Cases III & IV)
   N side: block-block-block

MxN	Case I 2x27	Case II 4x27	Case III 8x27	Case IV 16x27
Registration	5.6725	3.6197	2.7962	2.273
Data transfer	0.6971	0.3381	0.1636	0.1045
M side (put)	0.6971	0.3381	0.1636	0.1045
N side (get)	0.0012	0.0012	0.0012	0.0012

spectively. The Seine model decouples sends and receives. In Seine, a push model is used to asynchronously propagate data. As a result, the cost of a data send consists of data marshalling, establishing a remote connection, and sending the data, while the cost of data receive consists of only a local memory copy from the Seine buffer. Consequently, the total data transfer cost is essentially the data send cost, and is relatively higher than the data receive cost. We explicitly list the data transfer cost in the table since this metric has been used to measure and report the performance of other coupling frameworks.

Experiment with different array sizes: In this experiment, the size of the array and consequently, the size of the data redistribution is varied. The distribution of the array is (Block, Block, Cyclic) on the x-, y-, z-axis respectively on both the sender and receiver ends. The registration, data transfer, and send (put) and receive (get) costs are listed in Table 4. As these measurements show, the registration cost increases with array size. As explained for the experiment above, this is because the registered regions of interest are correspondingly smaller and the computation of intersections is quicker for smaller array sizes. As expected, the costs for send and receive operations also increase with array size.

Scalability of the Seine directory layer: In this experiment, the number of processors over which the Seine coupling and parallel data redistribution component is distributed is varied. Distributing this component also distributes the registration process, and registrations for different regions can proceed in parallel. This leads to a reduction in registration times as seen in Table 5. The improvement, however, seems to saturate around 4 processors for this experiment, and the improvement from 4 to 8 processors is not significant. Note that the actual data transfer is directly be-

Table 4. Cost in seconds of registration and data transfer for different array sizes.

MxN: 16x27
Data type: double
Distribution(x-y-z): block-block-cyclic
   M side: number of cycles at z direction=3
   N side: number of cycles at z direction=4

Array size	60^3	120^3	180^3	240^3
Registration	0.0920	0.9989	6.31	9.987
Data transfer	0.0423	0.1117	0.271	0.8388
M side (put)	0.0423	0.1117	0.271	0.8388
N side (get)	0.0001	0.0008	0.004	0.0136

Table 5. Scalability of the directory layer of the Seine coupling component.

MxN: 16x27
Array size: 120x120x120
Data type: double
Distribution(x-y-z): block-block-block

Number of processors running Seine coupling component	1	4	8
Registration	4.2	2.273	2.089
Data transfer	0.112	0.1045	0.1172
M side (put)	0.112	0.1045	0.1172
N side (get)	0.0012	0.0012	0.0012

tween the processors and is not effected by the distribution of the Seine component, and remains almost constant.

From the evaluation presented above, we can see that the registration cost ranges from less than a second to a few seconds, and is the most expensive aspect of Seine's performance. However, registration is a one-time cost for each region, which can be used for multiple *get* and *put* operations. Note that registration cost also includes the cost of computing communication schedules, which do not have to be computed repeatedly. Data transfers take place directly between the source and destination processors using socket-based streaming, which does not incur significant overheads as demonstrated by the evaluation. Overall, we believe that the costs of Seine operations are reasonable and not significant when compared to per iteration computational time of the targeted scientific simulations.

5. Conclusion and Future Work

This paper presents the Seine framework data coupling and parallel data redistribution. Seine addresses the model/code coupling requirements of emerging scientific

and engineering simulations, enables efficient computation of communication schedules, supports low-overheads processor-to-processor data streaming, and provides high-level abstraction for application developers. Communications using Seine are decoupled and the communicating entities do not need to know about each other or about each other's distributions. A key component of Seine is a distributed directory layer that is constructed as a distributed hash table from the application domain and enables decentralized computation of communication schedules. A component-based prototype implementation of Seine using the CAFFEINE CCA framework, and its experimental evaluation are also presented. The Seine CCA coupling component enables coupling and parallel data redistribution between multiple CCA-based applications. The evaluation demonstrates the performance and low overheads of Seine.

This paper has addressed some of the core computational issues in enabling large-scale coupled simulations. However, there remain several issues that still need investigation. For example, optimization of memory usage during redistribution, especially when data sizes are large. Current research is looking at optimizing the Seine communication layer to use adaptive buffering and maximize utilization of available network bandwidth.

References

[1] Distributed Data Descriptor. http://www.cs.indiana.edu/ febertra/mxn/parallel-data/index.html

[2] J.A. Kohl, G.A. Geist. Monitoring and steering of large-scale distributed simulations, In IASTED International Conference on Applied Modeling and Simulation, Cairns, Queensland, Australia, September 1999.

[3] K. Keahey, P. Fasel, S. Mniszewski. PAWS: Collective interactions and data transfers. In Proceedings of the High Performance Distributed Computing Conference, San Francisco, CA, August 2001.

[4] J.-Y. Lee and A. Sussman. High performance communication between parallel programs. In proceedings of 2005 Joint Workshop on High-Performance Grid Computing and High Level parallel Programming Models (HIPS-HPGC 2005). IEEE Computer Society Press. Apr. 2005.

[5] CCA Forum. http://www.cca-forum.org

[6] S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. Divorcing language dependencies from a scientific software library. In Proceedings of the Eleventh SIAM Conference on Parallel Processing for Scientific Computing. SIAM, Mar. 2001.

[7] J.W. Larson, R.L. Jacob, I.T. Foster, and J. Guo. The Model Coupling Toolkit. In V.N.Alexandrov, J.J.Dongarra, B.A.Juliano, R.S.Renner, and C.J.K.Tan, editors, International Conference on Computational Science (ICCS) 2001, volume 2073 of Lecture Notes in Computer Science, pages 185-194, Berlin, 2001. Springer-Verlag.

[8] S. Zhou. Coupling earth system models: An ESMF-CCA prototype. http://webserv.gsfc.nasa.gov/ESS/esmf_tasc/, 2003.

[9] F. Bertrand and R. Bramley. DCA: A distributed CCA framework based on MPI. In Proceedings of HIPS 2004, the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments, Santa Fe, NM, April 2004. IEEE Press.

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

[11] K. Zhang, K. Damevski, V. Venkatachalapathy, and S. Parker. SCIRun2: A CCA framework for high perfomance computing. In Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), Santa Fe, NM, April 2004. IEEE Press.

[12] L. Zhang, M. Parashar: A Dynamic Geometry-Based Shared Space Interaction Framework for Parallel Scientific Applications. In Proceedings of High Performance Computing - HiPC 2004: 11th International Conference, Bangalore, India, December 19-22, 2004. Proceedings p.189-199.

[13] T. Bially. A class of dimension changing mapping and its application to bandwidth compression. PhD thesis, Polytechnic Institute of Brooklyn, June 1967.

[14] L.A. Drummond, J. Demmel, C.R. Mechoso, H. Robinson, K. Sklower, and J.A. Spahr. A data broker for distributed computing environments. In Proceedings of the International Conference on Computational Science, pages 31-40, 2001.

[15] M. Govindaraju, S. Krishnan, K. Chiu, A. Slominski, D. Gannon, and R. Bramley. XCAT 2.0: A component-based programming model for grid web services. Technical Report TR562, Department of Computer Science, Indiana University, June 2002.

[16] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, M. Parashar. High Performance Threaded Data Streaming for Large Scale Simulations. In the Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, November, 2004.

[17] I. Manuilskiy and W.W. Lee. Phys. Plasmas 7, 1381 (2000).

[18] Q. Lu, M. Peszynska, and M.F. Wheeler. A parallel multiblock black-oil model in multi-model implementation. In 2001 SPE Reservoir Simulation Symposium, Houston, Texas, 2001. SPE 66359.