Cpl6: The New Extensible, High-Performance Parallel Coupler for the Community Climate System Model

Anthony P. Craig[1], Robert L. Jacob[2], Brian Kauffman[1],
Tom Bettge[1], Jay Larson[2], Everest Ong[2],
Chris Ding[3], Yun He[3]

February 3, 2005

[1] *National Center for Atmospheric Research, Boulder, CO*
[2] *Argonne National Laboratory, Argonne, IL*
[3] *Lawrence Berkeley National Laboratory, Berkeley, CA*

Submitted to *International Journal for High Performance Computing Applications* 10/26/2004
Revised 02/03/2005

ABSTRACT

Coupled climate models are large, multiphysics applications designed to simulate the Earth's climate and predict the response of the climate to any changes in the forcing or boundary conditions. The Community Climate System Model (CCSM) is a widely used state-of-the-art climate model that has released several versions to the climate community over the past ten years. Like many climate models, CCSM employs a coupler, a functional unit that coordinates the exchange of data between parts of the climate system such as the atmosphere and ocean. This paper describes the new coupler, cpl6, contained in the latest version of CCSM, CCSM3. Cpl6 introduces distributed-memory parallelism to the coupler, a class library for important coupler functions, and a standardized interface for component models. Cpl6 is implemented entirely in Fortran90 and uses the Model Coupling Toolkit as the base for most of its classes. Cpl6 gives improved performance over previous versions and scales well on multiple platforms.

1. Introduction

Climate modeling is an example of a complex multiphysics application, and it is one of the primary high-performance computing (HPC) challenges. The Community Climate System Model (CCSM) is a state-of-the-art global climate model consisting of four fundamental physical components[1]: an atmosphere model, a land surface model, an ocean model, and a sea-ice model. Each typically contains both fluid-dynamics solvers and detailed parameterizations to compute the internal and external forcing terms that come from such diverse phenomena as the passage of radiation through the atmosphere, the release of latent heat by phase changes of water, and the effects of friction and unresolved turbulent scales. CCSM is a *coupled* model because data calculated in one model is used as boundary conditions and forcing in another. For example, wind calculated in the atmosphere model is passed to the ocean, land, and sea-ice models to serve as boundary conditions in those models. CCSM is used to understand the Earth's global climate system, to predict the effects of climate change, and to understand past climates.

Like many climate models, CCSM is not developed from scratch as a single application but is instead a *coupled* application where each component (atmosphere, ocean, land, and sea ice) is a highly complex application developed separately with its own coding style and datatypes. The uncoupled, or standalone, versions have their own user bases and scientific application areas and may be developed at multiple institutions.

[1] A "component" in this paper is a submodel of a coupled system and does not refer to a component programming model.

To couple models developed separately into a single application, CCSM has developed, over the past ten years, a framework that permits coupling with minimum modification to component models. This framework takes advantage of the fact that most component models need external surface forcing data. For example, an atmosphere model needs surface temperature, and an ocean model requires wind stress. In the CCSM framework, the external forcing routines are replaced with routines that communicate with a central coupler, an additional and separate component. The resulting CCSM architecture is a hub-and-spoke model as shown in Figure 1.

An aspect of the CCSM architecture is its multiple executable runtime system. Each component model in Figure 1 is a separate binary program that executes concurrently on disjoint sets of hardware processors. Each component is run on multiple processors using the Message Passing Interface (MPI, specifically MPI-1), OpenMP, or a combination of the two parallelization paradigms. (The multiple-executable architecture of CCSM means that a given vendor's implementation of `mpirun` must allow starting multiple executables under a single MPI communicator context.) The multiple-executable system also allows models to keep their own build systems and eliminates the need to resolve any data space name conflicts.

The central hub coupler, also a distinct program, and separate executables provide an extremely flexible architecture. This flexibility allows each submodel to be scaled individually to decrease the total time to solution of the system. Taking advantage of the physical properties of the climate system, communication with the coupler is carefully organized so that the models execute concurrently, thereby allowing for systematic load balancing. Individual models at the end of each spoke can be substituted for others independently and without requiring a recompile of the system as long as their inputs and outputs are consistent. (CCSM is distributed with "data" and "dead" versions of the atmosphere, land, sea-ice, and ocean models that can be substituted for the active components to facilitate both scientific investigation and software testing.) Furthermore, individual models continue to be usable as single executables in the subdisciplines that control their development, an important requirement in the climate modeling community. Maintaining coupled and "stand-alone" versions of the component models is straightforward because the interaction with the coupler is confined to a handful of calls to send and receive data and the overall structure of the component model, its `main`, is unaltered.

The coupler's primary role as the hub is to facilitate communication of data between components, including interpolation of data from one grid to another and the merging of data from multiple components while ensuring global conservation of physical properties such as heat, momentum, and mass. As the hub, the coupler isolates grids and coupling frequencies in the physical component models from each other. This feature is necessary because each model can be run at multiple horizontal and vertical resolutions. The CCSM coupler computes some fluxes and derives some fields between components that are more conveniently calculated in the coupler instead of in one of the physical system models. The coupler coordinates the coupling frequency and synchronizes the models, computes diagnostics, and writes output of data computed in the coupler.

The first released version of the CCSM coupler, cpl3 (Bryan *et al.*, 1996), was included in the first public release of CCSM (Boville and Gent, 1998) in June 1996. Cpl3 and the hub-and-spoke system had been used internally at the National Center for Atmospheric Research (NCAR) as early as 1994. The separate executable, hub-and-spoke approach of CCSM is one of many possible ways to construct a coupler for a coupled climate model. Other examples of couplers for coupled climate models include OASIS (Valcke *et al.*, 2004), the Flexible Modeling System (FMS; www.gfdl.gov/fms), GEMS (DAO, 1997), the Distributed Data Broker (Drummond *et al.*, 2001), and the coupler for the Parallel Climate Model (PCM) (Washington *et al.*, 2000; Bettge *et al.*, 2001).

CCSM originally ran on vector-based supercomputers and other single-system image platforms. Over the past several years, CCSM has migrated to distributed shared-memory machines such as the IBM SP, the HP/Compaq EVs, and Linux clusters, and all component models have been modified or replaced as needed with versions that use distributed-memory parallelism with MPI. CCSM2 (Kiehl and Gent, 2004) still contained a coupler, cpl5, that had only shared-memory parallelism using OpenMP. Although cpl5 was not a barrier to performance at the time of CCSM2's release, it was determined that improving coupler performance in the current system could result in gains of 5 to 10 percent in the throughput of the model at current production resolutions and configurations. Future development of CCSM envisions both increasing resolution and adding more physical processes (*Community Climate*

System Model Science Plan (2004-2008) www.ccsm.ucar.edu/management/sciplan2004-2008.pdf). With these plans and evolving hardware, cpl5 quickly could become a significant bottleneck in CCSM because of its inability to scale to more than one shared-memory node. A parallel coupler was necessary for CCSM's future development.

Key requirements identified for the new coupler included duplicating the functionality and abilities of cpl5, including support for active, data, and dead components while still imposing few or no requirements on the internal datatypes or external packages of the standalone versions of the component models. The new coupler was also required to address shortcomings in the extensibility of cpl5. This effort included simplifying and standardizing the process of adding new models and new fields to the coupled system and creating a uniform model-coupler interface. Although the CCSM framework successfully allowed the inclusion of new, externally developed sea ice and ocean models in CCSM2, the interface between all models and the coupler was a cpl5 method in some cases and a direct MPI call in others. The new coupler also needed to maintain or improve its performance relative to cpl5 and not negatively impact the throughput of the fully coupled system. Further, the new coupler had to have enough support to ensure development as CCSM continues to evolve and as it migrates to the latest high-performance computing platforms. Other parallel couplers such as the Distributed Data Broker (Drummond *et al.*, 2001) and the couplers in the Parallel Climate Model (Bettge *et al.*, 2001) and the Fast Ocean Atmosphere Model (Jacob *et al.*, 2001) could not meet all the requirements. In addition, packages such as Earth System Modeling Framework (ESMF; Hill *et al.* (2004) and the Program for Integrated Earth System Modeling (PRISM; prism.enes.org) were not yet mature enough for this project in its required time frame. Thus, a new coupler, cpl6, was created by a team of researchers already involved with the development of CCSM.

The new coupler in the latest version of CCSM, CCSM3 (Collins *et al.*, 2005), is called cpl6. This new coupler successfully introduces distributed-memory parallelism to the CCSM coupler, standardizes the model-coupler interface, and simplifies the processes of adding new fields to the coupled system data flow or new models to the coupled system.

The remainder of this paper is divided as follows. In Section 2, we describe the architecture of cpl6. Section 3 describes some of the major algorithms used in cpl6. Section 4 presents performance results of the coupler on some CCSM3-supported platforms. We conclude in Section 5 with a brief discussion of future work.

2. Cpl6 Design

The transition to distributed-memory parallelism in the CCSM coupler could not be done with a simple modification to cpl5. The major reason is that a data-parallel coupler introduces a decomposition of each model's grid in the coupler. In CCSM, the three-dimensional models exchange two-dimensional fields of various quantities (such as temperature, wind, and humidity) at shared horizontal surfaces. The parallel versions of these component models are typically decomposed over one or both of the horizontal dimensions. In cpl5, not only was the coupler running on only one MPI task, but the communication to components was handled by one MPI message to the components' root processor only. This approach meant additional overhead in components to gather or scatter data sent to or received from the coupler. One of the main considerations for cpl6 was to avoid the communication bottleneck presented in cpl5 when communicating with a model running on M MPI processes. As the hub of the system in CCSM3, cpl6 now runs on multiple processes, thereby introducing a decomposition of each model's grid in the coupler. This situation leads to multiple instances of the "MxN problem," namely, the transfer of a distributed data object from a module running on M processes to another running on N processes (see http://www.cs.indiana.edu/feberta/mxn for a summary). Cpl5 had no datatypes or methods to handle this capability. New coupler software was needed to introduce distributed-memory parallelism to the hub of CCSM. In this section, we describe the overall architecture of the new coupler and its supporting software.

2.1. Architecture

Cpl6 is written entirely in Fortran90. Fortran90 provides enough object-oriented features to build a class library. Since the physical models in CCSM3 are also written in Fortran90, cpl6 avoids language interoperability issues. Cpl6 was developed to mimic the functionality of cpl5 and its role in the CCSM coupled system while providing data parallel capabilities to the coupler hub and extensibility to the coupled system.

A schematic of the CCSM3 architecture with cpl6 is shown in Figure 2. At the bottom are system-level layers (IV) such as MPI and OpenMP. Layer

III contains two new libraries written in part to address the challenges of creating a distributed-memory coupler for CCSM. The Model Coupling Toolkit (MCT) was written to provide a general solution to the MxN problem and the other operations required in parallel coupled models and is described in two companion papers (Larson *et al.*, 2005; Jacob *et al.*, 2005). MCT is the foundation layer for most of the cpl6 classes as described below, and all the models in CCSM ultimately use MCT to transfer data to and from the coupler. An additional library in Layer III created in response to the needs of cpl6 is MPH (Multi Program-component Handshaking). MPH provides a general method for configuring models in MPI processor space. A runtime component name registration file is used to specify the components and their processor allocations. MPH provides methods for each model to obtain a unique communicator and to locate another model's MPI processes. MPH contains other features and supports other coupled model configurations. MPH is described more fully in He and Ding (2005).

This paper focuses primarily on the cpl6 design and implementation (layers I and II in Figure 2). Layer I contains the individual models of CCSM and the coupler `main` program (the hub in Fig. 1). Layer II contains the new classes of cpl6 described in more detail below. "CPL6 main" in Fig. 2 is a new hub program written with the datatypes and methods of Layer II. Unlike the general datatypes of the cpl6 library (Layer II), the cpl6 `main` encodes a particular set of scientific choices that make up CCSM3, including the execution sequence, coupling frequency, supported grids, number of components, and some scientific calculations. The cpl6 `main` is a complex but specific application itself, and additional classes were created to aid in its implementation. Additional classes used only by the cpl6 `main` are considered part of Layer I ("CPL6 main datatypes" in Fig. 2); some of these are described in Section 3.

2.2. Cpl6 Software

Cpl6's major data structures are Fortran90 derived datatypes. Most datatypes consist of a combination of scalars and arrays of integers, characters, reals, and other derived datatypes, including some from MCT. One heavily used datatype from MCT is the AttributeVector.[2] This datatype serves as the primary distributed storage datatype for local real and integer data. Although MCT could have been used directly for model-coupler communication, it would have required adding several MCT datatypes and calls to each component model. Thus, cpl6 was designed to wrap many of the MCT routines and datatypes into simpler routines and more compact datatypes.

The cpl6 datatypes and methods that act on them are grouped into Fortran90 *modules*. At the highest level, cpl6 has a handful of important datatypes including domain, bundle, infobuffer, map, and contract. It also has several features that simplify model coupling, including fields and interface modules. Figure 3 shows the relationships between the most important data objects in cpl6 and MCT. The first two levels in Fig. 3 ("CPL6 Model Inter-

[2] We shall use the following typographic conventions: references to class or Fortran90 module names are indicated with classname. File names, subroutine names, and other parts of source code are indicated as `subroutine`. MCT datatypes use a MixedCase naming scheme. Cpl6 datatypes are all lowercase.

face" and "CPL6 Internal") are all contained within the cpl6 datatypes and methods (layer II in Fig. 2).

In general, everything at the MCT/MPH layer (layer III of Fig. 2) and below is treated like an external and frozen library in CCSM3, while everything in the higher layers (layers I and II) is considered part of the CCSM3 source code and may be modified by users. (Like many other scientific applications, "using" a climate model implies editing source code in many cases.) In practice, however, only a very limited amount of code in the cpl6 datatypes and methods layer (layer II) needs to be modified even when building new coupled systems.

In the discussion below, we consider three types of users of cpl6. The most common user request is to modify the number and identity of fields passed between the coupler and other models. A system integrator is interested in replacing one of the model's in CCSM with a new model. Finally, the coupler writer needs to either substantially modify or create a new coupler `main` program to make a major change such as adding a model, changing the model integration order, or changing the number of grids supported. Cpl6 provides new datatypes and methods for each class of users.

2.2.1) Fields

The user who wants to change the fields exchanged with the coupler needs only to edit the fields module to make the coupler aware of the change. The coupled data flow between models and the coupler in CCSM is fixed at compile time, and matching inputs to outputs is done by the scientists and programmers constructing the coupled system (CCSM does not do data "brokering"). The fields module contains a master list of the names of all scalar and gridded data fields transferred between the coupler and component models in the form of colon-delimited Fortran character strings. The coupler and each component model share this data through Fortran90 `use` association of this module. The number of tokens in each string determines the amount of local storage needed to hold data involved in coupling. Localizing this information in one module and requiring all component model subroutines that communicate with the coupler to use it have improved the robustness, flexibility, and extensibility of the coupled system significantly compared to cpl5 and meet an important goal of cpl6. It is now relatively trivial to add or remove coupling fields in the system.

2.2.2) Interface

A fundamental design goal of the cpl6 architecture was to abstract the coupling layer away from the components and present a simple interface to the components. System integrator whos want to replace a model on the end of a spoke in CCSM (Fig. 1) need only to familiarize themselves with the routines in the interface module. The interface module contains all the routines that component models use to interact with the coupler. The interface routines use the contract (Sec. 2.2.7), which is an argument in nearly all the interface routines. The system integrator would need to add a `use` statement for `cpl_interface_mod` in the routines where data is exchanged with the coupler. (In CCSM3 components, these routines are usually collected in a single module.) A `use` statement for the contract module is also required but only so instances of the datatype can be declared; the user does not need to know its contents or its methods.

The purpose of the interface module is to provide a simple, compact interface for component models to talk to the coupler while making no constraints on how the component models represent data internally. Four methods from `cpl_interface_mod.F90` do nearly all the work in allowing a component model to interact with the coupler:

`cpl_interface_init` Initialize the communication infrastructure, e.g., MPI communicator groups.

`cpl_interface_contractInit` Initialize a communication contract. This is where grid information is initially exchanged and communication methods are established.

`cpl_interface_contractSend` Send data (bundles and infobuffers via the contract) to another component.

`cpl_interface_contractRecv` Receive data (bundles and infobuffers via the contract) from another component.

Along with reducing the needed routines to a handful, the interface module also simplifies the argument list. Aside from the contract, which is a Fortran90 derived datatype, the remaining arguments in `cpl_interface_` methods are simple native Fortran (90 or 77) types such as real and integer scalars and arrays. This makes cpl6 significantly easier to incorporate into an existing model. These simple arrays are coordinated through the use of integer indices defined in the fields module as discussed in Section 2.2.1. The interface also makes it easy to change the underlying coupling strategy with limited impact on the components.

By making no assumptions on the relation between coupler datatypes and model datatypes, data between the model's internal data structures and the arguments to `cpl_interface_*` can be moved with a copy in the coupler interface. Since the models in CCSM occupy different processors and different memory spaces, the data buffers passed to the interface send and receive functions can be safely used as soon as the subroutines return.

The fields and interface datatypes are also used by the coupler itself. The remaining datatypes and methods of cpl6 are either used internally by the interface routines or used directly only by the coupler writer. They are not intended for average users of CCSM.

2.2.3) Domain

One of the fundamental datatypes in cpl6 is the domain. The domain contains information about the physical grid on which a quantity, such as temperature, is defined. This includes a descriptive name for the grid, the total number of points, and the number of points in each horizontal dimension. (Since all the fields exchanged by the coupler in CCSM3 are two dimensional, the domain currently supports only two-dimensional grids.) The domain also contains information about how the global grid is decomposed over processors. This information is stored within an MCT datatype called the GlobalSegmentMap. The GlobalSegmentMap information is identical on each coupler processor, so each processor knows the entire decomposition of a grid in the coupler. The GlobalSegmentMap is a compact description of the grid, requiring less memory than the total number of points.

The domain also includes numerical data about the grid, such as the latitude and longitude values and grid-cell area, but only for points local to the processor. The contents of this part of the domain datatype thus varies from processor to processor, and its memory size decreases as the number of processors increases. The values are stored in a locally sized MCT AttributeVector datatype. In general, one domain is instantiated for each numerical grid.

2.2.4) Bundle

Another fundamental data structure in the cpl6 infrastructure is the bundle. The bundle is the basic cpl6 parallel data storage type and is used throughout cpl6 to store and manipulate gridded data such as temperature. The bundle datatype has a name attribute, contains an MCT AttributeVector to store multiple fields, and has a pointer to a domain. All fields in a bundle must share the same domain (i.e., grid and decomposition). The bundle also has an integer element that serves as a counter for accumulating and averaging values in bundles. Like the domain, the bundle is locally sized, and its memory usage is proportional to the number of grid points on a processor and the total number of fields. The bundle module has several methods associated with it, including initialization, zero, accumulate, average, add, multiply, divide, sum, minmax, fill, print information, clean, test, dump, and copy. These methods provide all the functionality required to perform important coupler functions such as merge, diagnose, accumulate, and average fields in a bundle.

2.2.5) InfoBuffer

While the bundle is for gridded data, the infobuffer is used to hold scalar integer and real data exchanged between component models and the coupler. The integer data sometimes acts as logical control flags for communicating actions such as writing a restart file, computing diagnostics, or halting execution. The infobuffer is also used to pass real scalars between the models for CCSM scientific calculations. The infobuffer is the same size on all processors but is very small, on the order of 1 KB.

2.2.6) Map

The map datatype in cpl6 contains all the information required to carry out mapping (interpolation) in the coupler. Information contained in the map datatype are initialized at runtime, partially from data received by the coupler, and partially from data read from external files. The map datatype contains two domain pointers for the source and destination grids, an MCT SparseMatrix, a name, an intermediate grid, and an MCT Rearranger to handle mapping communication. The cpl6 map module contains initialize, clean, print information, and map methods. The mapping algorithm is discussed further in Section 3.4. The map is also locally sized, containing just enough space to hold the matrix weights for the points on a processor.

2.2.7) Contract

The contract is a key concept and an important datatype in cpl6. All components in CCSM3 have access to the contract module. A contract contains all of the information needed for a single model-coupler exchange of data. Contracts are initialized at model startup and then are used to facilitate all model-coupler communication. The main interaction between a model and the coupler is conceptualized as initializing and then using contracts. Section 3.3 describes the initialization step and communication method.

The contract contains the cpl6 datatypes infobuffer, bundle, and domain. The contract also contains an MCT datatype called the Router, which has all the information needed to do a parallel data transfer between a distributed-memory parallel model running on one set of processors and the distributed memory parallel coupler running on a different set of processors. Except for the small infobuffer, the other major datatypes are locally sized, and their memory usage is proportional to the number of local grid points times the number of variables involved in coupling.

The contract is the only cpl6 Fortran90 derived datatype required to be instantiated by the components, but all interactions with the contract datatype are handled through a handful of calls to the interface routines (Section 2.2.2). The contract allows flexibility and extensibility into the future for communication patterns other than the hub-and-spoke method; if the communication needs to be changed, the contract can be redefined and the coupler-model interface left unchanged.

Multiple contracts can exist between a model and the coupler in cpl6. In

CCSM3, the land surface model includes an embedded surface runoff model that runs on a different grid. The land surface model has a contract that communicates land model data to and from the coupler as well as a contract for communicating runoff data on the runoff grid to the coupler at a different coupling frequency.

3. Cpl6 Methods and Algorithms

This section provides more detail about some of the important methods and algorithms used in cpl6. In CCSM, important physical computations are performed in the coupler `main` and have an impact on the coupler's integration time. This section also describes these calculations and the reason for including them in the coupler.

3.1. Flux Computation

Some computations of physical quantities are performed in the coupler. By convention, fluxes between two models with different resolutions are calculated on the grid with the higher resolution. Consider the flux of heat between the ocean and atmosphere. If the atmosphere model were to calculate the fluxes, it would need both to interpolate its data to the ocean grid and to receive ocean data. In the CCSM3 instantiation of a coupled climate model, the ocean communicates with the coupler only once per day, which is not frequent enough for the scientific requirements of the heat flux calculation. Thus, the coupler calculates the fluxes of heat as well as momentum and freshwater between the atmosphere and ocean, and cpl6 includes flux calculation methods. The flux routines take bundles as arguments, and, in

CCSM3, ocean-resolution bundles are used. The flux routines are part of the software provided for the coupler `main` writer and are located in layer I of the CCSM architecture.

The flux computations in the coupler are carried out on a grid point basis without any requirements for neighbor communication or global sums. This operation is a trivially parallel computation but is one of the main sources of floating-point operations in the coupler outside of the mapping routines.

3.2. Merging

Merging is the relatively straightforward task of combining different fields into one field. For example, the surface under an atmosphere grid cell may be part land, part ocean, and part sea ice. In this case, forming the net surface heat flux into the atmosphere will require merging surface heat flux values from three sources: atmosphere-land fluxes from the land model, atmosphere-ice fluxes from the sea ice model, and atmosphere-ocean fluxes calculated in the coupler. This merging process results in a combination of multiply and add operations across gridpoints,

$$F = w_1 F_1 + w_2 F_2 + w_3 F_3, \tag{1}$$

where F is the merged field, w_1, w_2, and w_3 are weights (typically $w_1 + w_2 + w_3 = 1$), and F_1, F_2, and F_3 are source fields. Merging in cpl6 is also necessary for fields sent to the ocean model where sea ice and atmosphere coupling fields need to be combined. Once the fields are mapped onto the appropriate grid, this operation requires no communication between processors. This operation was parallelized in cpl5 via OpenMP directives, and it is parallelized in cpl6 by memory decomposition.

The merge subroutine is part of the cpl6 `main` only. Initially, this procedure was coded by using calls to the multiply, add, and accumulate methods from the bundle module, resulting in sequential operations on individual terms as represented by the following pseudocode:

```
M1(:) = 0.
M1(:) = M1(:) + w1(:)*F1(:)
M1(:) = M1(:) + w2(:)*F2(:)
M1(:) = M1(:) + w3(:)*F3(:)
M2(:) = 0.
M2(:) = M2(:) + w1(:)*F4(:)
M2(:) = M2(:) + w2(:)*F5(:)
M2(:) = M2(:) + w3(:)*F6(:)
```

Unfortunately, this pseudocode performs poorly because of poor cache usage. Each statement is executed for all grid points before moving to the next statement. As a result, cache reuse of $w1$, $w2$, and $w3$ is particularly poor. In addition, M1 and M2 are loaded three times and saved four times for each three-way merge. In the end, merges done this way were a bottleneck in the cpl6 coupler and had to be hand-coded in the merge subroutine in cpl6 `main` as follows:

```
do n=1,npoints
   M1(n) = w1(n)*F1(n)+w2(n)*F2(n)+w3(n)*F3(n)
```

```
    M2(n) = w1(n)*F4(n)+w2(n)*F5(n)+w3(n)*F6(n)
 enddo
```

This coding style is much less general but results in performance gains of factors of 2 to 4 over the more general method.

3.3. Communication

In previous versions of CCSM, root-to-root communication between the coupler and distributed-memory components was used to exchange data. This method required a gather to root before a send and a scatter from root after a receive for the MPI parallel models. In this case, the communication cost was at best constant with an increase in processor counts in distributed-memory components. This method also did not scale well to larger processor counts or higher resolutions.

Cpl6 uses MCT's "MxN communication" scheme (Jacob *et al.*, 2005) to move data between the coupler and the component models in the routines `contract_interface_Send` and `_Recv` (Section 2.2.2). MCT derives a set of point-to-point communications between the M and N processors that collectively transfer all the targeted data from the memory space of the M processors to the memory space of the N processors. One of the important aspects of this scheme is to send as few messages as possible in order to lower latency costs. Usually, ten to twenty fields are exchanged between the coupler and a component at any point during integration, so the fields are packed together to minimize the number of messages and maximize performance. Cpl6 uses the blocking versions of the MCT routines because the location of the data transfer calls in the models and the science-based data dependencies between models require them to complete before computation can proceed. Nonblocking versions are available and could be used to overlay communication with computation in a future version of CCSM.

An important method in cpl6 is the initialization of communication schedules for the MxN transfer performed by the call to `cpl_interface_contractInit`. Because each component in CCSM may be running on different sets of processors with each invocation, the schedules, stored in an MCT Router, need to be computed for each CCSM run. However, since the grids and decompositions in CCSM are fixed for each job submission, the schedules need to be computed only once at initialization.

A particular sequence of calls is required to initialize model-coupler communication, as illustrated in Figure 4. In the current version of CCSM, a unique contract exists for every communication pair, and there is a designated sender and receiver as well as a designated component that leads the definition of the grid at initialization. In cpl6, contracts are set up for both sending to other components and receiving from other components. In all CCSM3 contracts, the physical model is the "lead" and informs the coupler of its grid and decomposition during the initialization phase of runtime. The coupler then uses this information to construct its own decomposition of that model's grid and build a communication schedule, the MCT Router, between the coupler and the model. Contract initialization requires both one-way and two-way communication between components and hence must be coordinated carefully in the system to avoid deadlocks. Cpl6 (and MCT) dynamically al-locate all datatypes needed for communication at runtime, which gives the new coupler significant flexibility to handle new grids or processor counts. The contract initialization, including the Router initialization, is a small fraction of the overall initialization time in CCSM, which, in turn, is a small part of the total runtime for a typical CCSM integration. Details on the Router initialization time can be found in Jacob, Larson, and Ong (2005).

3.4. Mapping

Mapping (or interpolation) in CCSM is the operation of transforming data from one grid to another. In CCSM, interpolation is implemented as a matrix-vector multiply where the input and output vectors, also referred to as "source" and "destination" vectors, are the size of the two global grids and the mapping weights constitute the sparse matrix. For mapping atmosphere data to the ocean grid, the equation is

$$\overbrace{\begin{pmatrix} o_1 & o_2 & \dots & o_m \end{pmatrix}}^{m\ ocean\ grid\ points} = \overbrace{\begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix}}^{n\ atmosphere\ grid\ points} \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{pmatrix}. \qquad (2)$$

The matrix **A** contains all of the atmosphere grid points in a 2D horizontal plane unrolled into a single vector, while the matrix **O** contains all the points in a 2D horizontal slice of the ocean grid. For a T42 atmosphere and "x1" ocean grid, the matrix **W** would contain $(64 \times 128) = 8192$ rows and $(320 \times 384) = 122800$ columns. Fortunately, most of the elements of **W** are zero, making this a *sparse matrix-vector multiply.* The map datatype holds only the nonzero elements of $\mathbf{W}$ in an MCT SparseMatrix datatype. The mapping method for bundles uses MCT methods to perform the mapping for all the fields in a bundle at once. Mapping multiple fields in parallel is a significant performance benefit.

This mapping algorithm supports all linear interpolation schemes because $\mathbf{W}$ can contain any arbitrary values. In CCSM, both bilinear interpolation and first-order conservative (Jones, 1999) mapping is done, and weights are generated off-line by using the *Spherical Coordinate Remapping and Interpolation Package* (SCRIP; Jones (1998)).

Because cpl6 is a distributed-memory component, the decomposition across processors of the source values, destination values, and sparse matrix weights can have a large impact on performance. In general, some data rearrangement on either the source or destination grid will always be needed to complete the calculation. In CCSM, the mapping module supports two specific mapping algorithms. The first is *source mapping*, where the mapping weights associated with the source values are distributed in a way that allows the destination values to be computed locally. The redistribution is based only on the sparse matrix weights and decomposition of the destination grid. In the second algorithm, known as *destination mapping*, the mapping weights are distributed according to the source grid's decomposition. Source grid data is left on-processor, and the sparse matrix multiply results in partial sums of the final destination fields. The final results are obtained by rearranging the partial sums into the destination grid's decomposition and summing. This approach minimizes communication of the source fields but can lead to slight differences in answers based on the number of coupler processors. A runtime option is available in cpl6 to produce bit-reproducing answers by specifying that all mappings be done with the source mapping algorithm.

The mapping algorithm choice in CCSM depends mostly on the relative sizes of the grids and the amount of data transfer required. Experience with the Parallel Climate Model (PCM) (Washington *et al.*, 2000; Bettge *et al.*, 2001), where these algorithms were first implemented, showed a performance advantage in leaving the source data on-processor if the source grid is larger than the destination grid, as is the case in ocean to atmosphere mapping in typical CCSM resolutions. Hence, in CCSM, if the source grid has fewer gridpoints, the *source* algorithm is used, whereas if the destination grid has fewer gridpoints, the *destination* algorithm is used. The penalty associated with choosing the wrong algorithm in CCSM is about a factor of 3 in map timings for typical CCSM resolutions and decompositions. One can envision other algorithms that perfectly load balance the sparse matrix multiply or that minimize data transfer, but these have not yet been implemented in CCSM. MCT provides this flexibility in a datatype similar to map called the SparseMatrixPlus (Larson *et al.*, 2005).

All aspects of mapping, including weights distribution, algorithm, and communication requirements, are set at initialization in CCSM and stored in the map datatype. The MCT datatypes and methods used to support these operations are described in more detail in the companion papers (Jacob *et al.*, 2005; Larson *et al.*, 2005). Optional arguments are provided in the map methods to facilitate area fraction normalization. This approach is used for ocean to atmosphere conservative mapping to increase the mapping accuracy for fields where area fractions are constantly changing, as in the case of evolving sea ice.

4. Results

We investigated the scaling performance of important cpl6 routines in realistic configurations of the CCSM3 system and on platforms where CCSM3 was already ported. The first platform was an IBM p690 called "Bluesky." Bluesky is located at NCAR and is the main development platform for CCSM. Bluesky consists of 1600 1.3 GHz Power4 processors with 2 GB of memory per processor. Processors on Bluesky are grouped into 8-way and 32-way shared-memory nodes, but 8-way is the default choice for CCSM3 and is used in most of the timings below. The interconnect fabric for Bluesky is the IBM "Colony" switch with one communication channel per node (i.e., single rail). The second platform was an HP Alpha Cluster called "Lemieux," located at the Pittsburgh Supercomputing Center. Lemieux comprises 750 HP/Compaq Alphaserver ES45 nodes. Each computation node contains four 1 GHz processors with 4 GB of shared memory. The nodes are connected with a Quadrics network. The third platform was a Linux cluster called "Jazz," located at Argonne National Laboratory. Jazz contains 350 nodes, each with a single 2.4 GHz Pentium Xeon processor and either 2 GB or 1 GB of RAM. The processors are connected via Myrinet 2000. Vendor Fortran compilers were used on all platforms except Jazz which used Portland Group Fortran.

For all the timings below, a “dead-model” configuration of CCSM was used. In this configuration, there are still five separate executables, but the active physical components (atmosphere, ocean, sea-ice, and land) are replaced with “dead” versions that contain no numerics. They do not use received data and send artificial data but still run on multiple processors with decompositions and grids consistent with the active models. Since we are interested in timing only the performance of the coupler and the communication from components, this is an adequate test configuration. The coupler is the same in all configurations of CCSM. The freely available version of CCSM3 source was used in all cases, and any code modifications for timing are noted. The CCSM3 build and runtime scripts were also used. The integration length in all cases was 10 simulated days. The coupler has a one-hour timestep, so all results are the aggregate time for 240 calls to the various cpl6 routines. In all tests, we used the default resolutions for the various components. For the atmosphere and land models, the horizontal grid involved in coupling is the “T42” grid and contains 128 longitude and 64 latitude points (8,192 total) covering the globe. For the ocean and sea-ice models, the “x1” (“by one”) grid contains 384 latitude and 320 longitude points (122,880 total).

In all figures, cpl6 subroutine scaling is shown from 1 to 32 processors. Thirty-two processors is well above the default value for the coupler on the platforms and resolutions tested. The default number of processors for the coupler is 2 on Jazz and 8 on Bluesky and Lemieux. Default values are chosen in consideration of the load balance of the full CCSM system. The number of nodes allocated to the coupler varies for each machine. On Jazz, there are always one node and one communication channel for each processor. On Lemieux and Bluesky, the number of nodes can be obtained by dividing the number of coupler processors by the number of processors per node for each machine, 4 and 8, respectively.

The purpose of this section is to compare the relative scaling performance of cpl6 implementations of time-critical sections on possible production platforms for CCSM3 and demonstrate that adding distributed-memory parallelism to the coupler can reduce the time to solution in the routines considered. However, we do attempt to explain any intermachine performance differences, where encountered.

Direct comparisons between cpl5 and cpl6 are limited. The reason is partly that the version of CCSM with cpl5 does not have the dead model capability and is not ported to as many platforms. Performance of cpl5 compared to cpl6 is noted where known and direct comparisons were obtained from a beta version of cpl6 measured against cpl5 on a fourth platform, called Blackforest. Blackforest is an IBM Power3, also located at NCAR, and consists of 1172 375 MHz Power3 processors. The processors are grouped into 4-way nodes each with 2 GB of memory, and the interconnect fabric is the TMBX switch with two communication channels per node (i.e., dual rail).

4.1. Flux Calculation

The atmosphere-ocean flux calculation in the CCSM coupler is a compute-only section consisting of floating-point operations and intrinsic calls to functions such as `log` and `exp`. The calculation is computed on the ocean grid.

In cpl5, the flux calculation was parallelized by using OpenMP directives. In cpl6, the flux computation is done on multiple processors by using a decomposition that results in an equal number of grid points on each processor. However, the calculation is not carried out over land points, which typically leads to some load imbalance.

Performance of the flux computation routine, `flux_atmOcn`, in the coupler is shown in Figure 5 for the three platforms, with processor counts ranging from 1 to 32. These timings were extracted from a full dead-model run of CCSM, not just a test of this kernel. An MPI barrier call was placed before the subroutine call to eliminate load imbalance and to time only the cost of the call on each processor. Timing values were collected from all coupler processes; the maximum is plotted in Figure 5. Decompositions of the ocean grid on each machine are identical for each processor count.

The routine generally scales well to 32 processors. It is unclear at this point what is causing the relatively poor scaling of the flux calculation on the IBM between 2 and 8 processors, whether it is related to memory or cache performance, performance of intrinsics, or performance related to floating-point operations.

In general, the performance of the flux calculation in cpl6 is greater than or equal to that in cpl5 when the number of MPI processes in cpl6 is equal to the number of OpenMP threads in cpl5. However, the memory decomposition capability of cpl6 means that the total time can be greatly reduced on platforms with small numbers of processors per node. The cpl5 time for this calculation would be limited at best to the 1-processor cpl6 timing on Jazz, the 4-processor cpl6 timing on Lemieux, and the 8-processor cpl6 timing on Bluesky.

4.2. Merging

The performance of the two-way ocean merge routine merge_ocn in cpl6 is shown in Figure 6 using the maximum time over all processors. The two-way merge of atmosphere and sea-ice data is also a compute-only section of the coupler, consisting of memory access and two mult/add operations per merge field. In this case, about 15 distinct two-dimensional fields are merged. Like the flux calculation, the merge is performed on the ocean grid, and an MPI barrier call is added before the merge subroutine call to reduce the effect of load imbalance and to time only the cost of the subroutines.

As discussed in Section 3.2, the merge operation was hand-tuned for performance. Cache reuse, masking, and the use of scalar registers result in a performance gain over the bundle methods of about a factor of 2 to 4 for the IBM p690 system compared to use of more generic bundle methods. The bundle methods are more desirable because they hide overhead and complexity associated with the manipulation of cpl6 datatypes, and they make the code more readable. However, the performance degradation for this operation is unacceptable for the overall time to solution of the coupled model. This is an interesting example of a simple set of operations that are difficult to generalize by using simple datatypes and methods while still retaining high performance. The coding and memory access patterns still have a large impact on performance for many machines. This experience shows that a common infrastructure approach to simplifying a complex application code like the coupler can have hidden costs and penalties.

As shown in Figure 6, scaling of the Linux cluster to 32 processors is near-linear. Scaling of the routine on the HP is excellent while scaling on the IBM is unexpectedly poor from 1 to 8 processors. This poor scaling is likely related to saturation of memory bandwidth on the node. Below 8 processors, the node has idle processors and performs well. At 8 processors, all processors on the node are in use, and the memory subsystem is unable to scale. At 16 and 32 processors, multiple nodes are in use, the memory footprint of the calculation for each node decreases, and there seems to be some superscaling between 16 and 32 processors on the IBM, probably related to cache efficiencies.

The performance of the merge operation in cpl6 is improved by about a factor of 2 compared to cpl5, most likely because the OpenMP overhead is removed and the cache usage is better. For this computation, the performance is determined largely by the performance of the memory subsystem, including cache. Again, parallelism in cpl6 can reduce the cost of this calculation beyond what was possible in the shared-memory-only cpl5.

4.3. Communication

Communication scaling performance between the CCSM3 coupler, cpl6, and the CCSM3 ice model is shown in Figure 7. The minimum time over all processors for the coupler to complete the receive of data from the sea-ice model is shown. The timers are placed around the underlying MCT_recv call inside the cpl6 `cpl_interface_contractRecv` routine. The `MCT_recv` call on a processor includes several MPI calls, one for each ice processor that must send data to a given coupler processor. A total of 22 two-dimensional fields on the ocean/sea-ice grid are communicated. This communication is chosen for study because it is the most expensive in the CCSM3 system as a result of the relatively large number of grid points and fields and the relatively high frequency of communication, once each coupler time step. Another challenge associated with this communication is that the decompositions of the ice grid in the coupler and in the ice model are nearly orthogonal, so the total number of messages passed between the coupler and ice model components is typically the maximum number possible (i.e., the product of the number of ice and coupler processors). As indicated in Section 3.3, the contracts, including Routers, are set up during initialization of a given CCSM3 integration. During runtime, communication occurs with little overhead except for the packing of data from multiple fields (temperature, wind, etc.) into a single message.

In Figure 7, several different configurations are shown. Processor counts for the coupler vary between 1 and 32. The ice model is allocated 16 processors in all cases. In all cases except as noted below, the coupler and ice model are running on different nodes of the hardware, so timings are always associated with "off-node" communication cost.

Communication scales poorly on the IBM p690 from 1 to 8 processors. This is not caused by the software but instead by the ratio of processors to communication channels on the IBM. The coupler is on one 8-way node and is using all the off-node bandwidth available to the node regardless of decomposition or processor use. Above 8 processors, communication scaling improves dramatically. Scaling is nearly linear from 16 to 32 processors because the number of communication channels doubles (from 2 to 4) as the number of processors doubles above 8 processors in this case. As processors are added to the coupler, the size of the messages decrease, while the total number of messages passed to the coupler increases.

In order to gain further insight into the communication costs on the IBM p690 and their effect on cpl6 communication routines, the ice-coupler communication was also timed by using the 32-way nodes of Bluesky. For the coupler on 1 to 16 processors, the ice model and coupler are located on the same node. Communication in this case is via shared-memory operations in the IBM MPI library and results in the best performance and scaling of all platforms tested. Allocating 32 processors for the coupler, with the ice model still fixed at 16, requires a second 32-way node and forces communication off-node again. The communication performance for this case is an order of magnitude slower than the 16-processor, one-node coupler-ice model configuration and is only as fast as 4 and 8 coupler processor timings on other platforms. This degradation in communication performance is caused by both the doubling of the total number of messages and the communication now occurring entirely between nodes, compared to on-node communication of the 16-ice, 16-coupler case.

Figure 7 shows a direct comparison of communication cost of cpl6 vs. cpl5 on Blackforest. Although cpl5 used 4 OpenMP threads on a Blackforest node, the MPI communication was single-threaded. Figure 7 shows that cpl6 is about 20% faster then cpl5 on one processor and that the communication cost can be reduced by adding processors to the coupler.

The communication performance of IBM p690 "on-node" is a factor of 5 to 10 times better than the comparable IBM p690 "off-node" performance. This is a unique case and shows just how much communication penalty is incurred on the IBM for going "off-node." The CCSM makes an effort to place the coupler and ice model on the same nodes on the IBM whenever possible to take advantage of the on-node communication performance. Comparison of the IBM Power3 and p690 timings in Figure 7 show that there is little difference in the communication performance of these two machines for this case. The Power3 is, in fact, faster than the p690 at 8 processors because communication is through one channel on the 8-way p690 but two channels on the 4-way Power3 nodes.

Communication performance on the HP and 8-way IBM p690 is roughly comparable, but scaling is poor compared to the compute-only sections of the coupler discussed above. This poor scaling is a result of the balance between processors and communication channels. On Jazz, which maintains one communication channel per processor at all coupler processor counts, the scaling is very good.

4.4. Mapping

Figures 8, 9, and 10 show mapping performance for the cpl6 routine `cpl_map_bun` for three different cases. The first case (Fig. 8) is bilinear mapping of 9 fields from a T42 atmosphere grid to a "x1" ocean grid. The second case (Fig. 9) is a conservative mapping of 10 fields from a T42 atmosphere grid to a "x1" ocean grid. The third case (Fig. 10) is a conservative mapping of 13 fields from a "x1" ocean grid to a T42 atmosphere grid. The atmosphere-to-ocean mappings use the source mapping algorithm, which executes a redistribution communication of the source fields prior to a sparse matrix multiply (Section 3.4). The ocean to atmosphere mapping uses the destination mapping algorithm and executes a sparse matrix multiply followed by a redistribution of the destination fields and a local sum. The ocean-to-atmosphere mapping requires a normalization step, which means there is an additional premapping multiply of all ocean fields by the local area fraction followed by a postmapping divide of all atmosphere fields by the local area fraction.

As discussed in Section 3.4, the cpl6 mapping algorithm utilizes the interpolation capability in MCT, leveraging AttributeVectors in bundles, field indexing, and cache reuse. Communication in the redistribution is also handled by MCT. Other computations necessary in CCSM are performed with cpl6 bundle methods.

In spite of the differences in the three cases of Figures 8, 9, and 10, many common conclusions can be made. Mapping scales extremely well on the Linux cluster across all processor counts. The map routine scaling on the HP and IBM systems is good except between 4 and 8 processors on the IBM.

All of the mapping scaling curves resemble the flux results in Figure 5 more than the communication results in Figure 7. These results suggest that mapping cost is dominated by floating-point operations instead of communication. We made a closer examination of the components of the `cpl_map_bun` call for the 2 and 16-processor case on Jazz. At 2 processors, the communication costs associated with redistribution of points represent only 2% to 4%

of the total cost for all cases. At 16 processors, the redistribution cost is still only 8% for the bilinear case and around 20% for the conservative mappings. The reason for this difference is that the bilinear mapping has over twice as many nonzero matrix elements as does the first-order conservative mapping and so has more floating point work. This is also the reason that, at all processor counts, the bilinear mapping in Figure 8 is about 50% slower than the conservative mappings.

Comparing Figures 9 and 10, one sees that the conservative ocean-to-atmosphere mapping is about 2 to 5 times slower than the conservative atmosphere-to-ocean mapping. Part of this can be explained by the number of fields mapped in the timings (10 vs 13). However, the most important difference in these two calculations is the extra normalization steps required for ocean-to-atmosphere mapping. This normalization improves the accuracy of the mapping but results in a significant cost increase in the conservative ocean-to-atmosphere mapping. Closer examination of mapping performance on Jazz shows that the normalization takes almost as much time as does the sparse-matrix multiply.

5. Conclusions and Future Directions

Cpl6 has been used in the CCSM production system since March 2003. CCSM3 with cpl6 has successfully completed over 10,000 years of production climate simulations. Development of this new coupler was a highly successful multiyear collaboration between NCAR and Department of Energy laboratories. The portability, performance, and extensibility of cpl6 are much improved over cpl5. The cpl6 coupler was released to the community in the June 2004 release of CCSM3. CCSM3 is available for download at www.ccsm.ucar.edu.

Algorithms and performance scaling plots were presented for several important operations in cpl6, including the flux computation, merging, communication, and mapping. In general, memory and floating-point operations are about two times slower on the Linux cluster compared to either the IBM p690 or HP systems. Scaling of routines is generally good on all platforms, although the scaling on the Linux cluster far outperforms any other systems for these timing tests. The timed routines in cpl6 perform comparably on both the IBM p690 and HP systems for nearly all tests. The IBM p690 communication timing is particularly good if communication remains exclusively "on-node" but is quite poor if communication is "off-node" Some routines also perform relatively poorly on the IBM p690 when the memory subsystem is pushed.

Overall, cpl6 timings are improved compared to cpl5 for the same operations on the same processor counts. In addition, cpl6 can be scaled to give performance unreachable by the shared-memory-only cpl5. This can help reduce the total time to solution of the full system. Unfortunately, an evaluation of overall CCSM3 performance and cpl6's role in it is beyond the scope of this paper.

Cpl6 is more extensible than previous versions of the coupler and provides standardized coupler-model interfaces. With the fields module, cpl6 provides a single location for the coordinated addition of new fields to the coupled model data flow. In contrast to CCSM2, all components in CCSM3 now use the same model-coupler interface module. New models can replace current components by adding calls to these interface routines.

By abstracting the communication method away from the components using the contract datatype and simple interfaces, cpl6 is in effect attached to the components. This means cpl6 functionality can be migrated to the component processors if there is a performance benefit. In the extreme case, all cpl6 functionality (mapping, merging, flux calculation, diagnostics, etc.) could be migrated into the interface layer, and the coupler as a unique CCSM component could disappear.

Cpl6 was designed for extension. One future avenue of exploration is alternatives to the multiple executable-concurrent execution configuration and an effort is under way to experiment with other options such as single executable with mixed concurrent and sequential execution. The MPH library used by cpl6 to assign MPI communicators to models already supports many alternative execution modes. Cpl6 datatypes can be used to form new couplers and coupled systems readily. However, component models may require modification beyond what the current architecture requires. CCSM components are expected to move to higher resolutions. Benchmarks are already being carried out on higher-resolution grids using cpl6 and the dead versions of the components.

A close collaboration exists between CCSM and the Earth System Modeling Framework (ESMF) (Hill *et al.*, 2004). Much of what was learned and implemented in cpl6 is informing the ESMF design. The next CCSM coupler may have a combination of cpl6, MCT, MPH, and ESMF. The CCSM project is committed to continually improving the software engineering infrastructure for climate modeling, using the best tools available, with computational performance a high priority.

Acknowledgments This work was supported in part by the Climate Change Research Division subprogram of the Office of Biological & Environmental Research, Office of Science, U.S. Department of Energy through the Climate Change Prediction Program (CCPP), the Accelerated Climate Prediction Initiative (ACPI-Avante Garde), and the Scientific Discovery through Advanced Computing (SciDAC) Program, under Contract W-31-109-ENG-38. CCSM development is jointly funded by the Department of Energy and the National Science Foundation. We thank members of the Computation Science Section of the Scientific Computing Division at NCAR for their many constructive suggestions during the development of the cpl6 coupler. We thank John Michalakes for providing valuable comments on a early version of this paper. We gratefully acknowledge use of "Jazz," a 350-node computing cluster operated by the Mathematics and Computer Science Division at Argonne National Laboratory as part of its Laboratory Computing Resource Center. Additional computations were performed on the National Science Foundation Terascale Computing System at the Pittsburgh Supercomputing Center. The National Center for Atmospheric Research is managed by the University Corporation for Atmospheric Research under the sponsorship of the National Science Foundation.

REFERENCES

Bettge, T., A. Craig, R. James, V. Wayland, and G. Strand, 2001: The DOE Parallel Climate Model (PCM): The Computational Highway and Backroads. In V. N. Alexandrov, J. J. Dongarra,, and C. J. K. Tan (Eds.), *Proc. International Conference on Computational Science (ICCS) 2001*, Volume 2073 of *Lecture Notes in Computer Science*, Berlin, pp. 148–156. Springer-Verlag.

Boville, B. A., and P. R. Gent, 1998: The NCAR Climate System Model, Version One. *J. Climate,* **11**, 1115–1130.

Bryan, F. O., B. G. Kauffman, W. G. Large, and P. R. Gent, 1996: The NCAR CSM Flux Coupler. NCAR Tech. Note 424, NCAR, Boulder, CO.

Collins, W. D., M. Blackmon, C. Bitz, G. Bonan, C. Bretherton, J. A. Carton, P. Chang, S. Doney, J. J. Hack, J. T. Kiehl, T. Henderson, W. G. Large, D. McKenna, B. D. Santner, and R. D. Smith, 2005: The Community Climate System Model: CCSM3. *J. Climate,*, preprint.

DAO, 1997: The GEOS-3 Data Assimilation System. Office Note Series on Global Modeling and Data Assimilation DAO Office Note 97-06, NASA Goddard Space Flight Center.

Drummond, L. A., J. Demmel, C. R. Mechose, H. Robinson, K. Sklower, and J. A. Spahr, 2001: A Data Broker for Distirbuted Computing Environments. In V. N. Alexandrov, J. J. Dongarra,, and C. J. K. Tan (Eds.), *Proc. 2001 International Conference on Computational Science*, pp. 31–40. Springer-Verlag.

He, Y., and C. Ding, 2005: Coupling Multi-Component Models by MPH on Distributed Memory Computer Architecture. *Int. J. High Perf. Comp. App.,*, this issue.

Hill, C., C. DeLuca, V. Balaji, M. Suarez, A. da Silva, and the ESMF Joint Specification Team, 2004: The Architecture of the Earth System Modeling Framework. *Comp. in Science and Engineering,* **6**.

Jacob, R., J. Larson, and E. Ong, 2005: MxN Communication and Parallel Interpolation in CCSM3 Using the Model Coupling Tookit. *Int. J. High Perf. Comp. App.,*, this issue.

Jacob, R., C. Schafer, I. Foster, M. Tobis, and J. Anderson, 2001: Computational Design and Performance of the Fast Ocean Atmosphere Model. In V. N. Alexandrov, J. J. Dongarra,, and C. J. K. Tan (Eds.), *Proc. 2001 International Conference on Computational Science*, pp. 175–184. Springer-Verlag.

Jones, P. W., 1998: A User's Guide for SCRIP: A Spherical Coordinate Remapping and Interpolation Package. , Los Alamos National Laboratory, Los Alamos, NM.

Jones, P. W., 1999: First and Second-Order Conservative Remapping Schemes for Grids in Spherical Coordinates. *Mon. Wea. Rev.,* **127**, 2204–2210.

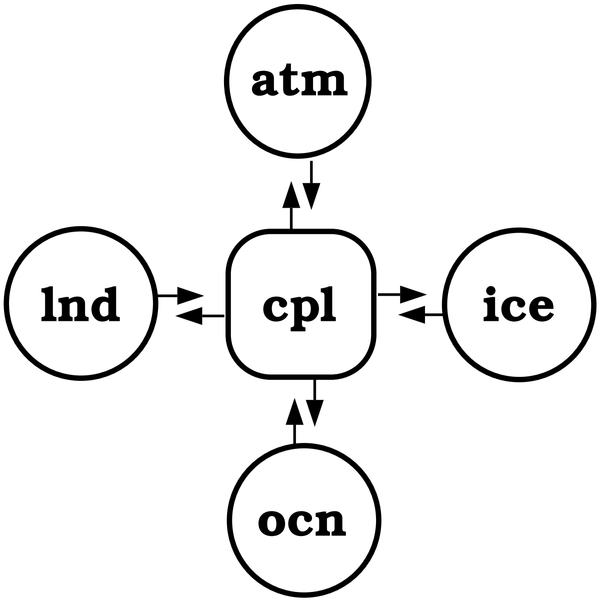
Kiehl, J., and P. R. Gent, 2004: The Community Climate System Model, Version Two. *J. Climate,* **17**, 3666–3682.

Larson, J., R. Jacob, and E. Ong, 2005: The Model Coupling Toolkit:

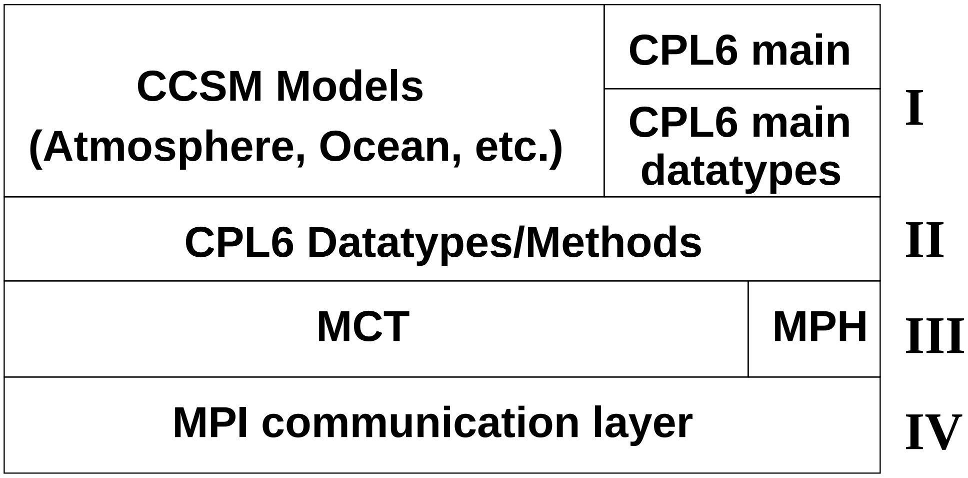
A New Fortran90 Toolkit For Building Multi-Physics Parallel Coupled Models. *Int. J. High Perf. Comp. App.*,, this issue.

Valcke, S., A. Caubel, R. Vogelsang, and D. Declat, 2004: OASIS3 Ocean Atmosphere Sea Ice Soil User's Guide. Technical Report TR/CMGC/04/68, CERFACS, Toulouse, France.

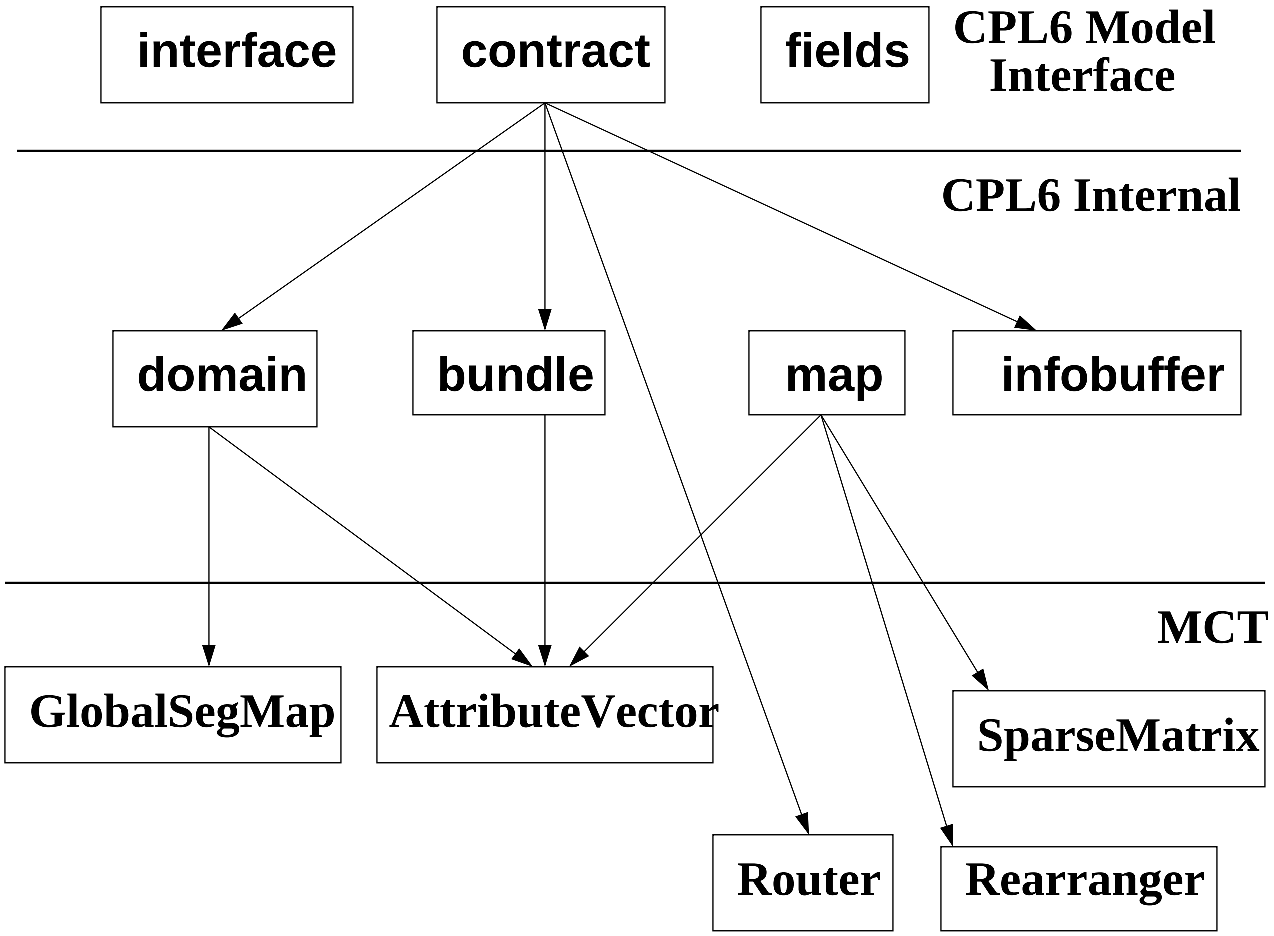
Washington, W. M., J. W. Weatherly, G. A. Meehl, A. J. Semtner, T. W. Bettge, A. P. Craig, W. G. Strand, J. M. Arblaster, V. B. Wayland, R. James, and Y. Zhang, 2000: Parallel Climate Model (PCM) Control and Transient Simulations. *Climate Dynamics,* **16**, 754–774.



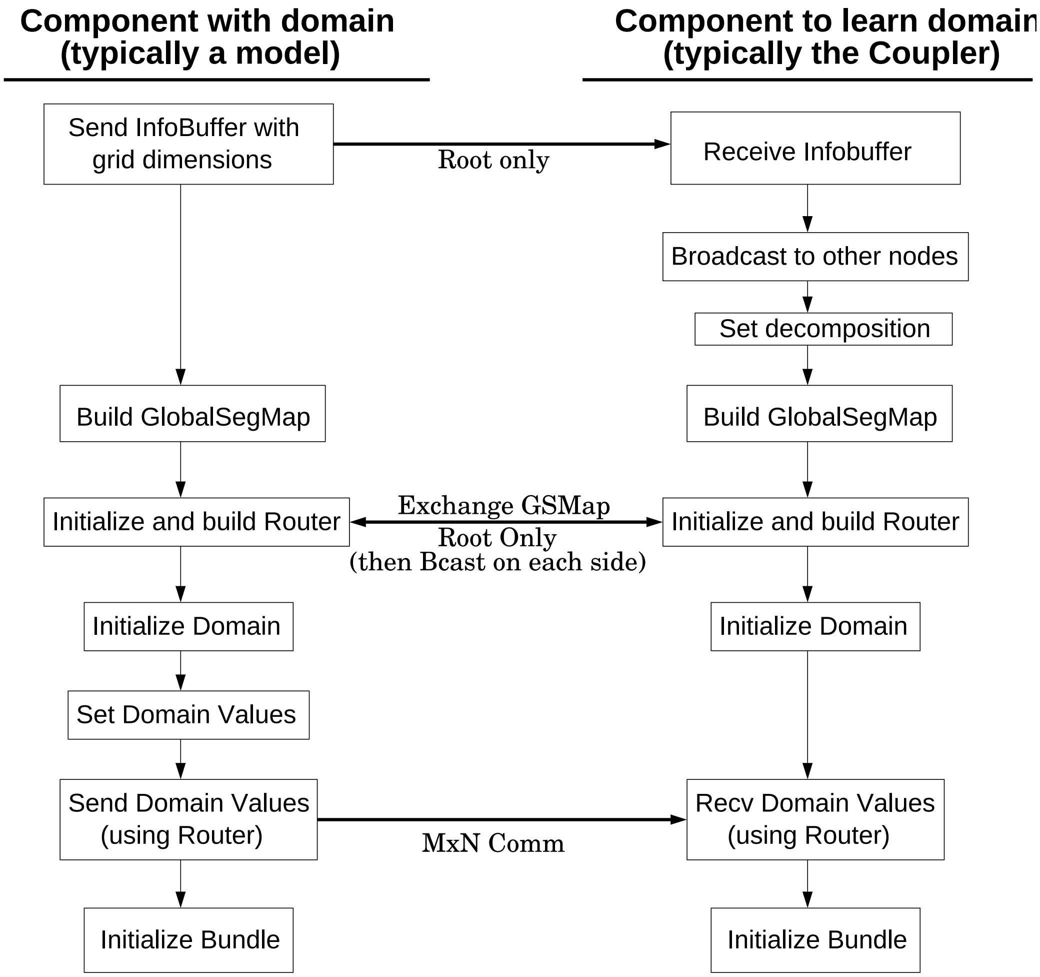
1


CCSM Models
(Atmosphere, Ocean, etc.)
CPL6 main
CPL6 main datatypes
I
CPL6 Datatypes/Methods
II
MCT
MPH
III
MPI communication layer
IV

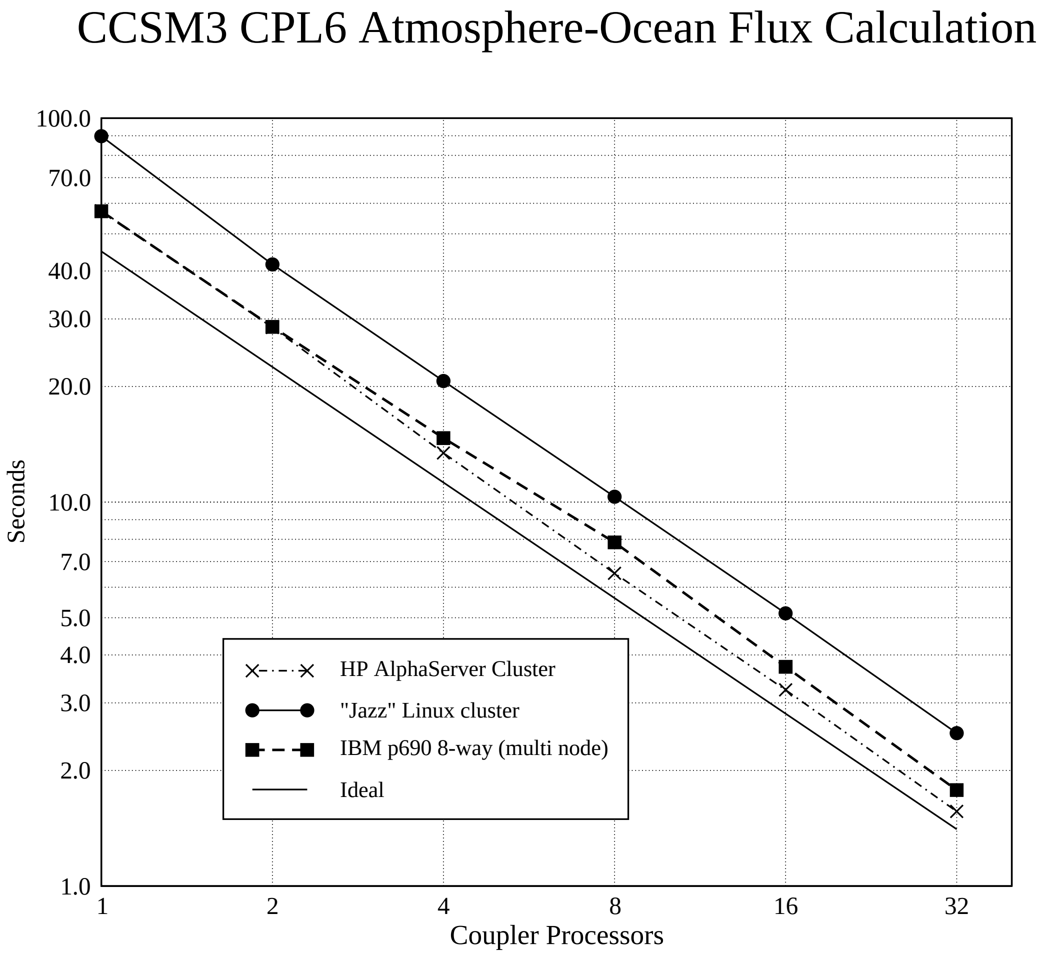
2


interface
contract
fields
CPL6 Model Interface
CPL6 Internal
domain
bundle
map
infobuffer
MCT
GlobalSegMap
AttributeVector
SparseMatrix
Router
Rearranger

3

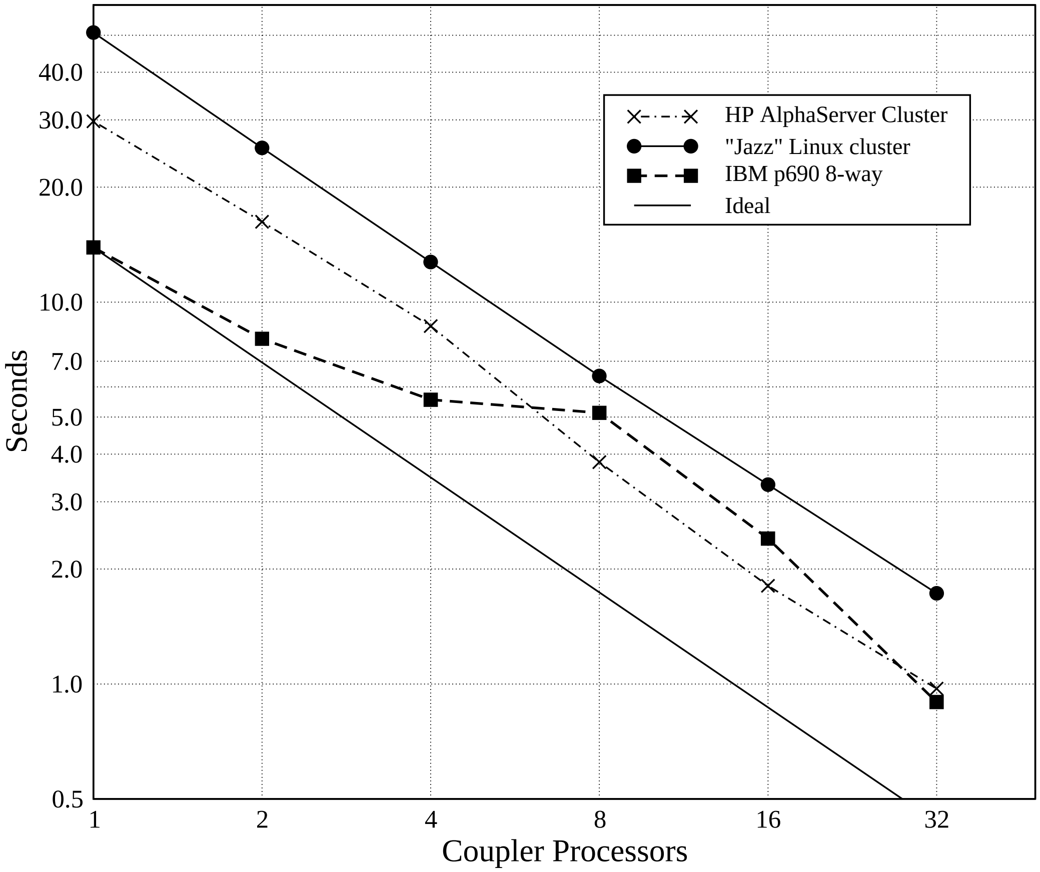

Component with domain
(typically a model)
Component to learn domair
(typically the Coupler)
Send InfoBuffer with
grid dimensions
Root only
Receive Infobuffer
Broadcast to other nodes
Set decomposition
Build GlobalSegMap
Build GlobalSegMap
Initialize and build Router
Exchange GSMap
Root Only
(then Bcast on each side)
Initialize and build Router
Initialize Domain
Initialize Domain
Set Domain Values
Send Domain Values
(using Router)
MxN Comm
Recv Domain Values
(using Router)
Initialize Bundle
Initialize Bundle

4



CCSM3 CPL6 Atmosphere-Ocean Flux Calculation

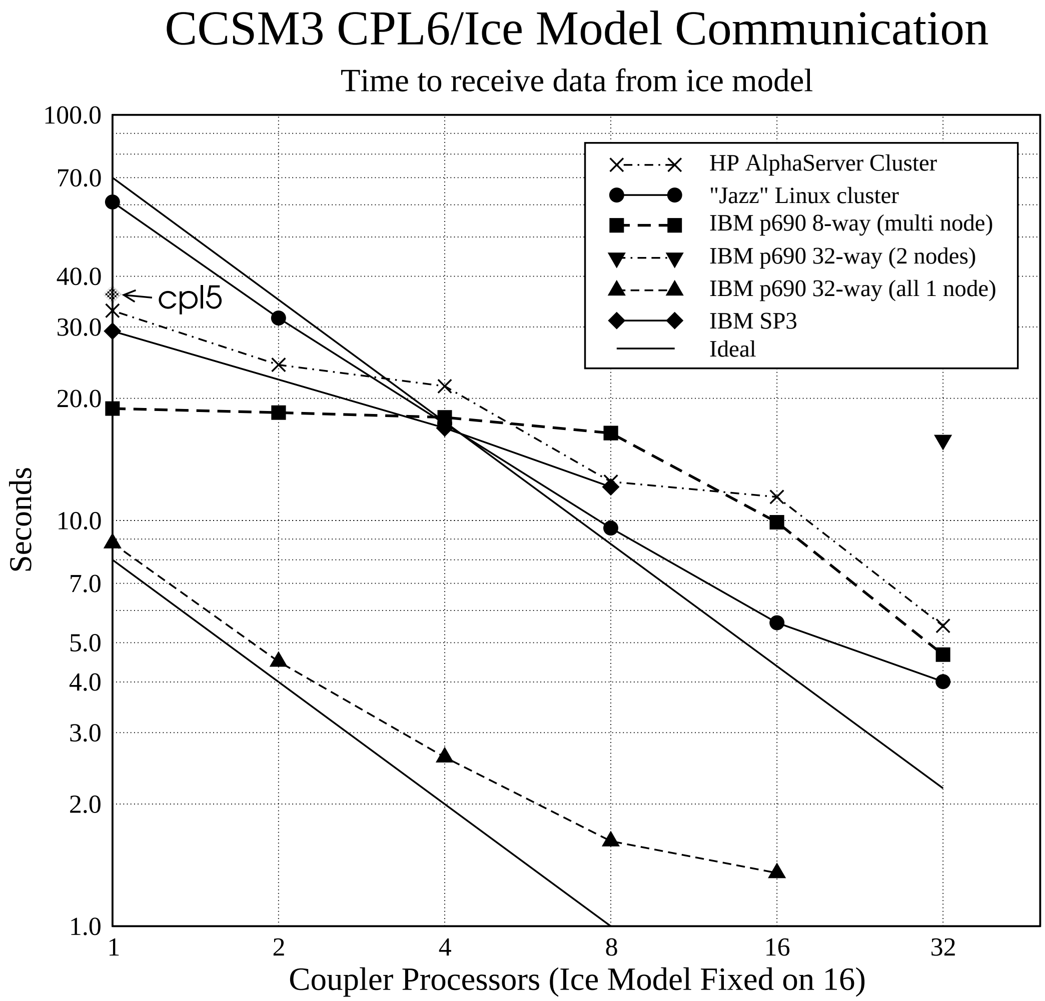
5



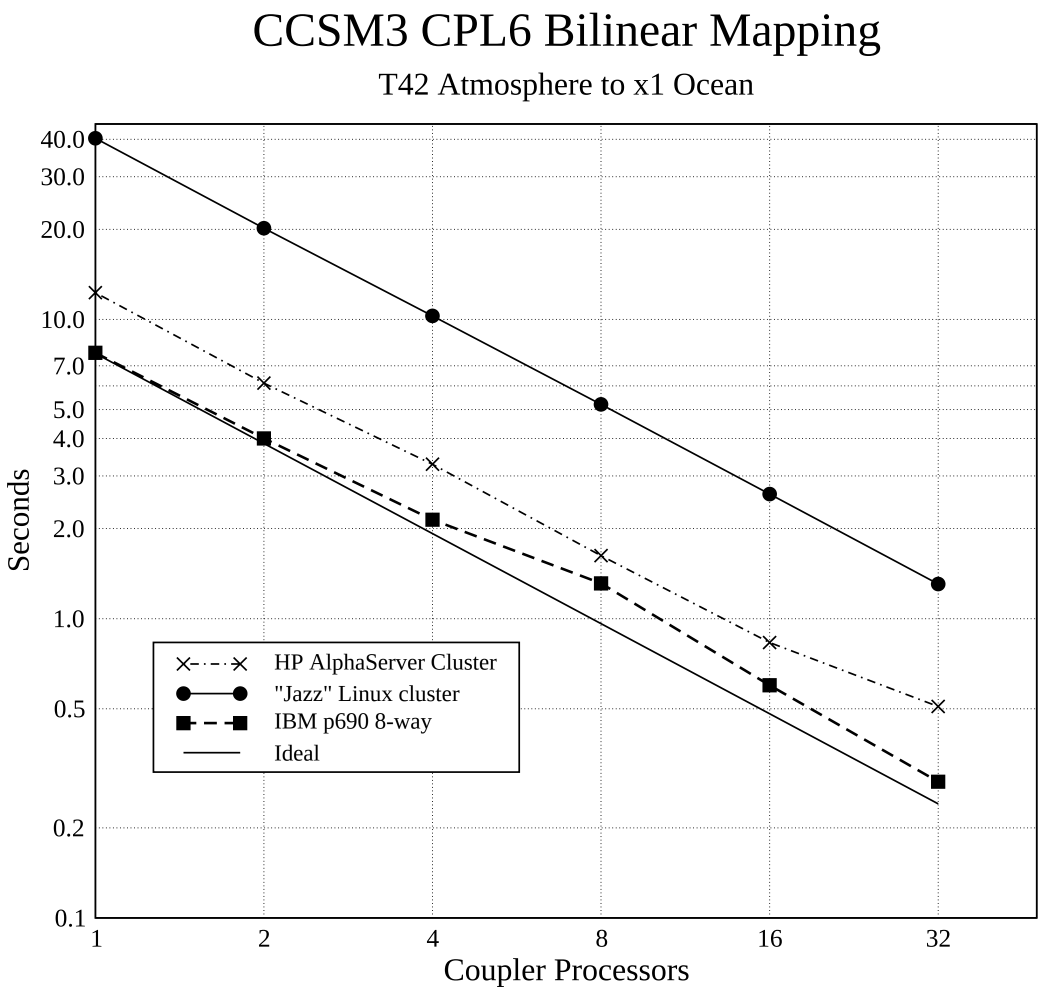
CCSM3 CPL6 Ocean Merge

Formation of Coupler To Ocean Bundle

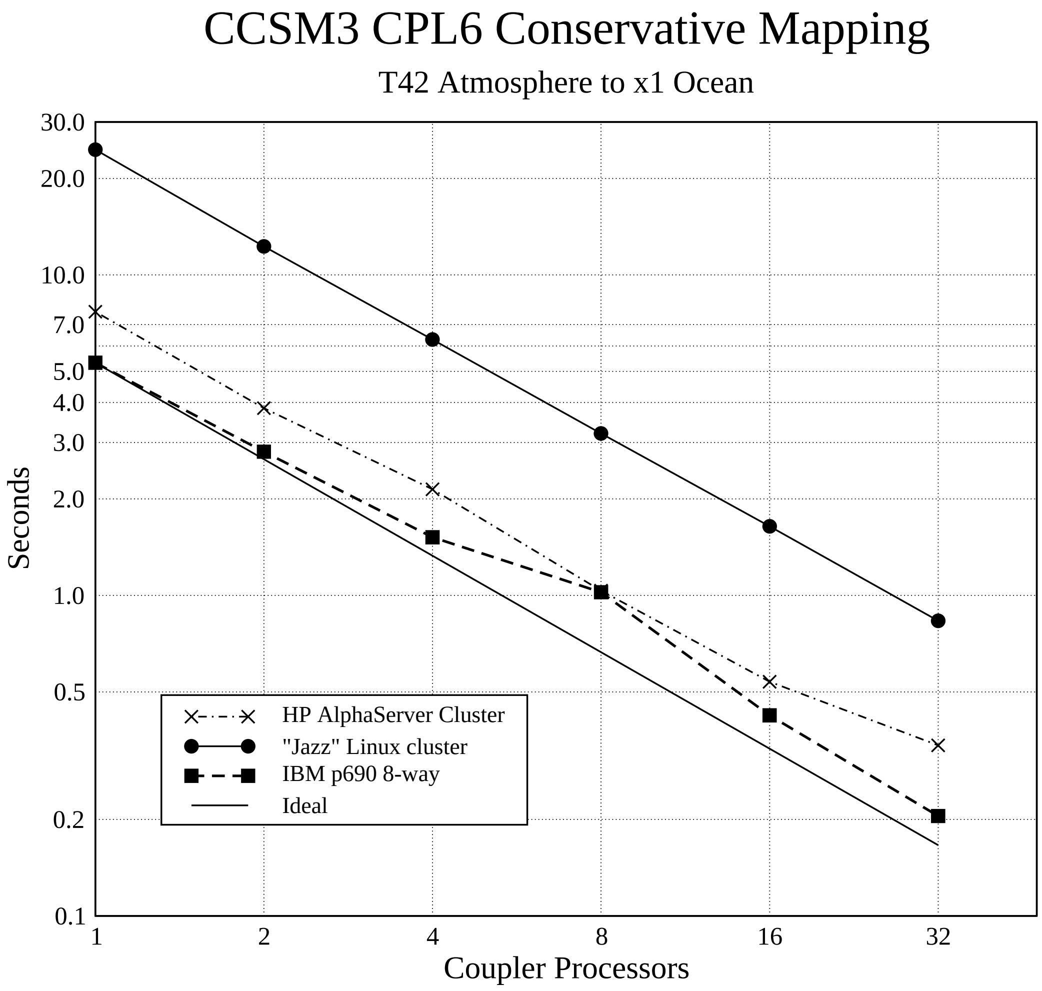
6


CCSM3 CPL6/Ice Model Communication
Time to receive data from ice model
Seconds
100.0
70.0
40.0
30.0
20.0
10.0
7.0
5.0
4.0
3.0
2.0
1.0
1
2
4
8
16
32
Coupler Processors (Ice Model Fixed on 16)
cpl5
HP AlphaServer Cluster
"Jazz" Linux cluster
IBM p690 8-way (multi node)
IBM p690 32-way (2 nodes)
IBM p690 32-way (all 1 node)
IBM SP3
Ideal

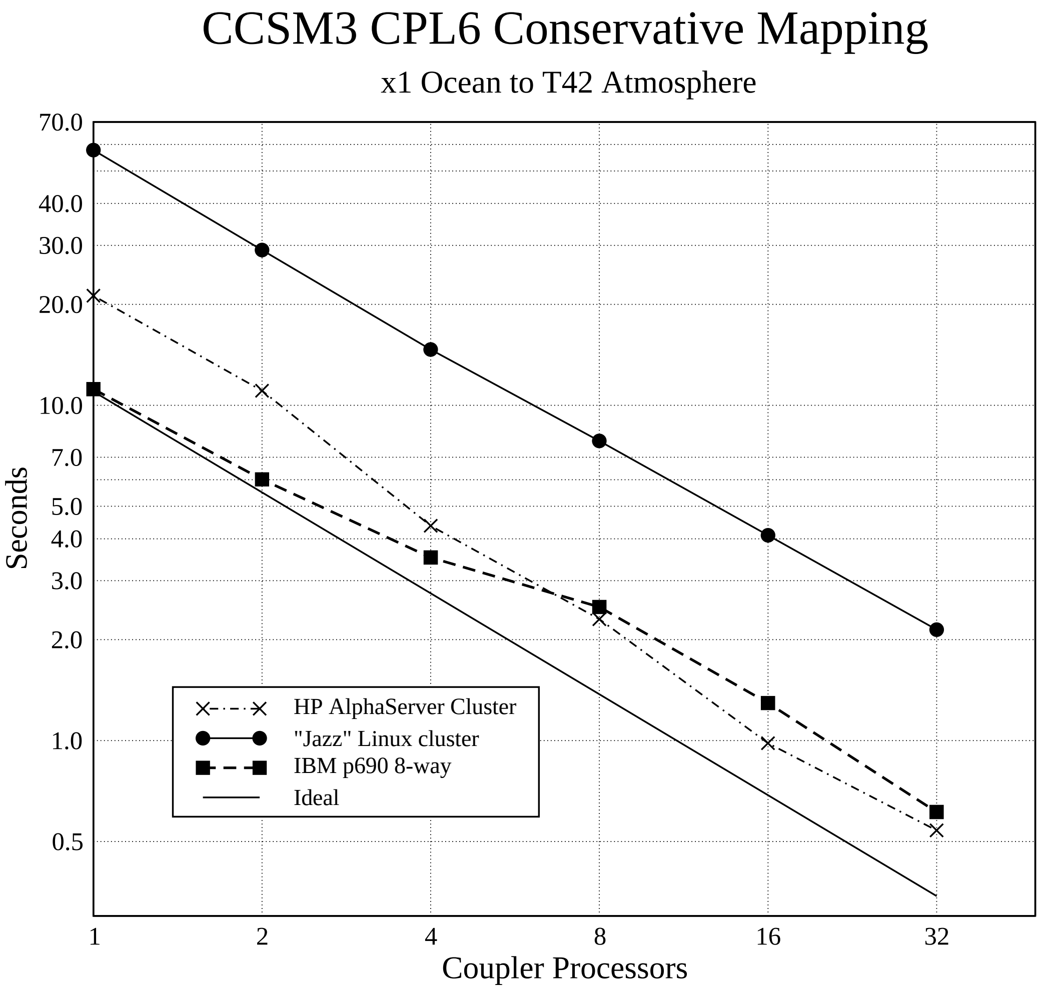
7


CCSM3 CPL6 Bilinear Mapping
T42 Atmosphere to x1 Ocean
Seconds
40.0
30.0
20.0
10.0
7.0
5.0
4.0
3.0
2.0
1.0
0.5
0.2
0.1
1
2
4
8
16
32
Coupler Processors
HP AlphaServer Cluster
"Jazz" Linux cluster
IBM p690 8-way
Ideal

8


CCSM3 CPL6 Conservative Mapping
T42 Atmosphere to x1 Ocean
30.0
20.0
10.0
7.0
5.0
4.0
3.0
2.0
1.0
0.5
0.2
0.1
Seconds
1
2
4
8
16
32
Coupler Processors
HP AlphaServer Cluster
"Jazz" Linux cluster
IBM p690 8-way
Ideal

9


CCSM3 CPL6 Conservative Mapping
x1 Ocean to T42 Atmosphere
70.0
40.0
30.0
20.0
10.0
7.0
5.0
4.0
3.0
2.0
1.0
0.5
Seconds
1
2
4
8
16
32
Coupler Processors
HP AlphaServer Cluster
"Jazz" Linux cluster
IBM p690 8-way
Ideal

10

FIGURE CAPTIONS

Figure 1.

The “hub-and-spoke” architecture of the Community Climate System Model.

Figure 2.

Layered software architecture of CCSM3 with cpl6.

Figure 3.

Cpl6 datatypes and their dependencies.

Figure 4.

Cpl6 algorithm for initializing a contract between two models. The datatypes Router, domain, and bundle are contained in each contract instantiated.

Figure 5.

Total time required for 240 calls to the cpl6 atmosphere-ocean flux calculation routine.

Figure 6.

Total time for 240 calls to the cpl6 custom ocean merge routine.

Figure 7.

Total time for 240 receives of data from the sea-ice model as a function of coupler processors.

Figure 8.

Total time for 240 calls to the cpl6 mapping routine for the interpolation of 9 fields from the atmosphere to the ocean grid using bilinear interpolation.

Figure 9.

Total time for 240 calls to the cpl6 mapping routine for the interpolation of 10 fields from the atmosphere to the ocean grid using 1st-order conservative interpolation.

Figure 10.

Total time for 240 calls to the cpl6 mapping routine for the interpolation of 13 fields from the ocean to the atmosphere grid using 1st-order conservative interpolation.