M × N Communication and Parallel Interpolation in Community Climate System Model Version 3 Using the Model Coupling Toolkit

Robert Jacob, Jay Larson and Everest Ong

International Journal of High Performance Computing Applications 2005 19: 293
DOI: 10.1177/1094342005056116

M × N COMMUNICATION AND PARALLEL INTERPOLATION IN COMMUNITY CLIMATE SYSTEM MODEL VERSION 3 USING THE MODEL COUPLING TOOLKIT

Robert Jacob
Jay Larson
Everest Ong

MATHEMATICS AND COMPUTER SCIENCE
DIVISION ARGONNE NATIONAL LABORATORY
ARGONNE, IL 60439, USA (JACOB@MCS.ANL.GOV)

Abstract

The Model Coupling Toolkit (MCT) is a software library for constructing parallel coupled models from individual parallel models. MCT was created to address the challenges of creating a parallel coupler for the Community Climate System Model (CCSM). Each of the submodels that make up CCSM is a separate parallel application with its own domain decomposition, running on its own set of processors. This application contains multiple instances of the $M \times N$ problem, the problem of transferring data between two parallel programs running on disjoint sets of processors. CCSM also requires efficient data transfer to facilitate its interpolation algorithms. MCT was created as a generalized solution to handle these and other common functions in parallel coupled models. Here we describe MCT's implementation of the data transfer infrastructure needed for a parallel coupled model. The performance of MCT scales satisfactorily as processors are added to the system. However, the types of decompositions used in the submodels can affect performance. MCT's infrastructure provides a flexible and high-performing set of tools for enabling interoperability between parallel applications.

Key words: Parallel coupling, parallel communication, climate modeling, coupled models

The International Journal of High Performance Computing Applications,
Volume 19, No. 3, Summer 2005, pp. 293–307
DOI: 10.1177/1094342005056116

1 Introduction

A growing trend in high performance scientific computing is the creation of new applications for a multidisciplinary problem by combining two or more separate applications from individual disciplines. One field that has pioneered this approach is climate modeling. A climate model usually contains multiple submodels that simulate the behavior of physical subsystems such as the global atmosphere, the global ocean, the land surface, and sea ice. Each model is produced by practitioners of a subdiscipline in the atmospheric and oceanic sciences. A coupled climate model is created by combining component[1] models and allowing them to mutually provide boundary conditions for each other.

Like other high performance scientific applications, climate models are implemented as parallel programs operating on physically distributed data. Coupled climate models are a combination of individual distributed-memory parallel programs.

Version 3 of the Community Climate System Model (CCSM; Collins et al., 2005) is an example of a state-of-the-art parallel coupled climate model. CCSM is a collection of high performance applications that simulate the interaction of the Earth's ocean and atmosphere, its land surface, and sea ice. Although the components of climate models are physically three-dimensional systems, their common interface is a two-dimensional surface. The coupling problem amounts to representing the physical fluxes across the two-dimensional interface in a consistent and coordinated way. CCSM's architecture is a hub-and-spokes model as shown in Figure 1. The exchange of information across the surface and overall time integration of the system are controlled by a fifth application called the coupler (the hub in Figure 1). All models send and receive data only with the coupler and not with each other. This approach provides a convenient central point of control for dealing with important scientific requirements in the model, such as enforcing global conservation of energy exchanged between the models and compensating for different time-steps in the course of simulating a day.

When creating a climate model such as CCSM from separate parallel codes, the standard approach in the community has been to combine them under a single, ad hoc software framework. The new version of CCSM standardizes its framework with a newly designed library called cpl6. Cpl6 marks a major advance over previous versions of the CCSM coupler because of its modular design and allowance for distributed-memory parallelism in the coupler itself. Cpl6 and the design of CCSM are described further in a companion paper (Craig et al., 2005).

For building parallel coupled models, cpl6 uses a new software library called the Model Coupling Toolkit

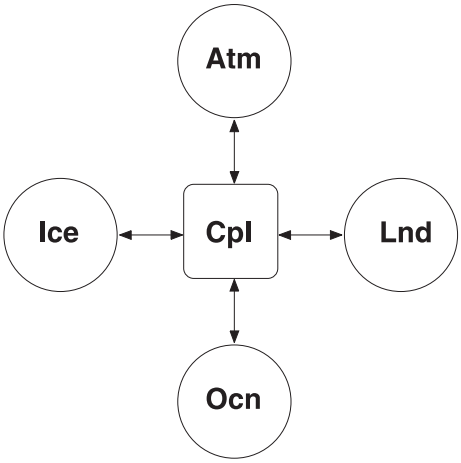

Fig. 1 The hub-and-spokes execution model of the CCSM. CCSM contains five separate executables: an atmosphere model (Atm), ocean model (Ocn), land model (Lnd), sea ice model (Ice), and a coupler (Cpl).

(MCT). Although written to address the coupling needs of an earth science model, MCT is a general-purpose library that can be used to couple any models exchanging data represented on a numerical grid, whether structured or unstructured. An overview of MCT is described in another companion paper (Larson et al., 2005). Here we focus on how MCT solves the largest problem posed by the CCSM coupler: its parallel data transfer needs. In Section 2 we describe the parallel data transfer characteristics of CCSM and the problems that motivated the creation of MCT. In Section 3 we describe MCT's solutions to these problems. In Section 4 we examine the performance of MCT's data transfer methods. We conclude in Section 5 with a discussion of MCT's role in other applications.

2 Parallel Communication and CCSM's Coupler

The component models in CCSM typically have different numerical methods for solving their respective system of partial differential equations, and these methods may employ different types of numerical grids or grids with different resolutions. The physical component models in recent versions of CCSM use distributed-memory parallelism and domain decomposition to distribute grid points in the horizontal (north–south and east–west) direction.

The Message Passing Interface (MPI, version 1; MPI Forum, 1994) is typically used to communicate data within each component as required by their numerical methods. This intramodel communication has typically been handled by libraries or methods chosen or developed by the individual model development teams. MCT address the model–coupler, or intermodel, parallel communication requirements of CCSM and the intramodel communication requirements of a distributed-memory parallel coupler. In addition, MCT satisfies other requirements identified for the new coupler, such as extensibility and generalization of the model–coupler interface. Further discussion of these requirements and how they are met by MCT and cpl6 can be found in the companion papers (Craig et al. 2005; Larson et al., 2005).

2.1 MODEL–COUPLER COMMUNICATION

In previous versions of CCSM, MPI was used to communicate data between the models and the coupler (along the spokes in Figure 1), often with direct calls to the send and receive functions of the MPI library.

Prior to the current release, the coupler itself was not a distributed memory parallel application. It was a separate executable that used OpenMP threading in some floating-point-intensive portions of the code but ran as a single MPI process. Communication to the coupler was achieved by a model first gathering data to its MPI root processor and then sending it to the coupler's single MPI process in a single message. Since the future development path of CCSM (CCSM Science Plan 2004–2008, http://www.ccsm.ucar.edu/management/sciplan2004-2008.pdf) points to increasing horizontal resolution (currently at about 250 km) and increasing the number of physical processes simulated, this one-processor communication point was going to become more of a bottleneck.

Since all data go through the coupler, the coupler must maintain a representation of each model's numerical grid. In a parallel coupler, a decomposition of that grid over the coupler's processors is required. With each model running on its own M processors and the coupler running on N processors, CCSM with a parallel coupler contains multiple examples of the $M \times N$ problem. The $M \times N$ problem is 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/~febertra/mxn for a summary). The exact pattern of communication will change as M and N change, as they do when load balancing CCSM for different problem sizes and hardware systems, and will also be different for each model–coupler pair. A solution that was both efficient and scalable was needed. MCT provides a general solution to this problem by deriving a set of point-to-point communications that transfers the data with a minimum number of messages (Section 3.2). With MCT, the new coupler in CCSM, cpl6, is now a distributed-memory parallel application.

2.2 INTRACOUPLER COMMUNICATION

An important function of the coupler is interpolating, or mapping, data between the numerical grids of the models as data are routed through the coupler. In CCSM, interpolation is performed as a matrix–vector multiply (see Section 3.3). Each two-dimensional grid (for the atmosphere, ocean, land, or sea ice models) is unrolled into a vector, one for the source grid and one for the destination. The matrix of mapping weights contains a row for each source element and a column for each destination. The two-dimensional grids have between 10^4 and 10^5 grid points, but most of the matrix elements are zero, so that interpolation is a sparse-matrix–vector multiply. In CCSM, the grids are fixed and regular, and the non-zero elements are computed once off-line by using the SCRIP program (Jones, 1998) and read in at run-time.

In the previous versions of the coupler, interpolation was trivial because all the data on the source and destination grids and the matrix elements were held in a single address space. A parallel coupler requires distributing the matrix elements accounting for the decomposition of the two grids. Given that the grids of the ocean and atmosphere model, for example, are decomposed arbitrarily in the coupler, there is no guarantee that all the atmosphere data on the atmosphere grid needed to complete an interpolation onto the ocean grid will be colocated on the same processor. Thus, although the matrix elements need to be distributed only at initialization, source data, which are updated each time-step, need to be redistributed each time-step before interpolation. MCT also provides methods for these parallel communication needs.

3 Parallel Communication with MCT

The introduction of distributed-memory parallelism to the CCSM coupler created new parallel communication needs, both for data transfer between models and the coupler and for parallel data interpolation within the coupler.

Other software packages contain general solutions for communication between parallel data structures, including PAWS (Beckman et al., 1998; Keahey et al., 2001), CUMULVS (Geist et al., 1997), MetaChaos (Ranganathan et al., 1996; Edjlali et al., 1997) and its successor InterComm (Lee and Sussman, 2004). There are also domain-specific or domain-inspired solutions such as Roccom (Jiao et al., 2003), the Distributed Data Broker (DDB; Drummond et al., 2001), and the Flexible Modeling System (http://www.gfdl.noaa.gov/fms). Others have based their solution on the Common Object Request Broker Architecture (CORBA), such as MCEL (Bettencourt, 2002) or proposed extensions to CORBA such as GridCCM (Perez et al., 2003) and Pardis (Keahey and Gannon, 1997). In view of the requirements above, however, these packages were not entirely suitable as the basis for a parallel coupler in CCSM. Some, such as CUMULVS and MCEL, did not provide $M \times N$ data transfer at the time this project began. Others lack a Fortran interface, introduce a major new package dependency such as the Parallel Virtual Machine (PVM; Geist et al., 1994), handle only certain parallel data types, or do not provide interpolation abilities.

Part of the MCT approach was to use only languages and libraries already present in CCSM: MPI and Fortran90. MCT is written entirely in Fortran90. Fortran90 supports derived types and allows grouping public and private subroutines and functions into modules. (In this paper we occasionally refer to the modules as classes and the subroutines as methods even though Fortran90 is not a true object-oriented language.) Since CCSM was already using MPI to pass data between models and the coupler (as a single, root-to-root message), MCT provides an MPI-style double-sided message passing model for solving the $M \times N$ problem: MCT calls replace the single MPI calls of the non-parallel cpl5 coupler at the same locations in each model's code (see Section 3.2). MPI is still used underneath MCT as the communication layer. These choices allowed MCT to provide solutions to CCSM's needs without impacting the model's portability.

3.1 DATA STORAGE AND DECOMPOSITION

To cope with the wide variety of data types in CCSM, MCT introduces a standard data representation called the AttributeVector.[2] The AttributeVector is a Fortran90 derived type that consists of a two-dimensional array of reals and one of integers. The first index refers to the Attribute corresponding to the physical quantity being stored, such as temperature, wind, or humidity. The second index refers to the value of each attribute at a physical grid point. All the fields sent to or received from the coupler can each be contained in an instance of the AttributeVector. In CCSM3, the AttributeVector is the data type exchanged between the coupler and other models, and AttributeVectors are the parallel data type used within the coupler. AttributeVectors are locally sized: they contain just enough space to hold the data local to a processor in a model's decomposition. Methods that operate on AttributeVector, such as communication methods equivalent to an MPI broadcast and gather, operate on all the attributes at once: the temperature, wind speed, and other values in the AttributeVector are sent in one message to their destinations.

The GlobalSegmentMap or GSMap is the MCT data type for describing a decomposition of a numerical grid or the portion of a grid used in coupling. The data type is

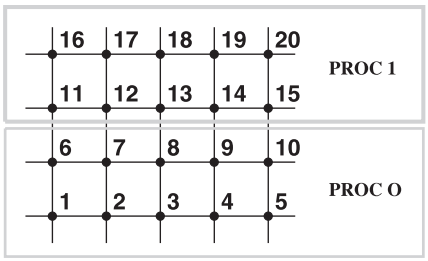

Fig. 2 Two-rows × one-column decomposition of a numbered grid.

global because it contains a description of the decomposition of the entire grid. After initialization, which is typically done collectively with each processor describing its portion of the grid, the returned data type is identical on all processors. Thus, each processor can inquire the processor ID of any point in the numerical grid.

The GSMap is defined by numbering all the points in the numerical grid to be described. A very simple example of a grid with 20 numbered points distributed over two processors is shown in Figure 2. The GSMap data type values for this decomposition are as follows:

```
ngseg:  2      ! total number of segments
gsize:  20     ! total number of points
start:  1 , 11 ! value of starting point
               !          for each segment
length: 10, 10 ! length of each segment.
pe_loc:  0 , 1 ! MPI rank of processor
               !  containing each segment.
```

Using the three integer arrays start, length, and pe_loc, it is possible to describe the decomposition of any grid, structured or unstructured, provided the grid points can be conceptually numbered sequentially. The GSMap is efficient at describing block decompositions because that type is used most often in climate modeling.

Before data are sent to the coupler, values must be copied from the model's internal data type into the correct location in the AttributeVector. The correct location is determined by the implied relationship between an AttributeVector and the GSMap. The implied relationship between local memory in the AttributeVector and the local portion of the GSMap is shown in Figure 3 for the points on processor 1 of Figure 2. Because the data types of individual models in CCSM are varied and unknown

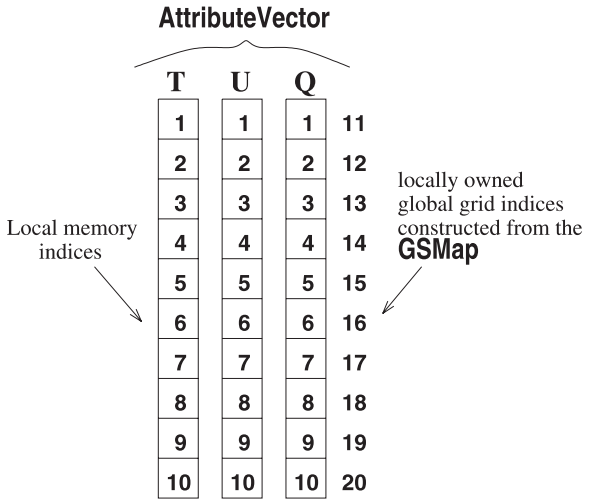

Fig. 3 Relationship between local memory indices in an AttributeVector and the grid point numbering contained in the GSMap for the points owned by processor 1 in Figure 2. In this example, the AVect is being used to store three variables: temperature (*T*), east–west velocity (*U*), and specific humidity (*Q*).

to MCT, MCT serializes the grid of the model. MCT then uses the implicit relationship between the memory of the AttributeVector and the locally owned portion of the GlobalSegmentMap shown in Figure 3 to map data between processor–grid point spaces. The GSMap method GlobalToLocal translates between global values of locally owned grid points and memory indices in the AttributeVector. The MCT user has the responsibility for copying data between gridded data stored in their model's internal data types and the AttributeVector according to the relationship in Figure 3. By generalizing the decomposition of the grid instead of trying to generalize the model's unknown internal parallel data type, MCT gains great flexibility at the cost of a single data copy.

3.2 COMMUNICATION SCHEDULES AND $M \times N$ TRANSFER

In order to send data between two models, an AttributeVector with the same number of Attributes and related to the same numerical grid with the same grid point numbering scheme must exist on each side of the communication. The local size of the data may be different depending on the different decompositions. For example, the CCSM

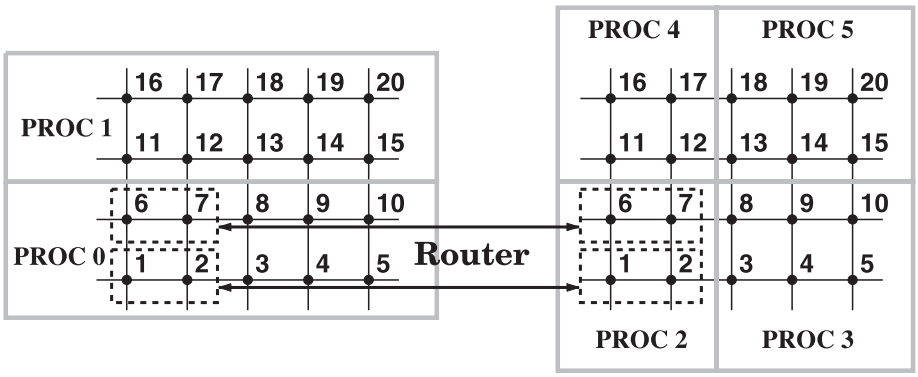

Fig. 4 Schematic diagram showing how an MCT `Router` maps points between two decompositions of the same grid.

atmosphere model may decompose its grid over 64 processors while the coupler decomposes its representation of the atmosphere's grid over 16 processors. The model and coupler will then have two different GSMaps, and the local size of the AttributeVector for "atmosphere-to-coupler" data will be different, but MCT can still transfer the data so long as the atmosphere grid is numbered the same way in each model.

Given two decompositions of the same numbered numerical grid specified in two GSMaps, one can easily build a mapping between the location of one grid point on a processor to its location on another processor. The set of all these mappings forms a customized $M \times N$ routing table. This table can be used by a processor to indicate the destination or source processor for each of its data points in the alternative decomposition. In MCT, this assembled table is stored in another Fortran90 derived type called the Router.

Router initialization is a form of "handshaking", where two models learn how to exchange data with each other in parallel. The algorithm has two phases. First, the models exchange their GSMaps. This is a synchronization point between models. The received GSMap is then broadcast to each processor of that model. At that point, each processor has the GSMap for each side of the communication and can build its local Router in parallel with the other processors. This is the usual method for CCSM, where the models are each on disjoint sets of processors. MCT also provides methods to send a GSMap asynchronously and to initialize a Router if the two GSMaps are already available. Since the grids in a climate model such as CCSM are fixed, Router initialization is typically done once at startup.

After initialization, the Router can be read two ways: as both a list of local memory indices of the local AttributeVector and a list of MPI processor ranks to which they must be sent, and as a list of local memory indices to receive data from a list of processors. The Router is a two-way map from one decomposition to another, and therefore the same Router data can be used for an $M \times N$ send or receive.

Figure 4 illustrates a portion of a Router for processor 0 in a system with two components spread over six processors. On the first two processors, the grid has been given a one-dimensional decomposition, while on the last four the same grid has been given a two-dimensional decomposition. The Router allows processor 0 to know that during a send, it must send four points to processor 2 and during a receive it will receive four points from processor 2. Processor 0's Router also contains information about the shared points on processor 3 (not shown).

The Router also contains an integer ID uniquely identifying the model intended to be the partner in any communication. This integer ID is used to look up the other model's processors and their global MPI rank using a lookup table created as part of MCT initialization.

Once a Router has been initialized and an AttributeVector has been filled with new data to be sent, the $M \times N$ transfer is accomplished with a matched pair of calls analogous to MPI message passing

```
MCT_Send(Model1_AttributeVector,Model1_Router)
```

and on the receive side

```
MCT_Recv(Model2_AttributeVector, Model2_Router)
```

In MCT, the AttributeVector takes the place of the buffer address in MPI communication routines. The Router, "pointing" to another model and its decomposition of the grid, takes the place of the destination/source MPI rank.

To avoid latency costs, MCT_send packs all attribute values (temperature, wind, etc.) into one message for a given set of grid points destined for a processor. The MCT_send blocks until the underlying MPI sends are complete. MCT also provides nonblocking versions, MCT_Isend and MCT_Irecv, for asynchronous $M \times N$ data transfer.

If the Router indicates a given processor must send or receive a message from/to more than one processor, these messages are posted together through successive calls to MPI_Isend and MPI_Irecv (the blocking MCT_send includes a call to MPI_Waitall). Posting several non-blocking calls at once can, in principle, be faster than explicitly scheduling matching pairs of sends and receives (Gropp et al., 1999).

Although two sides of a communication with a Router must reference the same numbering scheme for a grid, the two grids may differ on the total number of points. This flexibility is useful in CCSM where the land model uses the same global grid as the atmosphere but allocates storage only for land points. As long as those points have the same index in their respective GSMaps, a Router can still be constructed and MCT can exchange gridded data between the two models. A similar situation occurs in the ocean model, which has a grid defined over the entire globe but allocates memory only for the ocean points.

3.3 COMMUNICATION SUPPORT FOR INTERPOLATION

In previous versions of the CCSM coupler, interpolation of data was trivial because the single-node coupler held all data points for two grids and their interpolation weights in the same memory. The introduction of distributed-memory parallelism to the coupler creates new requirements for parallel data transfer within the coupler.

As discussed in Section 2.2, interpolation is performed as a sparse-matrix–vector multiply. For example, interpolating data from the atmosphere model's grid to ocean model's grid is

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

The atmosphere vector **A** is the input, or source, while the ocean vector **O** is the output, or destination. The matrix **W** is sparse, and in CCSM only the non-zero elements and their locations are stored. MCT provides a data type called the SparseMatrix to store the weights and their row and column indices using an AttributeVector. MCT also provides a matrix-AttributeVector multiply method that takes advantage of the storage order in the AttributeVector. All attributes for a grid point are interpolated before moving to the next point, thereby allowing cache reuse of the attribute data (Larson et al., 2005).

In the parallel coupler, both the source and destination vectors have a decomposition that is set independently of any consideration for interpolation. Some additional communication is required to obtain the weights, source, and destination points all on the same processor. MCT's tools for doing this are based on the GSMap and AttributeVector. First, just as the GSMap defined the decomposition of a numerical grid based on a sequential numbering of its points, the vectors on either side of the matrix equation above are also serializations of these numerical grids. MCT assumes these two serializations are the same: the row number in the matrix corresponds to the index value of the grid numbering used in the GSMap for the source grid's decomposition, and the column number is the index value of the destination grid. This relationship is crucial to MCT's support for parallel interpolation.

3.3.1 Initialization of Parallel Interpolation The first task is to distribute the non-zero interpolation weights from the matrix **W**, which in CCSM are calculated offline by using SCRIP and read in from files. CCSM does not yet use parallel I/O, and in most cases input data are read from the root node and scattered. MCT provides methods to derive a decomposition for the sparse-matrix elements and scatter weights into this decomposition. The source and destination vector decompositions imply two possible decompositions of the matrix elements: by the decomposition of the rows (source) or by the columns (destination). Both decompositions are equally valid, and MCT provides methods for completing the sparse-matrix multiply in each case.

In a destination-based decomposition, entire columns of non-zero weights are scattered to the coupler processors according to the destination vector's decomposition, as shown in Figure 5(a). MCT's Attribute-Vector_Scatter is used for this operation. This scatter method takes an AttributeVector on one processor that contains values for all the points of a grid and sends them to locally sized AttributeVectors on each processor according to the decomposition described in the destination vector's GSMap. This is analogous to MPI_ScatterV.

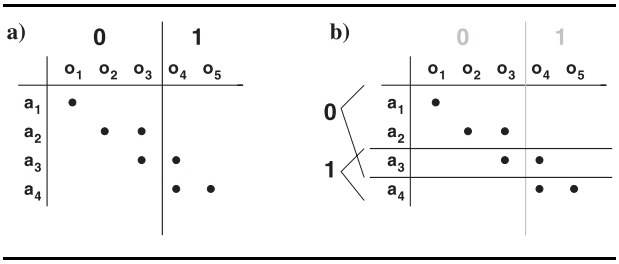

Fig. 5 Steps in determining the decompositions involved in parallel interpolation. See text.

In this case, however, the GSMap is used to determine the destination instead of the MPI rank. This scatter is performed once at startup in CCSM.

The additional communication required to make the interpolation data-local in this case brings the necessary source points to each processor. The necessary source points are the elements from **A** needed to calculate each point in **O** local to a processor. These points can be determined from the non-zero elements of the scattered sparse matrix. Since MCT stores the non-zero matrix element along with its row and column number, the column-scattered SparseMatrix can be examined to collect which corresponding row points are needed on that processor as shown in Figure 5b. This information, a set of row numbers that are also grid point numbers and a processor rank, is sufficient to build a second GSMap for the source grid. The MCT routine SparseMatrixToXGlobalSegMap derives a GSMap for **A** (or **X**) from the column-decomposed **W**. Constructing this GSMap is also a one-time initialization cost in the model. Note that this derived GSMap may contain duplicated points, as shown in Figure 5(b), because the same physical source point may be needed for multiple destination points that may be on different processors. The MCT GSMap allows this situation, and MCT Routers and transfer methods can move data between duplicated and distinct decompositions. These structures and methods allow MCT to position exactly the points needed for interpolation with a minimum in both the number and size of messages.

3.3.2 Parallel Interpolation Before each interpolation, the data in **A** must be moved from their coupler-defined default decomposition to the sparse-matrix derived decomposition. MCT provides the Rearranger class for this kind of data transfer. Rearranger transfers data within a group of processors between two different decompositions. This approach is in contrast to MCT_ Send and MCT_Recv, which move data between disjoint sets of processors. Rearranger takes data in one AttributeVector with an associated GSMap and rearranges it into another AttributeVector associated with a different GSMap all within a group of processors.

In the CCSM coupler, two copies of **A** are necessary because the coupler performs additional computations with atmosphere data where duplicate points, such as those in the sparse-matrix derived decomposition, would be problematic. If this were not the case, the atmosphere could send data directly to the sparse-matrix derived decomposition using a Router, and the rearrange step could be skipped.

A complete atmosphere–ocean parallel interpolation with matrix elements scattered according to the ocean decomposition proceeds as follows. First, the atmosphere data are sent to the coupler by using the $M \times N$ MCT_Send and are stored in an AttributeVector. A call to Rearrange rearranges the atmosphere data into a second AttributeVector with the interpolation-ready decomposition derived from the scattered matrix elements. This AttributeVector is the input in a call to the data-local MCT AttributeVector-SparseMatrix multiply routine. This call interpolates data for all attributes and stores the output in an ocean-resolution AttributeVector, which can then be used in the coupler for additional computation or passed to the ocean model.

Using the same parts of MCT, the interpolation can be done with the matrix weights distributed according to the source vector's decomposition. In this case, the source data are left in place, and a portion of the interpolation is performed on each processor. An intermediate AttributeVector and GSMap are created to hold the output, which now may be only a portion of the final destination point's value. An optional argument to the Rearrange call adds the partial sums while forming the final output AttributeVector.

4 Results

We investigated the performance of MCT routines involved in parallel communication and interpolation. The platform used 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. Portland Group Fortran was used for compilation.

For the timings below, we used a test application that exercises MCT data types in the context of a real climate simulation consisting of an atmosphere model, an ocean model, and a coupler. The atmosphere model uses the same horizontal grid as CCSM3, which covers the globe with 64 latitudes and 128 longitudes. The ocean model also uses the same grid as the CCSM3 ocean model, which contains 384 latitude and 320 longitude points covering the globe. As in CCSM, the mapping weights were

calculated off-line by using SCRIP. The grid point numbering scheme for both grids is as shown in Figure 2; the southernmost point on the western boundary is point number 1, and subsequent points are numbered from west to east and south to north. As in CCSM3, the atmosphere, ocean, and coupler run on distinct sets of processors, and the coupler contains a decomposition and storage for data on both the atmosphere and ocean grids. In most cases, the number of coupler processors is varied from the two or eight processors used in the released version of CCSM3 to 32 processors.

4.1 ROUTER INITIALIZATION

Since the Router plays such an important role in MCT parallel communications, we timed the initialization of a Router under several possible decompositions. The GSMap description of a decomposition can differ based on both the number of processors and the strategy. A typical decomposition in the atmosphere can be imagined by placing the grid shown in Figure 2 over a map of the world. The decomposition shown in Figure 2 is called latitudinal (or by-row) strategy because it divides the world into latitude bands. With the grid numbering shown in Figure 2, a latitudinal decomposition would result in one segment per processor. The orthogonal decomposition to Figure 2 is longitudinal (or by-column) and, with the same grid numbering scheme, would yield one segment per latitude per processor. A combination of those two strategies, dividing the grid into latitude–longitude squares, is called checkerboard (row–column), or Cartesian (Figure 4, right-hand side), and would yield one segment per latitude owned by a processor. A 2 × 2 checkerboard decomposition of the 64 × 128 atmosphere grid would yield a total of 128 segments, 32 for each processor. The GSMap description of the decomposition differs for each strategy and each processor count.

Since the Router is constructed between two GSMaps, the full performance space is four-dimensional, two for the decomposition strategy and two for the processor count. A small portion of that space is considered in Figure 6. Figure 6 shows time to construct a Router in the coupler between the coupler's decomposition (local GSMap) and the atmosphere's decomposition (remote GSMap) of the atmosphere grid. This Router is used by the coupler to send or receive data to and from the atmosphere. The timing considers only the calculation of the Router after the GSMaps have been exchanged. The total time for 10 calls to the Router initialization routine was measured and the maximum time over all processors plotted.

Figure 6 shows the combined effect of changes to the two GSMaps involved in a Router initialization on the time to calculate the Router. For the remote GSMap received from the atmosphere, the number of atmosphere processors is fixed at 16, but the strategy varies between latitudinal (circle), longitudinal (square), and checkerboard (diamond). The grid point numbering scheme is as shown in Figure 2. For 16 atmosphere processors the total number of segments for latitudinal is 16, for longitudinal it is 1024 (64 × 16), and for the checkerboard decomposition it is 256. For the coupler's local GSMap, the strategy is fixed at latitudinal, but the number of processors, and thus the total number of segments, varies from 1 to 32.

The Router initialization algorithm searches first over all segments in the local GSMap looking for those it owns. For each segment it owns, it then loops over all the segments of the remote map looking for a match. Scaling is nearly linear for all decomposition combinations between two and eight processors because the local segment length decreases as processors are added. Some superlinear scaling occurs from eight to 16 processors for the latitudinal decomposition because at 16 processors the coupler and atmosphere GSMaps are exactly the same and it is easy to find matching segments. The longitudinal remote GSMap is the slowest because this decomposition is completely orthogonal to both the grid point numbering scheme used for the grid and the latitudinal decomposition used in the coupler that is building the Router. The combination of 32 possible segments in the local decomposition and 1024 segments in the remote decomposition leads to the longest initialization time. This suggests that, whenever possible, one should choose a grid point numbering scheme and decomposition strategy that minimizes the total number of segments. Also, decompositions with very different strategies lead to increased initialization times.

4.2 $M \times N$ DATA TRANSFER

The performance of MCT_Send in the atmosphere component of the MCT test program is shown in Figure 7. In our test model, the atmosphere AttributeVector has 17 total fields to send, a typical number for a climate model like CCSM3. The atmosphere grid contains 8192 points. Assuming 64-bit real number representation, roughly a megabyte of information must be sent out of the atmosphere each coupling time-step (every simulated hour in CCSM3). The focus on the send, rather than the receive, operation is arbitrary; similar results are obtained if the receive is measured (not shown).

As in the Router initialization, the performance space is four-dimensional. The performance of an $M \times N$ data transfer between components on disjoint sets of processors will depend on the decomposition strategy and the number of processors on each side of the transfer. Both the per-message size and the number of messages can

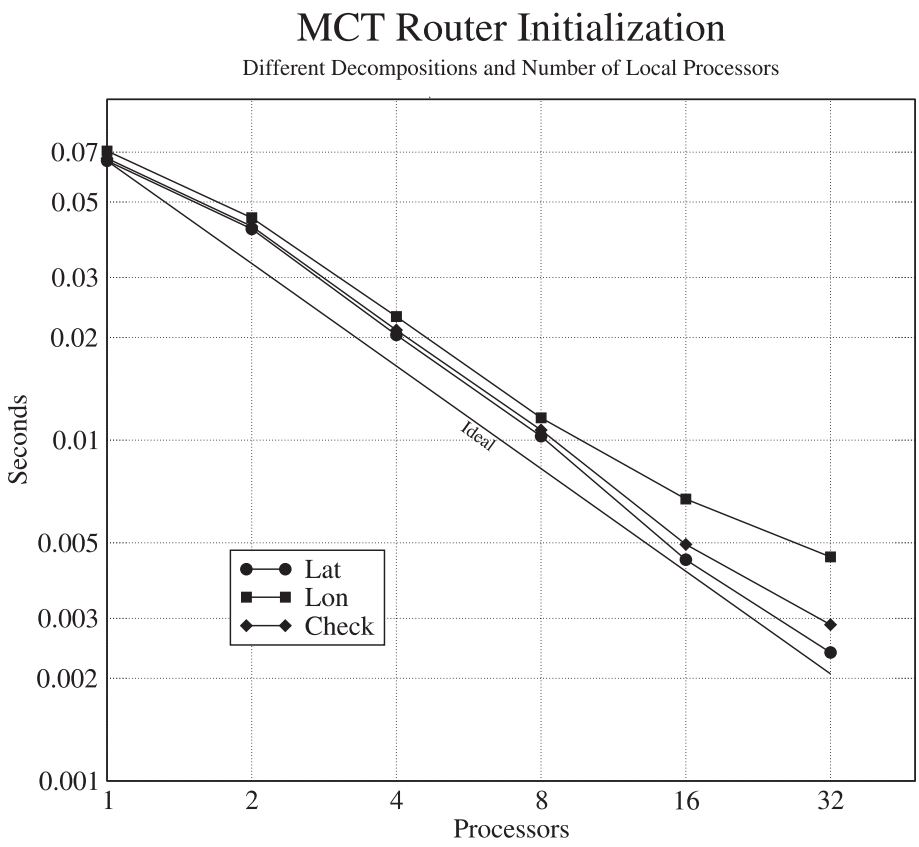

Fig. 6 Total time for 10 calls to the MCT Router initialization routine for different pairs of GlobalSegMaps.

change if the decomposition or number of processors changes. A portion of this performance space is examined in this section.

Figure 7(a) shows the cumulative time for 100 calls to MCT_Send from an atmosphere on 16 processors to a coupler with a varying number of receiving processors. A barrier was placed before the send to decrease the effects from load imbalance in the rest of the MCT test application. The minimum over all processors is shown. The coupler decomposition strategy is fixed at latitudinal, while the atmosphere uses two decompositions, latitudinal (circles) and checkerboard (squares).

For the processor counts considered, as the number of receiving processors increases, the total time decreases. In the latitudinal case, the decomposition strategy is the same as in the coupler, and performance is more predictable than if the send is from a checkerboard decomposition. As in the Router initialization (Section 4.1), when the decompositions are very different, there is an increased cost.

Figure 7(b) shows the results for the same measurement except that the atmosphere is fixed at 32 instead of 16 processors. For this case, communication time again decreases as the number of processors on the receive side increases. When comparing the two latitudinal curves, there is a minimum when the number of processors and the decomposition strategies match exactly (16 processors for Figure 7(a) and 32 for Figure 7(b)). The 32-processor send measurement is faster at all receive processor counts than is the 16-processor send, which shows that MCT can realize some performance gains when processors are added to either side of an $M \times N$ communication.

The departure of the curves from ideal speedup is caused by accumulated latency costs. While the size of each

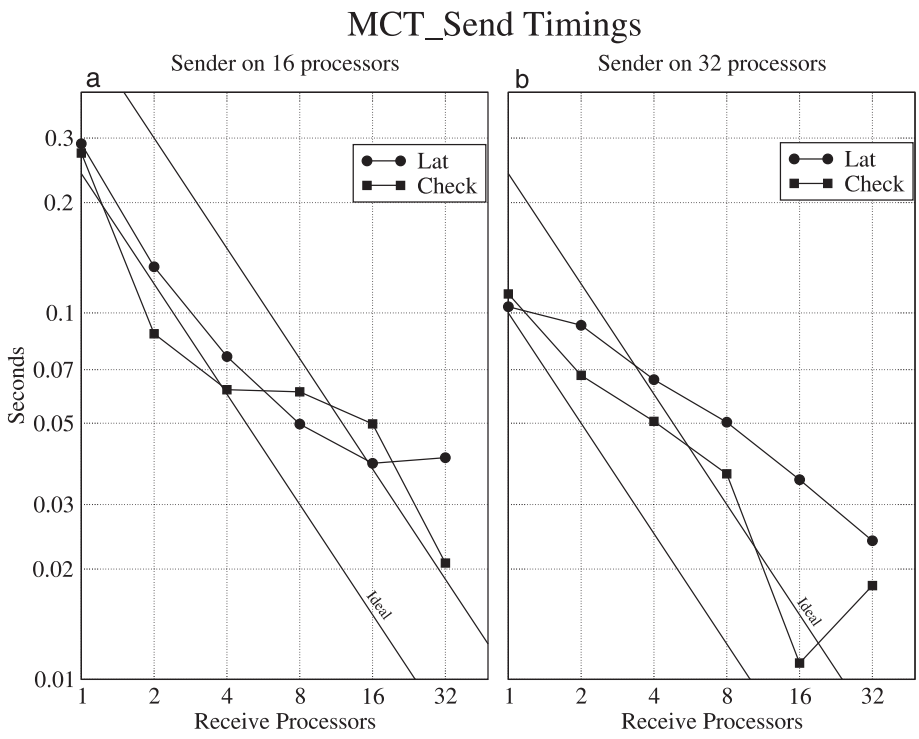

Fig. 7 Total time for 100 $M \times N$ sends of data from one model to another. The Sender is given two different decompositions and either 16 (a) or 32 (b) processors. The Receiver decomposition is fixed (latitudinal), but the processor count varies from 1 to 32.

message decreases as the number of processors increases, there are also more messages to send. The maximum number of messages will equal the product of the number of processors on each side of the communication. This maximum is reached when the decomposition strategies on each side are orthogonal. The checkerboard decomposition contains some orthogonality to the receiver's latitudinal decomposition, and this probably accounts for the different shapes of the checkerboard and latitudinal curves in Figure 7. At the extreme case of very high numbers of processors and very different decompositions, the number of messages could saturate the available bandwidth. In that case, it may be necessary for the coupled model programmer to adjust the decomposition on one side to match the other. If the decompositions, and M and N, are identical, only one message per processor pair is required.

The latency cost of sending a message was anticipated in the design of MCT's parallel transfer routines. Instead of sending one message for each attribute in an AttributeVector, MCT_Send first copies all the grid points and fields destined for a processor from the AttributeVector into an internal buffer. This copy cost was also measured and was between one and two orders of magnitude less than the transmission costs shown in Figure 7 at all configurations tested. While the copy cost increased as the number of messages increased, it was still an order of magnitude less than the smallest transmission time shown in Figure 7.

4.3 PARALLEL INTERPOLATION

The performance of MCT's parallel interpolation functions is presented in Figures 8 and 9. In Section 3.3 we explained how parallel interpolation is divided into two phases: a rearrangement of data for the interpolation and the matrix-multiply doing the interpolation. In this section we consider the cost of both phases and how they vary for number of processors and decompositions.

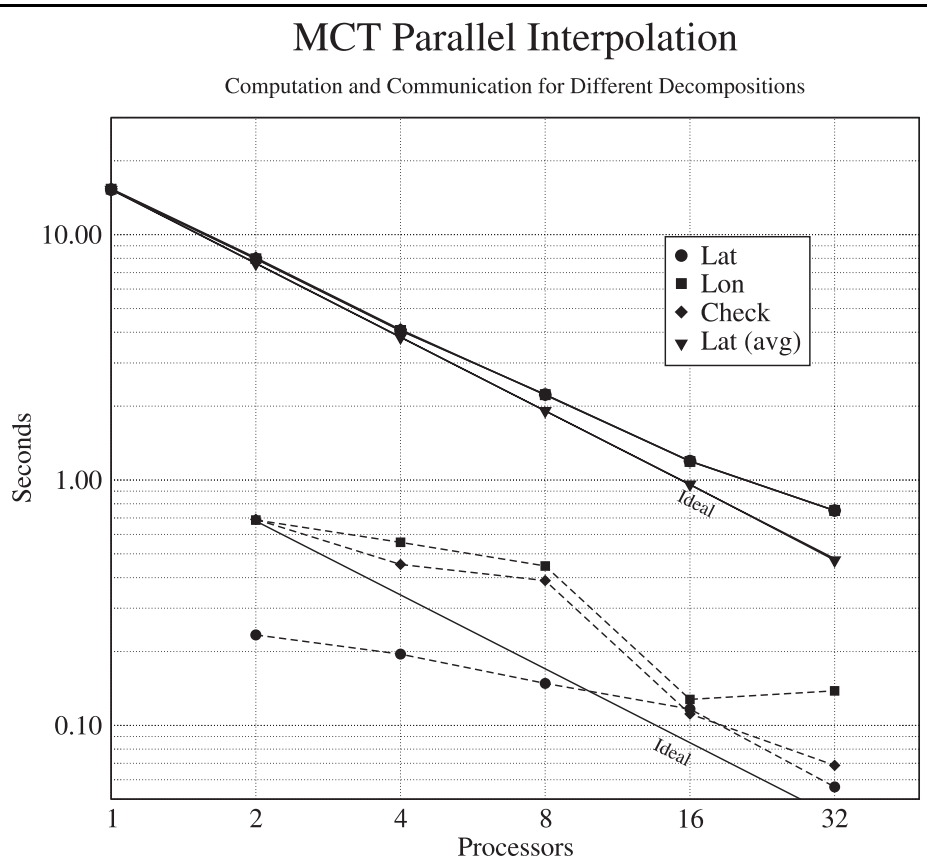

Fig. 8 Total time for 10 calls to MCT parallel interpolation routine with different decompositions of the source grid (see text). The destination grid decomposition is fixed at latitudinal. Source grid resolution is lower than the destination grid. Both the computation (solid) and communication (dashed) costs associated with interpolation are shown. For computation cost, the maximum over all processors is plotted. For the latitudinal decomposition, the average cost is also shown (triangles). Ideal scaling curves are provided for comparison.

While the previous measurements required two components, interpolation is performed entirely in one component, in this case the coupler of the MCT test application. As in the previous sections, the decomposition strategy is still an important factor in performance. Figure 8 shows the performance of the parallel interpolation routine when transforming data from the atmosphere to the ocean grid subject to changes in the decomposition of the atmosphere grid and the number of processors in the coupler. The destination ocean grid decomposition is fixed at latitudinal in all cases. The solid line shows the total cost of the compute-only part of the interpolation for 10 calls to the MCT parallel interpolation subroutine for three different decompositions of the source atmosphere grid. All three decompositions have the same cost because, after rearrangement, the computation is identical. The maximum over all processors is plotted in Figure 8 and shows some departure from ideal scaling because the work load is distributed according to the ocean decomposition and not according to the number of matrix weights, and therefore there is a load imbalance. The average time over all processors, shown for the latitudinal decomposition only in Figure 8 (other decompositions are similar), scales linearly.

The cost of the communication of data between the atmosphere's user-determined decomposition and the

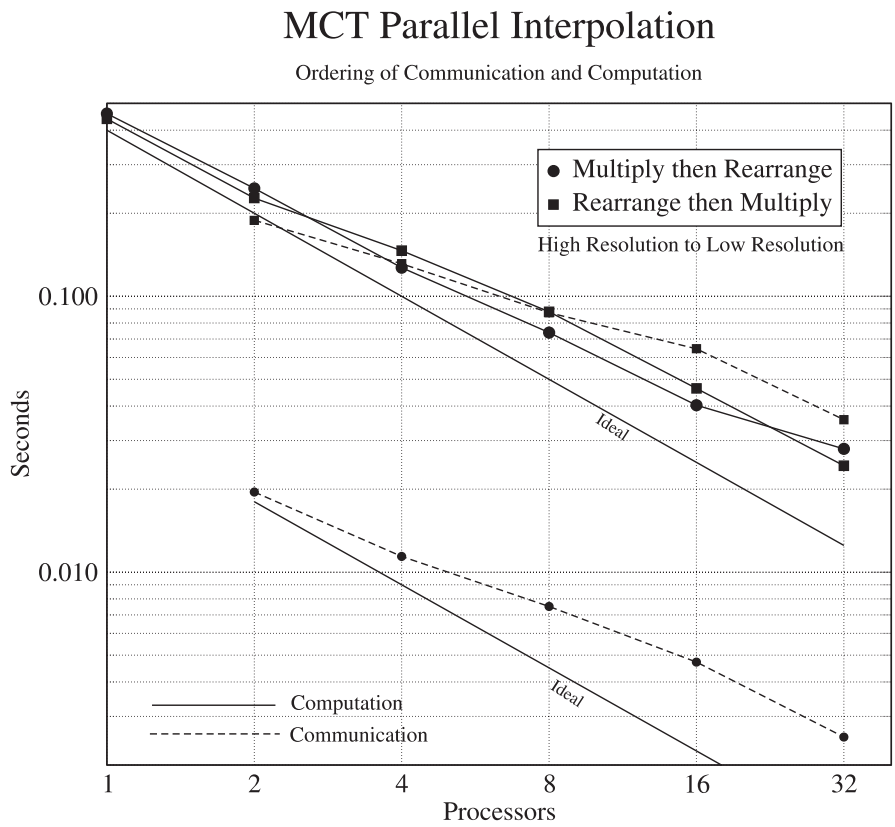

Fig. 9 Total time for 10 calls to MCT parallel interpolation routine with different ordering of communication and computation. Source grid is higher resolution than destination grid. Both the computation (solid) and communication (dashed) costs associated with interpolation are shown. Ideal scaling curves are also provided for comparison.

matrix-multiply derived decomposition within the coupler is plotted by using dashed lines in Figure 8. Some of the behavior seen in previous sections appears again for this measurement. The latitudinal decomposition has the smallest communication cost because it matches the sparse-matrix-derived decomposition, which in turn was based on the latitudinal decomposition of the destination ocean grid. Rearranging data from a longitudinal decomposition to the sparse-matrix derived decomposition has the highest communication costs, and the checkerboard decomposition is close to the cost of the longitudinal. As for the send performance (Section 4.2), the difference in communication costs is caused by the difference between the decompositions.

The cost of communication is around an order of magnitude less than the interpolation at nearly all processor counts. The ratio of computation to communication decreases as the processor count increases because the amount of work remains the same while the total number of messages (and their associated latency costs) increases. Like the Router and MCT_Send, the Rearranger keeps communication costs to a minimum by sending only the points necessary to complete the interpolation and sending all attributes in a single message. The Rearranger performs an in-memory copy for those points located on the same processor in the two different decompositions so processors can avoid sending MPI messages to themselves.

MCT allows the order of communication and computation to be reversed (Section 3.3) so that communication can be performed on the coarser grid. The reason for including this capability is shown in Figure 9. Figure 9

shows the cost of the communication and computation phases of interpolating data from the ocean to the atmosphere grid for two different orderings of communication and computation. Multiply then Rearrange means that first interpolation is performed by using whatever ocean points are available in the user-defined latitudinal decomposition (solid line with circles) and then partial sums are reduced to their resident process IDs on the destination grid decomposition (dashed line with circles). The total cost of the parallel interpolation is the sum of the two measurements. As above, the communication is an order of magnitude smaller than the computation for all processor counts.

For the "Rearrange then Multiply" case, ocean data are first rearranged into a sparse-matrix derived decomposition, and then the interpolation is performed. The costs of communication and computation in this case are comparable, and the total cost of parallel interpolation is nearly double the "Multiply then Rearrange" case. The reason rests with the ratio of the number of points in the atmosphere and ocean grids, which is nearly a factor of 10. In "Rearrange then Multiply" a large amount of ocean data must be communicated between coupler processors, while in "Multiply then Rearrange" a smaller amount of atmosphere data needs to be communicated. For the ocean-to-atmosphere interpolation, there is a performance advantage to performing a partial interpolation of data from the ocean to the atmosphere grid and then rearranging and summing the partial results to form the final atmosphere resolution data. This performance advantage was first noted in the Parallel Climate Model (PCM; Bettge et al., 2001), and PCM's unique coupler contained specially written routines to perform a similar operation (Tony Craig, private communication). MCT's general routines for parallel interpolation matches or exceeds the performance of PCM's custom-written routines (Ong et al., 2002).

5 Conclusions

MCT provides efficient, scalable data types and subroutines for the parallel communication problems found in multiphysics parallel coupled models such as the CCSM. If data are stored in a AttributeVector, the numerical grid is numbered sequentially, and the decomposition is described with a GlobalSegMap. MCT contains several methods to accomplish parallel data transfers in intermodel and intramodel communication.

MCT takes a different approach from other $M \times N$ communication solutions by serializing the numerical grid used in the model instead of the memory space of the model's parallel data type. The implicit relationship between the grid's decomposition and the AttributeVector allows MCT to handle any decomposition of any grid at the cost of a data copy and some effort by the programmer to determine the mapping between a model's internal data structures and the AttributeVector.

MCT provides several methods to scatter interpolation weights stored in files for CCSM and read in on the root node of CCSM's coupler. If future versions of CCSM perform parallel I/O or calculate interpolation weights in parallel, the results could be stored in the same SparseMatrix, and the rest of the parallel interpolation methods would still be applicable. Because the specific choice of interpolation method affects the scientific output of a model, MCT currently leaves the calculation of the interpolation weights to the application developer, but future versions will include some support for calculating these weights from an MCT GeneralGrid.

MCT performance was measured on a Linux cluster, and good scaling was observed for the tested parallel methods. Performance is better if two decompositions of the same grid have a similar strategy. This result suggests that the designers of a coupled system should be aware of the decompositions used in the component models and incorporate this information in their coupling strategy to maximize coupled system performance. Data transfer costs of the parallel interpolation routines can grow if the problem size remains fixed while the number of processors increases. This cost can be avoided if the component performing the mapping does not need the source data for other purposes. MCT allows communication on either side of the interpolation calculation, which can lower the overall time required when interpolating from a high- to low-resolution grid.

The overhead of MCT in CCSM is difficult to measure without examining the full performance of this complex parallel application. A performance study of CCSM is beyond the scope of this paper. Experience with MCT within CCSM3, however, shows that the overhead of copying in and out of AttributeVectors is less than 1% of the total time to simulate a day of climate interaction. This low overhead is because coupling is relatively infrequent and the rest of the model's activity, such as radiation calculations and fluid dynamics, is very time-consuming. Other costs of MCT do not count as overhead because they are performing critical functions of the coupler. The copy costs can be avoided if an application adopts AttributeVectors as their internal data type. The total cost of using MCT's parallel interpolation algorithms was as good as or better than hand-written versions for one coupled model (Ong et al., 2002).

Besides the lack of a common data type, the biggest barrier to constructing more coupled systems in the earth sciences and in other fields is that candidate submodels are seldom developed with coupling in mind. The CCSM architecture is a way to build coupled systems with such applications, but the cost is multiple executables and less

flexibility in integration schemes (concurrent versus sequential). The most significant benefit of the Earth System Modeling Framework (Hill et al., 2004) effort is that it is encouraging many modeling groups to refactor their codes for coupling. These groups are cleanly separating initialization and run-time methods and data structures. Once this refactoring is complete, it will be straightforward to construct new coupled models using libraries currently in operational use such as FMS, MCEL, cpl6, and MCT.

Although MCT was created to solve the problems of a parallel coupler for CCSM, MCT contains no earth-science-specific assumptions and could be used to couple any two models that use a numerical grid. MCT also does not require the use of a coupler and can be used to implement direct intercomponent coupling. An MCT_Send can be called directly from an atmosphere to an ocean model provided the ocean model has a decomposition and AttributeVector for the atmosphere data. MCT's parallel methods can also be used for sequentially coupled systems and mixed sequential-concurrent systems such as the Fast Ocean Atmosphere Model (Jacob et al., 2001).

Because MCT is based on MPI and follows the MPI programming model, MCT can support Grid (Foster and Kesselman, 1998) computing by linking to a Grid-enabled version of MPI such as MPICH-G2 (Karonis et al., 2003). A Grid-enabled version of MCT was used to couple a regional-scale coupled ocean-atmosphere system across a Grid (Dan Schaffer, private communication).

Future versions of MCT will include some OpenMP directives for the computationally intensive parts of the code such as the matrix-multiply routines and new data transfer schemes which can account for masking of unnecessary grid points. The robustness of MCT's data transfer routines has already been demonstrated in a real-world application. Over 10,000 years of climate simulation have been performed with CCSM3 (Lawrence Buja, private communication). All the data transfer between models and interpolation in the coupler performed for these production simulations used MCT's $M \times N$ data transfer and parallel interpolation routines.

MCT is available at http://www.mcs.anl.gov/mct.

ACKNOWLEDGMENTS

We thank Tony Craig, Brian Kauffman, Tom Bettge, John Michalakes, Jace Mogill, and Ian Foster for valuable discussions during development of MCT. We also thank Jace Mogill for his improvements to the algorithm for Router initialization. We thank Michael Tobis for providing valuable comments on an early version of this paper. This work was supported by the Climate Change Research Division subprogram of the Office of Biological and Environmental Research, Office of Science, U.S. Department of Energy through the Climate Change Prediction Program (CCPP), the Accelerated Climate Prediction Initiative (ACPI-*Avant Garde*), and the Scientific Discovery through Advanced Computing (SciDAC) Program, under Contract W-31-109-ENG-38. 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.

AUTHOR BIOGRAPHIES

Robert Jacob is a computational scientist in the Mathematics and Computer Science division at Argonne National Laboratory and a fellow in the Computation Institute at the University of Chicago. He has been greatly involved in the development and application of climate models. He is the lead developer of the Fast Ocean Atmosphere Model and co-lead developer of the MCT. His current interests include long-term climate variability, the role of the ocean in climate change, and high performance computing applications to scientific problems. He received a Ph.D. in atmospheric science from the University of Wisconsin-Madison.

Jay Larson is a software engineer in the Mathematics and Computer Science division at Argonne National Laboratory and is a senior fellow in the Computation Institute at the University of Chicago. He has published in the fields of nonlinear dynamics, plasma physics, climate, weather forecasting, and high-performance computing. In recent years, his work has focused primarily on the development of high performance software infrastructure for the earth system modeling community, most notably as co-lead developer of the MCT. He received a Ph.D. in plasma physics from the College of William and Mary in Virginia.

Everest Ong is a software developer in the Mathematics and Computer Science Division at Argonne National Laboratory. Everest is a co-developer of the MCT and assisted in coding, verification and performance analysis of the CCSM coupler software. He is also involved with building software components for the Common Component Architecture (CCA). His interests also include paleoclimate and statistical climate analysis. Everest received a B.Sc. degree in geophysical sciences from the University of Chicago.

NOTES

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

2 We 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`.

References

Beckman, P. H., Fasel, P. K., and Humphrey, W. F. 1998. Efficient coupling of parallel applications Using PAWS. *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computation*, Chicago, IL, July.

Bettencourt, M. T. 2002. Distributed model coupling framework. *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing*, Edinburgh, UK, July, pp. 284–290.

Bettge, T., Craig, A., James, R., Wayland, V., and Strand, G. 2001. The DOE Parallel Climate Model (PCM): the computational highway and backroads. *Proceedings of the International Conference on Computational Science (ICCS) 2001*, San Francisco, CA, May, *Lecture Notes in Computer Science*, Vol. 2073, Springer-Verlag, Berlin, pp. 148–156.

Collins, W. D. et al. 2005. The Community Climate System Model: CCSM3. *Journal of Climate* in press.

Craig, A. P., Kaufmann, B., Jacob, R., Bettge, T., Larson, J., Ong, E., Ding, C., and He, H. 2005. Cpl6: the new extensible high performance parallel coupler for the Community Climate System Model. *International Journal of High Performance Computing Applications* 19(3).

Drummond, L. A., Demmel, J., Mechose, C. R., Robinson, H., Sklower, K., and Spahr, J. A. 2001. A data broker for distributed computing environments. *Proceedings of the 2001 International Conference on Computational Science (ICCS) 2001*, San Francisco, CA, May, *Lecture Notes in Computer Science*, Vol. 2073, Springer-Verlag, Berlin, pp. 31–40.

Edjlali, G., Sussman, A., and Saltz, J. 1997. Interoperability of data-parallel run-time Libraries. *International Parallel Processing Symposium*, Geneva, Switzerland.

Foster, I. and Kesselman, C. 1998. *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Mateo, CA.

Geist, G. A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. 1994. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA.

Geist, G. A., Kohl, J. A., and Papadopoulos, P. M. 1997. CUMULVS: providing fault tolerance, visualization and steering of parallel applications. *International Journal of High Performance Computing Applications* 11(3):224–236.

Gropp, W., Lusk, E., and Skjellum, A. 1999. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition, MIT Press, Cambridge, MA.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., da Silva, A., and the ESMF Joint Specification Team. 2004. The architecture of the Earth System Modeling Framework. *Computing in Science and Engineering* 6:18–28.

Jacob, R., Schafer, C., Foster, I., Tobis, M., and Anderson, J. 2001. Computational design and performance of the Fast Ocean Atmosphere Model. *Proceedings of the International Conference on Computational Science (ICCS) 2001*, San Francisco, CA, May, *Lecture Notes in Computer Science*, Vol. 2073, Springer-Verlag, Berlin, pp. 175–184.

Jiao, X., Campbell, M. T., and Heath, M. T. 2003. Roccom: an object-oriented, data centric software integration framework for multiphysics simulations. *Proceedings of the 17th Annual ACM International Conference on Supercomputing*, San Francisco, CA, June.

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

Karonis, N., Toonen, B., and Foster, I. 2003. MPICH-G2: a grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing* 63(5):551–563.

Keahey, K. and Gannon, D. 1997. PARDIS: a parallel approach to CORBA. *Proceedings of the High Performance Distributed Computing Conference*, Portland, OR, pp. 31–39.

Keahey, K., Fasel, P., and Mniszewski, S. 2001. PAWS: collective interactions and data transfers. *Proceedings of the High Performance Distributed Computing Conference*, San Francisco, CA.

Larson, J., Jacob, R., and Ong, E. 2005. The Model Coupling Toolkit: a new Fortran90 toolkit for building multiphysics parallel coupled models. *International Journal of High Performance Computing Applications* 19(3).

Lee, J. and Sussman, A. 2004. Efficient communication between parallel programs with InterComm. CS-TR-4557 and UMIACS-TR-2004-04, University of Maryland, Department of Computer Science and UMI-ACS.

MPI Forum. 1994. MPI: a message passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing* 8(3/4):159–416.

Ong, E., Larson, J., and Jacob, R. 2002. A real application of the Model Coupling Toolkit. *Proceedings of the International Conference on Computational Science*, Amsterdam, the Netherlands, April, *Lecture Notes in Computer Science*, Vol. 2330, Springer-Verlag, Berlin, pp. 748–757.

Perez, C., Priol, T., and Ribes, A. 2003. A parallel COBRA component model for numerical code coupling. *International Journal of High Performance Computing Applications* 17(4):417–429.

Ranganathan, M., Acharya, A., Edjlali, G., Sussman, A., and Saltz, J. 1996. Run-time coupling of data-parallel programs. *Proceedings of the International Conference on Supercomputing*, Philadelphia, PA.