Chapter 3
The Model Coupling Toolkit

Robert Jacob and Jay Larson

3.1 Introduction

The Model Coupling Toolkit (MCT) provides datatypes and methods for creating parallel couplers and parallel coupled models out of one or more models of physical systems. MCT handles common coupling tasks in a distributed memory parallel application.

Development of MCT began in 2000 under the U.S. Department of Energy's Office of Science and an 18-month program called the "Accelerated Climate Prediction Initiative (ACPI)—*Avante Garde*". A goal of this project was to enable the Community Climate System Model (CCSM, see Volume 1 of this series) to better exploit the then new microprocessor-based parallel computers. In particular, the coupler within CCSM (then called CSM) was not parallel while other components were developing limited parallelism. Communicating data required gathering and scattering all data to/from one processor creating a bottleneck to performance. The CCSM community had already settled on a hub-and-spoke design for future versions. CCSM was also using multiple numerical grids and contemplating more in the future. Thus MCT development focused on creating parallel data types, domain decomposition descriptors and parallel communication and interpolation methods for generic grids. Another goal was to provide a general solution to those problems, one that could be used in coupled applications from other fields. Climate models, and other coupled models, are supercomputing applications that require high performance to achieve simulation speed sufficient for their intended application. Supercomputers

R. Jacob (✉)
Argonne National Laboratory, Argonne USA
e-mail: jacob@mnsc.anl.gov

J. Larson
Research School of Computer Science, The Australian National University, Canberra Australia
e-mail: larson@mnsc.anl.gov

come in many varieties so the codes must also be portable. Thus MCT was designed with an eye toward high-performance portability. Other software products developed for similar problems were surveyed but none could satisfy all these requirements.

MCT's designers had previous experience in developing a parallel coupler for a climate model (Jacob et al. 2001), which used a field regridding technique similar to the *exchange grid* (Sect. 5.4), and parallel analysis tools for data assimilation (Larson et al. 1998). The knowledge and techniques learned in those efforts greatly influenced the design of MCT. For example, the work with the Physical-space Statistical Analysis System (PSAS) encouraged the designers to write MCT entirely in Fortran90. Fortran90 has just enough object-oriented features to make it possible to write a class library and, because MCT's primary target applications are almost universally written in Fortran90, it was possible to avoid language interoperability issues. The experience with FOAM's coupler and examination of CCSM's coupler made it clear that MCT should focus on the hardest and most general problem: storage and communication of coupling fields in parallel between generic grids and decompositions.

MCT's first version was complete by November of 2002 and it was used as the basis for the coupler in CCSM3, released in June of 2004. This chapter presents an overview of the MCT programming philosophy and some of the features available in the toolkit.

3.2 Architectural Overview and Programming Philosophy

Elements of MCT's programming philosophy are found in many software applications. In general, MCT values flexibility over completeness and follows the "less is more" principle. For example, it was decided early on that MCT would not include a ready-to-use coupler application. From coupled models that existed at the time, it was clear that there were many choices regarding timestep, model sequencing, interpolation and physical flux calculations implemented in each model and the reasons behind these choices were more scientific then structural. In other words, the coupler is *part of the science* described in the model and must also be designed for readability: its inner workings must be easily comprehensible and it must also be reconfigurable as experimentation needs require. Trying to provide both a ready-to-use, universal coupler as well as readability and configurability are conflicting goals. The better way to simplify coupled model development would be to develop a set of programming guides that, with the help of a small but powerful set of software tools, would greatly reduce the development time of a new coupled model's coupler.

MCT is the small set of software tools. Architecturally, MCT is a Fortran90 library and calls to MCT are added to a user's program through F90 *module use* and compiled by linking with the library. MCT has three main pieces. At the lowest level is the Message Passing Environment Utilities (MPEU) which was part of PSAS. MPEU provides useful fortran-based utilities, e.g. sorting, multiprocessor stdout and stderr, and provides elements of MCT low-level classes. (MPEU can be used separately from

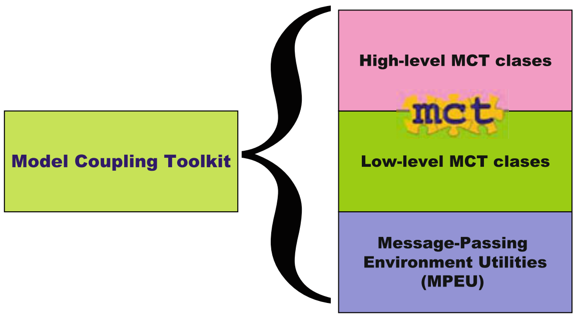

Fig. 3.1 The MCT architecture

MCT.) MCT's low level classes handle the basics of coupling: a component model registry, domain decomposition descriptors, communication schedulers for parallel data transfer, a flexible and indexible data storage type, and parallel interpolation algorithms (which use offline-generated weights). The higher level classes provide useful functions within a coupler such as spatial averaging, time averaging, and merging (Fig. 3.1).

A programming guide for coupled modeling would include the following elements:

- A model ideally should not assume it has sole control of the standard input and use named logical units instead.
- Similarly, a model may not be the only one outputting messages to standard out and text output should include a unique string identifying what model they came from or write to a named file.
- For parallel models that use the Message Passing Interface (MPI), a uniquely named communicator should be used instead of the general MPI_COMM_WORLD one.

At a higher level, models are easy to configure with several coupling strategies if their design follows a pattern that comes from object oriented programming: i.e. each model is separated into an initialization method and a run method (and possibly a "finalize" method.) This same design pattern is advocated by other coupling software (see e.g. ESMF in Sect. 6.5.1). The initialization method of a scientific application often has many duties besides allocating memory; it must read in initial and boundary value data and set values for important physical constants. With these standards, the stand-alone model could have its topmost level rewritten as a relatively small piece of software that calls the initialize, run and finalize methods of the model. Once initialization methods are separated, they can be arranged to be each called in sequence in the new coupled application before the run methods are called.

But for a legacy application, the effort to make an application callable from another application can be as much or even more effort then building the complete coupled model. For example, it can involve taking data structures which exist far down in the calling tree and bringing them up to the run method if they contain coupling state information. For the cases where it is not possible to make a legacy application callable from a new model or follow other of the guidelines above, MCT's methods also support a programming model where MCT communication routines are placed deep in to the model's calling sequences so they can communicate with each other (direct coupling) or with a central, user-created, coupler. Hybrids of these two programming models (some embedded and others separate) are also possible with MCT. Part of MCT's philosophy is to leave these important decisions up to the application developers and the scientific needs of the application.

3.3 MCT Datatypes and Methods

Some of MCT's methods are patterned after the Message Passing Interface (MPI), version 1. MCT uses MPI1 two-sided communication to move data between two models. It is expected, but not required, that the models in the coupled system are parallelized with MPI. A small, serial MPI replacement library is included in MCT so that MCT-coupled applications can be compiled without a full MPI library. Many MPI concepts such as *root* and *rank* are used in MCT with the same meaning. MCT is compatible with MPI2 but does not yet employ its one-sided communication model.

Because of the focus on parallelism one of the low-level classes in MCT is MCT_WORLD. MCT_WORLD contains an internal table of how many models are present in the coupled system and the local and global ranks of the processes they are running on. The result is a registry of all models running in the coupled application. This is initialized once at startup of the model and is referenced internally by MCT's communication routines.

MCT's communication routines provide a solution to what is sometimes called the "M × N problem" (Bertrand et al. 2006) which is the problem of how to transfer a distributed data object from a module running on M processes to another running on N processes. In a parallel coupled model, this would be the problem of getting values held in a one model's internal distributed data type to the other model's distributed type. This is the fundamental problem to solve when building a high-performance parallel coupled model. One of the core classes used to solve this problem is the GLOBALSEGMAP (GSMAP) or *Global Segment Map*. The GSMAP is the MCT datatype for describing a decomposition of a numerical grid or the portion of the grid used in coupling. It provides a linearization of the grid. Each point must be assigned a unique integer value by the user. It is possible to describe any decomposition and grid, including unstructured, this way. Two simple decompositions of a grid are illustrated in Figs. 3.2 and 3.3. The choice of how exactly to number the grid points is left to the user.

The heart of the GSMAP datatype contains three arrays that are sized according to the total number of segments in the decomposed grid. These three arrays contain, for each index, the starting grid point number, the run length of the segment and the rank of the processor the segment resides on (the processor location). For the decomposition shown in Fig. 3.2, the GSMAP arrays would be *start*(1, 11):*length*(10, 10):*pe_loc*(0, 1) while for the decomposition in Fig. 3.3 the arrays would be *start*(1, 6, 3, 8, 11, 16, 13, 18):*length*(2, 2, 3, 3, 2, 2, 3, 3): *pe_loc*(0, 0, 1, 1, 2, 2, 3, 3). Each application needs to construct its own GSMAP. A copy of that GSMAP is available on

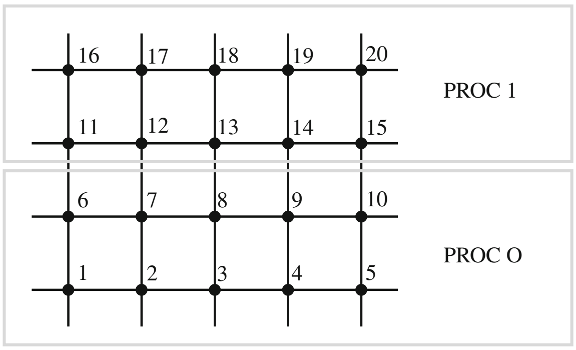

Fig. 3.2 A 2 × 1 decomposition of grid with 20 points. The points have been numbered

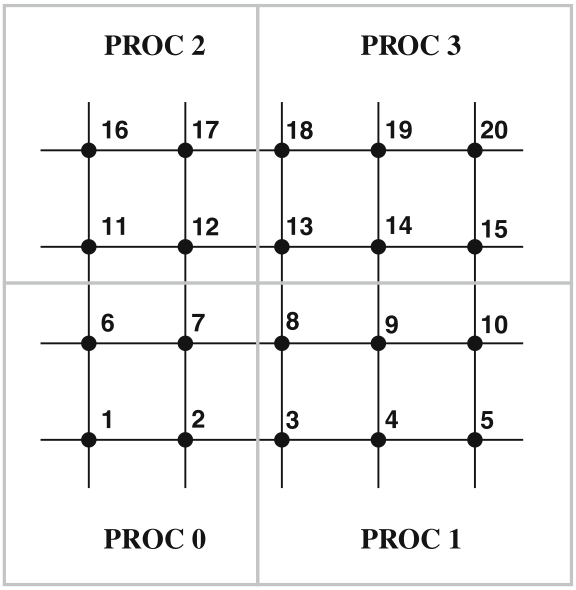

Fig. 3.3 A 2 × 1 decomposition of the same grid in Fig. 3.2

each processor after initialization. This does not present a memory scaling problem because the size of the three integer arrays grows only by the number of segments, not the number of points.

Once the GSMAP is initialized, MCT provides a method for determining and storing a communication schedule that describes for other routines how to transfer data between two different decompositions of a grid. The decompositions could be within the same set of processors, two completely disjoint sets of processors, or on overlapping sets of processors. This data type is called the ROUTER. The only input needed is either two GSMAPs, when a model has two decompositions of the same grid, or one GSMAP and the ID from MCTWORLD of the other component in the communication. The initialization of a router is like the old "handshake" performed by modems: the two models are learning how to exchange data with each other.

An instance of the ROUTER class is returned and this becomes an argument for the communication routines.

One of the biggest obstacles to creating coupled models is that two independently developed models seldom use the same datatype to describe the same physical concept such as a wind field. One might use a simple 3D array while the other uses a derived type. MCT provides a standard datatype called the ATTRVECT or *Attribute Vector*. The *Attribute Vector* acts as a translator between possibly very different data types within a coupled system. One *Attribute Vector* holds all of the data on a local processor that must be communicated. Methods are provided to access individual vectors that may contain data for fields such as temperature, wind or humidity. (The *Attribute Vector* is one of the parts of MCT based on MPEU).

All the data that needs to be communicated to another model must be copied in to an *Attribute Vector*. A second *Attribute Vector* on the other side of the communication receives the data and copies it in to that model's data type. The user must provide the logic to translate between their model datatype and the *Attribute Vector*. That is, they must know how to copy data from a location in the *Attribute Vector* to the corresponding (same physical location, e.g. latitude and longitude) location in their internal data type. However MCT can then handle all the communication for any parallel configuration between the two ATTRVECTS. Once a model can copy its data in and out of an *Attribute Vector*, it can use MCT routines to send data, in parallel, to any other model which has its own interface to the ATTRVECT.

There is an important relationship between the *Attribute Vector* and the global segment map illustrated in Fig. 3.4. When copying between the local ATTRVECT and the model's internal datatype, the MCT user must be mindful of the relationship between the order of the local grid point indices and the order of data in the ATTRVECT. Figure 3.4 shows for example how the "PROC 1" data of the decomposition illustrated in Fig. 3.2 is organized locally in monotonic increasing order of the grid point number (but monotonicity is not strictly necessary).

With a GSMAP, ROUTER and ATTRVECT defined, users can transfer data between two decompositions using either MCT_SEND/MCT_RECV, for transferring data between disjoint sets of processors, or REARRANGER for two decompositions on the same set of processors. The REARRANGER *init* method creates a ROUTER but keeps track of the points that are on the same physical memory and can be directly copied. The run method will then use message passing for those points which are on other processors and local copies for local points. REARRANGER is a collective operation while MCT_SEND/MCT_RECV follow the two-sided message passing model of MPI. Non-blocking versions of the later are also available.

Another common problem in creating coupled models is interpolating data from one grid representation to another. MCT currently supports interpolation as a linear transformation implementable as a sparse matrix-vector multiply. MCT provides the SPARSEMATRIX data type to hold matrix weights and the MATATTRVECMULT to perform the interpolation between data contained in an *Attribute Vector*. MCT does not currently provide methods for calculating the weights because there are potentially many ways to perform that crucial calculation. It is another science aspect that is best left to the coupled model developers. In practice, MCT typically

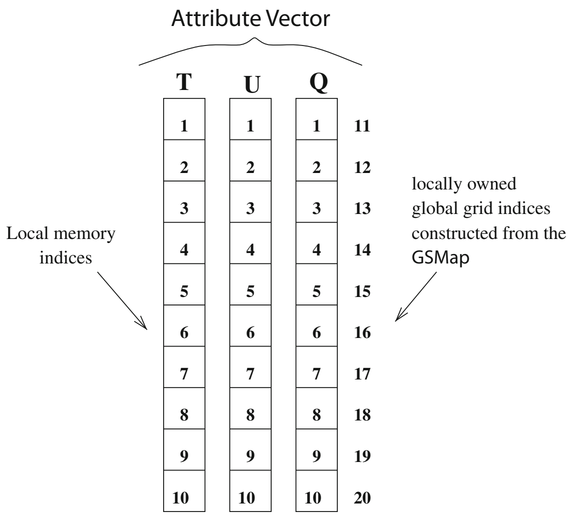

Fig. 3.4 The relationship between the local memory in an ATTRVECT and the grid point numbering contained in the GLOBALSEGMAP for this model. In this example, the ATTRVECT is being used to store three variables: temperature (T), east-west velocity (U) and specific humidity (Q)

uses weights calculated by the Spherical Coordinate Remapping and Interpolation Package (SCRIP, Jones 1999).

The high-level classes in MCT include the ACCUMULATOR for time-averaging of data involved in coupling. This level also includes a set of transformation routines, including: time integration that computes ACCUMULATOR elements from ATTRVECT input; spatial integration routines for performing global sums of data in an ATTRVECT, which are often necessary for checking the results of interpolation; and support for weighting outputs from two or more models that must be sent to a third model as input—a process called *merging*. All of the classes mentioned here and above contain additional routines for functions such as query, and MCT datatype-specific support for standard parallel operations such as broadcast, gather and scatter. Further details can be found in Larson et al. (2005) and Jacob et al. (2005) and the MCT web site.[1]

3.4 MCT Multi-Language Interface

MCT's code base and programming model were based on Fortran modules and derived types, and this approach has been highly successful for Fortran-based scientific software. MCT's long-term goal is to create universally applicable code to support multiphysics and multiscale model construction for all areas of computational science and engineering—a goal that requires a strategy for language interoperability. MCT's philosophy of creating light coupling infrastructure dictated that language interoperability be separated from MCT's core functionality. A layer of interlan-

[1] http://www.mcs.anl.gov/mct.

guage "glue" was created that allows MCT's programming model to be exported to other languages. Below, MCT's interlanguage interfaces are briefly described. A more complete discussion is given by Ong et al. (2008).

MCT is implemented in adherence to the Fortran90 standard, which offers neither a complete standard for array descriptors nor an Application Programming Interface (API) for querying them. This makes interfacing codes in Fortran90 to other programming languages notoriously difficult. Fortran2003 offers a BINDC feature, but at the time MCT's interlanguage bindings were created, this feature was not sufficiently widely available to consider it a portable solution. The interfaces were implemented using the Babel language interoperability tool. Babel supports a variety of programming languages, including Fortran, C, C++, Python, and Java. The MCT multilingual interfaces were semi-automatically generated using Babel, with some intervention to implement references to MCT's code base. MCT APIs for C++ and Python were then automatically generated using Babel. The result is an MCT programming model for the destination languages that looks and feels natural to native programmers of these languages. For example, in MCT's Python API, Fortran module use is replaced with Python package import, Fortran variable declaration is replaced with a null constructor, and method invocation remains unchanged. The performance of MCT's C++ interfaces has been found to be close to their Fortran counterparts. The Python API is significantly slower, but still performs well enough to support a test implementation of the CCSM coupler in Python.

3.5 Conclusions and Perspectives

The most prominent user of MCT continues to be the Community Climate System Model and its developers. CCSM is one of the main climate models in the U.S. and is used for national assessment simulations of the effects of climate change. MCT is the basis of the "cpl6" coupler (Craig et al. 2005) that was part of the third version of the Community Climate System Model (Collins et al. 2006). It is also the basis of "cpl7" in the fourth version of CCSM and the first version of the Community Earth System Model[2] released in April and June of 2010, respectively. "cpl7" is much more flexible in that it allows several different ways for the components to interact (see Volume 1 of this series). The coupled model developers also have exposed more of MCT to its users, recognizing that MCT's simple concepts and functions help to clarify the flow of information in the coupler. Other researches at the National Center for Atmospheric Research are using MCT for more advanced coupled systems such as nesting a regional ocean model in a global ocean model.

Other users of MCT are notable because many were able to use MCT successfully and create new coupled applications while referring only to MCT's API document and some example programs distributed with the code. (MCT currently lacks a *Users' Guide*). These include a model for simulating hurricanes consisting of a weather

[2] http://www.cesm.ucar.edu

model coupled to a regional ocean model developed by researchers at NCAR and the NOAA Pacific Marine Environmental Laboratory and a system for studying coupled atmosphere-ocean processes along coastlines created by researchers at Oregon State University (Warner et al. 2008). Support for MCT users is provided by emailing the developers. The MCT development team has been 2–3 people throughout its history augmented by contributions to the code from users.

MCT's architecture has inspired the design of a coupling API (Larson & Norris, 2007) for the Common Component Architecture (CCA; Bernholdt et al. 2006). CCA's goal is to leverage component-based software engineering techniques to serve the computational science community. The coupling API comprises a set of composable components that can be assembled to create coupler applications, with components for the major types of data processing operations encountered in code coupling, including: component model registration; field, mesh, and domain decomposition data description; parallel data transfer and redistribution; linear transformations of field data; time averaging and integration of field and flux data; and merging. An MCT-based reference implementation of these coupling components is currently under development.

MCT continues to be improved. MCT has recently implemented a hybrid programming model (especially important for machines with low memory per node) using OpenMP in some of its routines. MCT has augmented its key communication routines so that they are scalable to tens of thousands of processors. In the future, MCT will also support internally some widely used methods for calculating interpolation weights and provide better support for unstructured grids. MCT continues to be supported by the US Department of Energy's Office of Science under its Scientific Discovery through Advanced Computing program. The MCT approach has been shown to be both flexible and easy to learn and should continue to grow in use.

References

Bernholdt DE, Allan BA, Armstrong R, Bertrand F, nneth Chiu K, Dahlgren TL, Damevski K, Elwasif WR, Epperly TGWE, Govindaraju M, Katz DS, Kohl JA, Krishnan, M, mfert GK, Larson JW, Lefantzi S, Lewis MJ, Malony AD, Mclnnes LC, Nieplocha J, Norris B, Parker SG, Ray J, ende SS, Windus TL, Zhou S (2006) A component architecture for high-performance scientific computing. Int J High Perform Comput Appl 20(2):163–202. doi 10.1177/1094342006064488

Bertrand F, Bramley R, Damevski K, Kohl J, Bernholdt D, Larson J, Sussman A (2006) Data redistribution and remote method invocation for coupled components. J Parallel Distributed Comput 66(7):931–946

Collins WD, Bitz CM, Blackmon ML, Bonan GB, Carton JA, Chang P, Doney SC, Hack JJ, Henderson TB, Kiehl JT, Large WG, McKenna DS, Santer BD, Smith RD (2006) The community climate system model: CCSM3. J Clim 19(11):2122–2143

Craig AP, Jacob RL, Kauffman BG, Bettge T, Larson J, Ong E, Ding, C, He H (2005) Cpl6: The new extensible high-performance parallel coupler for the community climate system model. Int J High Perform Comput Appl 19(3):309–327

Jacob R, Schafer C, Foster I, Tobis M, Anderson J (2001) Computational design and performance of the fast ocean atmosphere model. In: Alexandrov VN, Dongarra JJ, Tan CJK (eds).In: Proceedings of 2001 international conference on computational science, Springer-Verlag, pp 175–184

Jacob R, Larson J, Ong E (2005) MxN communication and parallel interpolation in CCSM3 using the model coupling tookit. Int J High Perform Comput Appl 19(3):293–307

Jones P (1999) First- and second-order conservative remapping schemes for grids in spherical coordinates. Monthly Weather Rev 127:2204–2210

Larson J, Norris B (2007) Component specification for parallel coupling infrastructure. In: Gervasi O, Gavrilova M (eds) Proceedings of the international conference on computational science and its applications (ICCSA 2007) Lecture notes in computer science, vol 4707. Springer-Verlag, pp 55–68

Larson J, Jacob R, Ong E (2005) The model coupling toolkit: a new Fortran90 toolkit for building multi-physics parallel coupled models. Int J High Perform Comput Appl 19(3):277–292

Larson JW, Guo J, Gaspari G, da Silva A, Lyster PM (1998) Documentation of the physical-space statistical analysis system (PSAS) Part III: The software implementation, Technical report DAO Office Note 98-05, NASA/Goddard Space Flight Center, Greenbelt, Maryland

Ong ET, Larson JW, Norris B, Jacob RL, Tobis M, Steder M (2008) A multilingual programming model for coupled systems. Int J Multiscale Comput Eng 6(1):39–51

Warner J, Perlin N, Skyllingstad E (2008) Using the model coupling toolkit to couple earth system models. Environ Modelling Softw 23(10–11):1240–1249