Development of the Simple Coupler “Scup” for Earth System Modeling

Hiromasa Yoshimura and Seiji Yukimoto
Meteorological Research Institute, Tsukuba, Japan

Abstract

We have developed a simple general-purpose coupler “Scup” (Simple coupler) which is usable in earth system modeling with flexible coupling among component models, such as atmosphere and ocean general circulation models and chemical transport models.

Scup provides sufficient functions required for efficient development of the earth system model at Meteorological Research Institute (MRI), and its fundamental design is also applicable for other earth system models. Each process of the component models communicates with each other directly by calling the Scup library, which is necessary for efficiently exchanging a large amount of grid data among the processes of the component models. Scup supports grid transformation for three-dimensional data in addition to that for two-dimensional data with accurate local and global conservation, which is necessary for coupling component models with different coordinates or grids.

Simplicity and easiness to use are the strong points of Scup. All the source
code of Scup is described in Fortran and can be compiled on various platforms with a Fortran 95 compiler and MPI (Message Passive Interface) library, which contributes to good portability of Scup.

1. Introduction

An earth system model has been developed at Meteorological Research Institute (MRI) as a next generation model for climate change projection. The earth system model should represent interactions among climate system components, for example, atmosphere, land, ocean and sea ice, and their subsystem components, such as atmospheric chemical processes and biogeochemical processes. To represent their interactions, field variables and fluxes must be exchanged among the component models. Since each component model is developed as an independent program, a coupler is needed to exchange the variables and fluxes with keeping the independency of each component model in the earth system model.

A coupler should include following functions for general purposes:

- Exchange of data among the component models;
- Management of data to be exchanged;
- Operations on data, such as time average and spatial interpolation.

There are several general-purpose open-source couplers with these functions, for example, OASIS3 (Valcke et. al. 2004b), OASIS4 (Valcke et. al. 2004a, 2004c), ESMF (Hill 2004), cpl6 (Craig 2005) and Fujin (Arakawa 2001), each
of which has its own excellent features.

On the other hand, for the earth system model being developed at MRI, functions, such as:

- Efficient data exchange by direct communication among component models;
- Spatial interpolation for not only horizontal two-dimensional (2-D) but also three-dimensional (3-D) grid transformation, are necessary to exchange a large amount of 2-D and 3-D data among component models with different coordinates or grids. Moreover, to introduce a coupler easily, the coupler should have preferable features such as:
  - All sources are coded by Fortran without using any external C library to compile easily;
  - Only minimum modifications of the component models are required to make them coupled.

However, no coupler among the above-mentioned ones has all of these functions and features. Moreover, since they are large-scale software programs, it will be hard work for us to make them executable in our computing environment and add some functions necessary for our purposes.

Therefore, we have considered that it is more practical to develop a new simple general-purpose coupler “Scup” (Simple coupler) with necessary functions and preferable features for the MRI earth system model.

We started to develop Scup in January 2006 and created a preliminary
version of Scup in April 2006. After that, we improved functionality and efficiency of Scup while the MRI global model development team improved the component models and introduced Scup into them. And then our team completed creating the first version of the earth system model (Yukimoto et al. 2008; manuscript in preparation) in December 2006, where three component models, a coupled atmosphere-ocean model (MRI-CGCM3; Yukimoto et al. 2008), an aerosol transport model (MASINGAR; Tanaka et al. 2003) and a chemical transport model (MRI-CTM; Shibata et al. 2005), were coupled by Scup. At present another built-in coupler is used for coupling the atmosphere model and the ocean model in MRI-CGCM3, but Scup will be used to couple them in the future version of the earth system model.

2. Functions and features of Scup

Scup has necessary functions and preferable features for the MRI earth system model as follows. We expect that the following points are useful not only for the MRI earth system model but also for other earth system models and other kinds of model coupling such as coupling between global and regional models.

(1) For efficient exchange of a large amount of data among component models, each process of the component models directly communicates with each other by calling the Scup library. The efficiency is necessary to exchange a large amount of 2-D and 3-D data, for example, temperature,
wind components, water vapor, CO2, aerosol, ozone and so on. Detailed description about communication among component models is given in Section 3.

(2) Interfaces of the Scup library are designed so that component models would only require minimum modification for coupling them, which makes Scup easy to introduce. See Section 4 for detail about interfaces of the Scup Library.

(3) Model coupling is flexibly controlled by the communication configuration text file, “Scup namelist”, where configuration of coupling, such as variables to be transferred among component models and coupling timing, are described. By modifying “Scup namelist”, it is possible to flexibly change the coupling behavior or the configuration of component models without modifying source code of the models and re-compiling them, which makes the component models easy to use for model users. Detailed description about “Scup namelist” is given in Section 5.

(4) Scup supports three types of "time lag" for model coupling. How models are connected in time is controlled by the "time lag", which is described in Section 6 in detail.

(5) Scup supports 2-D and 3-D grid transformation between models, which is necessary to couple component models with different coordinates or grids, for example, a 120km-mesh atmosphere model, a 200km-mesh aerosol transport model and a 300km-mesh chemical transport model. In 2-D grid
transformation, a scheme with accurate local and global conservation is supported, which is necessary in order not to produce false sources or sinks of substances. Remapping tables for horizontal 2-D grid transformation are separately created by an offline calculation, which contributes to simplicity of Scup itself. Detailed description about grid transformation is given in Section 7.

(6) Scup (Simple coupler), as its name indicates, is simple. The source code of Scup consists of about only 8000 lines, which is about 1/10 or 1/20 compared to OASIS3, OASIS4 and ESMF. This simplicity makes Scup easy to be developed, maintained and used.

(7) Scup is portable. All the source code of Scup is written in Fortran 95 without any other external libraries (except MPI library) so that it can be easily compiled, which provides good portability of Scup.

(8) An object-oriented approach is adopted in Scup using Fortran 95 modules, derived types and pointers to easily handle data. A derived type object of Fortran 95 is considered as a object which has data and methods (i.e. subroutines to operate the data). Detailed description is given in APPENDIX.

3. Communication among component models

To efficiently exchange a large amount of data among component models, they communicate with each other directly using the Scup library. In Scup,
Message Passing Interface (MPI) Library\(^1\) is used to exchange data among component models. The component models are usually executed concurrently on different MPI processes. Each component model may be parallelized into several MPI processes. The MPI processes of the component models directly communicate with each other by calling subroutines of the Scup library as shown in Fig.1 (a). In this method (method (a)), Scup does not have its own exclusive MPI processes, but works on the processes of the component models.

There is also another method as shown in Fig.1 (b) (method (b)), where the MPI processes of the component models communicate indirectly via a particular coupler process. In this method, since the data are collected into the coupler process during data exchange, concentration of communication processing on the coupler process is possibly a bottleneck. Especially, when a large number of MPI processes are used for the component models, communication processing of a large amount of data is concentrated on the coupler process at the same timings. In the method (a), on the other hand, there is no such problem because the communication processing is distributed over many processes. Moreover, in the method (b), since data is transferred via the coupler process, the amount of communication itself is twice as much as in the method (a). Therefore, the method (a) is more efficient than the method (b), and we have adopted the method (a) in Scup.

\(^1\) We use only MPI-1 subroutines, not additional MPI-2 subroutines so far.
In the method (a), each process receives only the necessary data for the grid area of its own process. Therefore, each process receives data only from the processes that have the necessary data for its own process (e.g. solid lines in Fig. 1 (a)), not from the processes that do not have the necessary data (e.g. dotted lines in Fig. 1 (a)).

4. Scup interfaces

We designed interfaces of Scup library so that component models would only require minimum modification for coupling them, which makes it easy to introduce Scup into the component models. When designing the interfaces, we referred to OASIS3 and OASIS4 user guides (Valcke et. al. 2004b, 2004c) about the concept of coupler interfaces.

Figure 2 illustrates a sample flowchart of coupling procedures in each component model with typical Scup library subroutine calls. The subroutine calls should be inserted to the each component model in an appropriate order.

In each component model, initial settings of Scup are performed in the order of steps (a)~(g) as in Fig.2. In step (a), Scup is initialized and the model name of each component model is informed to Scup. In step (c), “Scup grid” objects (e.g. obj_grid_atmos_2d in Fig.2; See APPENDIX about the object using Fortran 95) are initialized with grid information given by arguments. The grid information consists of the array size assigned on its own MPI
process and a mapping table indicating the correspondence between local array indices in its own MPI process and global grid addresses. Inside Scup, the mapping table is used in grid transformation as described in Section 7. In step (d), “Scup put-variable” objects (e.g. obj_varp_sens in Fig.2) and “Scup get-variable” objects (e.g. obj_varg_tsea in Fig.2) are initialized, each of which belongs to one of the “Scup grid” objects. These Scup variables are used later in steps (i) and (j). In step (f), inside Scup, the communication configuration file “Scup namelist” (See Section 5) is read and information for coupling is exchanged among the processes of the component models for preparing data transfer and grid transformation among them (See Section 7).

Either step (e) or (g) is called in each model to share start and end date (time) information. In step (e), the date information must be provided to Scup by at least one component model. If contradictory date information is provided by more than one component model in step (e), execution of the component models stops after output of error messages. In step (g), shared date information is obtained from Scup by the other component models.

Next, subroutines of Scup library in steps (h)~(k) in Fig.2 are called every time step in the each component model for exchanging data among the models. The current time and the timestep size are informed to Scup in step (h). In step (i), data of “Scup get-variable” is obtained from Scup. In step (j), “Scup put-variable” data are put into Scup.
Inside Scup, data is sent to other processes of the component models in a non-blocking manner (with the MPI subroutine MPI_ISEND) in an appropriate step: either step (h) or step (k) (or optionally step (j)) depending on which "time lag" (See Section 6) is used and whether communication is between component models or within each component model. The data is received from other processes (with MPI_IRECV and MPI_WAITALL) in an appropriate step: either step (h) or step (i). The data sending is waited to finish (with MPI_WAITALL) in step (h), if necessary.

5. Communication configuration file “Scup namelist”

Model coupling is flexibly controlled by the communication configuration text file, “Scup namelist”, which is commonly read by all processes of the component models in step (f). In the sequence of the steps (h)~(k), data is transferred according to “Scup namelist”.

In “Scup namelist”, settings on the data transfer among the component models (or within each component model) are described in the order as follows.

(A) Model name and grid name for each sending side and receiving side, and a filename for horizontal grid transformation.

(B) Variable name for each sending side (put-variable) and receiving side (get-variable), and its attributes on the communication: time interval, time lag (See Section 6) and time flag for selecting either average or
snapshot.

There can be more than one entry for (B) after one entry for (A). By repeating entries (A) and (B), we can describe all the data transfer configurations.

Figure 3 shows an example of “Scup namelist”. In this example, the model ‘ATMOS’ holds the grid ‘ATMOS_2D’ that contains put-variables ‘SENS’ and ‘EVAP’. The put-variables’ data are transferred into the get-variables of the grid ‘OCEAN_2D’ of the model ‘OCEAN’, where the put-variable ‘SENS’ (‘EVAP’) in ‘ATMOS_2D’ corresponds to the get-variable ‘SENS’ (‘EVAP’) in ‘OCEAN_2D’. ‘Fl_remap’ indicates the file name of the horizontal grid transformation table (See Section 7 for the detail), and is set as fl_remap=’ ’ when grid transformation is not necessary. The parameter ‘intvl’ specifies the communication time interval, for instance:

- intvl=1 and intvl=3 indicate 1-hour interval and 3-hour interval respectively;
- intvl=–1 indicates every time step;
- intvl=–2 and intvl=–3 indicate half-hour interval and one-third hour interval respectively.

In the current version of Scup, the communication time interval needs to be a multiple of a time step of each component model. The time flag ‘flag’ designates Scup whether data is averaged over the communication time interval or not: flag=’AVR’ indicates that the data is averaged during the time interval and flag=’SNP’ indicates that the latest snapshot value is
transferred. The parameter ‘lag’ specifies "time lag" of the coupling, which is explained in Section 6. The parameters ‘fl_rs_scup_in’ and ‘fl_rs_scup_out’ at &nam_scup_restart specify names of restart files for input and output (See Section 6) respectively for each component model.

By modifying the description of “Scup namelist”, it is possible to flexibly change the coupling behavior or the configuration of component models without modifying source code of the models and re-compiling them. For example, not only transferred variables can be selected and coupling timing can be changed, but also a component model to be coupled can be replaced with another one.

6. "Time lag" of model coupling

How to connect component models in time is controlled by the "time lag", which is specified by the parameter 'lag' in “Scup Namelist” shown in Section 5. Figure 4 illustrates patterns of model coupling using three types of "time lag" in coupling two component models. The three types are:

(A) lag=-1: the receiver component model receives the past values of one timestep before from the sender component model;
(B) lag=1: the receiver component model receives the future values of (coupling interval – one timestep) after from the sender;
(C) lag=0: the receiver component model receives the values of the same time level (zero time lag) from the sender.
Combination of these "time lag" types enables flexible model coupling.

The type (A) (lag=-1) is the most basic one. This type is mainly used in concurrent coupling between the models, where the two component models are executed concurrently, exchanging data at given time intervals. Figure 4(a) illustrates an example of concurrent coupling, where an atmosphere model and a chemical transport model are coupled at 1-hour time intervals. The component models exchange the data with each other after 1-hour concurrent execution. An ozone field obtained from the chemical transport model after the time integration from t=0 to t=1 is used in the atmosphere model as values for t=1 through t=2. In this type, when restarting from a previously executed run, values of data at the final state of the previous run should be received at t=0 (i.e. at the start of the execution). For this purpose, Scup supports restart files for input and output: the values at the final state of the previous run is read from the restart file for input at t=0 in step (f) of Fig.2 and the values at the final state of the present run is written to the restart file for output at the end of the execution in step (k) of Fig.2 as shown in Fig.4 (a). Names of the restart files for input and output for each component model are specified at &nam_scup_restart in “Scup namelist” as shown in Fig.3.

The type (B) (lag=1) is used in sequential coupling, where the two component models are executed sequentially and alternately at given time intervals. Figure 4(b) illustrates an example of sequential coupling using the
type (B) (lag=1) and the type (A) (lag=-1), where an atmosphere model and an ocean model are sequentially coupled at 3-hour intervals. Flux fields (sensible heat, latent heat) for $t=0$ through $t=3$ in the atmosphere model are used in the ocean model as the value for $t=0$ through $t=3$ (lag=1). Sea surface temperature for $t=0$ through $t=3$ in the ocean model is used in the atmosphere model as the value for $t=3$ through $t=6$ (lag=-1).

The type (C) (lag=0) can be used, for example, when boundary data is transferred once at the start of the execution, or when an atmosphere model and a physics scheme in the model, such as land scheme, are coupled every time step as shown in Fig.4(c).

In the types (A) and (B), either time average or snapshot can be transferred. On the other hand, in the type (C), only snapshot can be transferred.

### 7. Grid transformation

Each component model is allowed to use independent coordinates and grids from others. When data is transferred between component models with different coordinates or grids, 2-D (horizontal) or 3-D (horizontal and vertical) grid transformation is performed in Scup. In 2-D grid transformation, a scheme with accurate local and global conservation is supported, which is necessary in order not to produce false sources or sinks of substances, for example, false sea level rise or fall when exchanging water
with water flux (i.e. precipitation and evaporation) between an atmosphere model and an ocean model. In 3-D grid transformation, a conservative interpolation method is used in vertical grid transformation in order to improve conservation, which is also important, for example, when exchanging substances such as CO$_2$ or water vapor between an atmosphere model and a chemical transport model.

7.1 Horizontal grid transformation

In horizontal grid transformation, remapping tables are used, which are read from a file (See Section 5 about the file name) in the beginning of the model execution (in step (f) of Fig.2). The horizontal remapping tables are created by offline calculation using the Spherical Coordinate Remapping and Interpolation Package (SCRIP; Jones 1998, 1999). The offline calculation of the remapping tables contributes to keeping Scup simple and saving execution time. SCRIP can be used for grid transformation between any coordinates or grids. SCRIP supports interpolation schemes for grid transformation, for example, bilinear, bicubic and conservative schemes. We have introduced a ‘filter grid transformation’ scheme to SCRIP in order to raise accuracy of the conservative scheme when boundary lines between grid areas are not along great circles (e.g. latitude lines). In this scheme, grid areas are divided into small ‘filter’ elements during calculations of overlapped areas to create the remapping tables.

The remapping tables consist of
• src_address(1:num_links) : global grid address of a source side;
• dst_address(1:num_links) : global grid address of a destination side;
• remap_matrix(1:num_links) : remapping matrix,

where num_links is the array size of the tables that is the number of links for overlapping grids in the remapping. Grids are transformed by the following Fortran program (Jones 1998):

```
do n=1,num_links
   dst_array(dst_address(n)) = dst_array(dst_address(n)) +
   remap_matrix(n)*src_array(src_address(n))
end do ,
```

where src_array(:) and dst_array(:) are data arrays of the source side and the destination side of the grid transformation respectively. In this program, the values of dst_array(:) are calculated from src_array(:) with the remapping tables. In the first-order conservative remapping, remap_matrix(:) corresponds to area weight. This program is a particular example, in which the global grid addresses of the source and destination sides agree with the data array indices of the source and destination sides respectively. Since how to set the global grid addresses and the data array indices is arbitrary, these do not generally agree, especially in case that the model’s grid is divided by MPI parallelization. To deal with this disagreement, a mapping table between the local array indices in each MPI process and the global grid addresses is given to every Scup grid by the subroutine call (c) in Fig.2. From
this original mapping table and the original remapping tables, a mapping table for each sender process and remapping tables for each receiver process are created as shown below.

The component models are usually executed in parallel on MPI processes, and the data exchange is done by multi-lateral direct communication among the models’ processes as shown in Fig.1(a). A sender process sends the requested data to the multiple receiver processes and a receiver process receives the necessary data from the multiple sender processes. A sender process should hold a mapping table relating the array of its own process to the send-buffer array. Data of the send-buffer array is distributed to multiple receiver processes. A receiver process should hold remapping tables, with which the data array of its own process is calculated from the receive-buffer array. Data of the receive-buffer array is collected from multiple sender processes (only from the sender processes that have the necessary data for remapping of its own process; See solid lines and dotted lines in Fig. 1 (a)). The mapping table of the sender process and remapping tables of the receiver process are created in step (f) in Fig.2, based on the original remapping table read from the file and the mapping tables given by the step (c) in Fig.2.

7.2 Vertical grid transformation

In 3-D grid transformation, a vertical grid transformation is performed after the horizontal 2-D transformation. In the current version of Scup, only
eta-coordinate, which is a hybrid vertical coordinate of sigma (terrain following coordinate) and pressure, is supported in the vertical grid transformation. Piecewise Rational Method (PRM; Xiao and Peng 2004) is used for the vertical grid transformation, which guarantees a monotonic and conservative interpolation.

Figure 5 illustrates conservative vertical grid transformation. In Fig.5, $q_k$ is a variable at k-th vertical layer called “full level” and $p_{k-1/2}$ is a pressure at the boundary level called “half level” between the (k-1)-th full level and the (k)-th full level. $q_k$ and $p_{k-1/2}$ are the values before transformation and $\tilde{q}_k$ and $\tilde{p}_{k-1/2}$ are those after transformation. When $p_{k-1/2}$ and $\tilde{p}_{k-1/2}$ are different, grid transformation is necessary to calculate the value of $\tilde{q}_k$ from $q_k$. First, an interpolation function $q = q(p)$ is calculated from $q_k$ with PRM, where the mean value of $q(p)$ in each full level before the transformation agrees with the value of $q_k$. Next, $\tilde{q}_k$ are calculated from the mean value of $q(p)$ in each full level after the transformation. See Xiao and Peng (2004) and Colella and Woodward (1984) for detail about the interpolation function.

The surface pressure represents the total mass of the air column. Therefore, to make this vertical grid transformation with PRM conservative and, more generally, to make 3D data transfer between component models to be conservative, it is necessary that unapproximated prognostic surface pressures should be used and the surface pressures between component models should be the same when calculating the 3D data transfer and the
vertical grid transformation. Generally, the above necessary condition is not easy to realize. For example, when an atmosphere model and a chemical model are coupled with the pattern of Fig. 4(a), the prognostic surface pressures of both the models are independently integrated in time. Hence, even if both the surface pressures are the same at $t=0$, both are different at $t=1$. Another example is that, when horizontal grids used in component models are different and hence surface heights between component models are different, surface pressures between them are also different.

In the current version of Scup, surface pressures approximately calculated from only surface heights are used for easy calculation in the vertical grid transformation with PRM. Nevertheless, this method using PRM is expected to be superior in conservation and precision to simpler linear interpolations.

8. Summary and future plan

We have developed a simple and flexible coupler Scup for earth system modeling. Scup has sufficient functions necessary for flexible coupling between the component models of the MRI earth system model. As described in Section 2, Scup has good features as follows.

(1) Scup is efficient by direct communication among the processes of the component models.

(2) Interfaces of the Scup library are designed so that only minimum modification of the component models would be required.
(3) By modifying “Scup namelist”, configuration of coupling, such as variables to be exchanged, coupling timing and also models to be coupled, can be flexibly changed.

(4) Combination of three types of the "time lag" enables flexible model coupling in time such as concurrent coupling, sequential coupling and zero time lag coupling.

(5) Scup supports 3-D grid transformation and conservative 2-D grid transformation between component models with different grids.

(6) Scup is simple and therefore easy to developed, maintained and used.

(7) Scup is written in Fortran 95, which contributes to good portability.

(8) An object-oriented approach is adopted in Scup using Fortran 95 modules, derived types and pointers to easily handle data.

Although Scup has been developed for the MRI earth system model, its fundamental design is also applicable for other earth system models. We expect that Scup can be used for other earth system models and other kinds of model coupling such as coupling between global and regional models by some modifications or some additions of functions.

There is a plan in which we will collaborate with other institutes in developing an open-source general-purpose coupler in the future, which will be developed on the basis of the fruits of Scup and other couplers. We hope that the coupler will be superior and easy to use while keeping simplicity, and will be widely used by many component models in Japan and in the
world.

Acknowledgments

We are grateful to members of the MRI global model development team, especially M. Deushi, T. Tanaka and M. Hosaka, for useful comments on the development of Scup.

APPENDIX

Object-oriented approach using Fortran 95

We have adopted an object-oriented approach using Fortran 95 modules, derived types and pointers to easily handle data in Scup, where a derived type object of Fortran 95 is considered as an object which has data and methods (i.e. subroutines to operate the data). This object-oriented approach is usable, for example, for a database used in a coupler or a model monitor.

We use a module that contains a derived type definition and subroutines whose first argument is its derived type object. This module can be considered as a “class” in an object-oriented language such as Java. Components of the derived type object can be considered as “object data” and the subroutines can be considered as “object method”s to handle the object data. The accessibility of the object data is made private so that only the object methods can handle the object data. This hiding of the object data avoids unexpected modification of data from outside the module and
enhances safety of data. From a “class” module, an arbitrary number of objects of the same kind can be created by declaring derived type objects of the “class” module and calling a method for initializing data of the objects.

In Scup, since the objects such as “Scup grid”, “Scup put-variable” and “Scup get-variable” shown in Section 4 needs to be referred from various places inside and outside the coupler, the derived type objects should be declared as pointers. For example, since “Scup put-variable” objects are referred not only from a component model at the subroutine call of Fig. 2 (j), but also from inside Scup in the subroutines of Fig. 2 (f), (h) and (k), pointers to “Scup put-variable” objects have to be registered inside Scup. But in Fortran there is a problem that an array whose elements are pointers cannot be declared. To be able to declare the array of the objects with pointer characteristic, we find a way that the derived type object has a pointer to a derived type object in its component and the pointer to a derived type object has data of the object in its component. By this idea, the object has the same feature as a pointer without the pointer attribute and can also be an element of an array at the same time. Here we call the object of this type “pointer-like object”. The assignment (=) of the pointer-like object has the same meaning as the pointer assignment (=>).

Figure 6 shows an example of the “class” module and the “pointer-like object” described above, which is simplified source code of the module for “Scup grid”. “Scup grid” objects are initialized by calling the initializing
method “scup_grid__new”. “Scup put-variable” objects are initialized and registered with “Scup grid” objects in the subroutine “scup_grid__create_varp”. The derived type “type_scup_grid” has a pointer to a derived type object “this”, which makes the object “Scup grid” a pointer-like object. This pointer-like object can be an element of an array, for example:

```fortran
    type(type_scup_grid) :: obj_grid(20).
```

The object “Scup put-variable” is also a pointer-like object. “Scup grid” has an array of “Scup put-variable”, `obj_varp(100)`, which is also an example of an array whose elements are pointer-like objects. As shown in these examples, the pointer-like object works well.

References


Craig, A., R. Jacob, B. Kauffman, T. Bettge, J. Larson, E. Ong, C. Ding and Y.


和文概要

地球システムモデルのためのシンプルなカップラー「Scup」の開発

大気・海洋大循環モデルや化学輸送モデルのような要素モデルを柔軟に結合し地球システムモデルを作成する目的で使用可能な、シンプルな汎用カップラー「Scup」(Simple coupler)の開発を行った。

Scupは気象研究所の地球システムモデルを効率的に開発するために必要十分な機能を備えており、その基本設計は他の地球システムモデルにも適用可能である。Scupライブラリを呼ぶことにより要素モデルの各々のプロセスはお互いに直接通信を行っており、このことは要素モデルのプロセス間で大容量のグリッドデータを効率的に交換するために必要である。Scupは局所的かつ全球的に正確に保存する2次元データのグリッド変換だけでなく、3次元データのグリッド変換もサポートしており、このことは異なる座標やグリッドを持つ要素モデルを結合するために必要である。

単純さと使用しやすさがScupの長所である。ScupのすべてのソースコードはFortranで記述しており、Fortran95コンパイラとMPIライブラリを有する多様なプラットフォームでコンパイル可能である。このことはScupの可搬性の良さに寄与している。
Figure Captions

Figure 1  Schematic diagram of methods (a) and (b) for communication among three component models (ATMOS, OCEAN and CHEMI).
(a) Direct communication among MPI processes of the component models. This method is used in Scup. Each process receives data only from the processes that have the necessary data for its own process (solid lines), not from the processes that do not have the necessary data (dotted lines).
(b) Indirect communication via a particular coupler process.

Figure 2  Flowchart of procedures in each component model for coupling with Scup library.

Figure 3  Example of a "Scup Namelist" file, where configuration of coupling, such as variables to be transferred among component models and coupling timing, are described.

Figure 4  Schematic diagrams for patterns of model coupling:
(a) Concurrent coupling using lag=-1, where two component models are executed concurrently;
(b) Sequential coupling using lag=1 and lag=-1, where two component models are executed sequentially and alternatively at given time intervals;

(c) No time lag coupling using lag=0, which is used, for example, for boundary data exchange at the start of the execution or data exchange at every time step,

where, three types of “time lag” are used:

(A) lag = –1 : The receiver receives the past values of one timestep before from the sender;

(B) lag = 1 : The receiver receives the future values of (coupling interval – one timestep) after from the sender;

(C) lag = 0 : The receiver receives the values of the same time level (zero time lag) from the sender.

Figure 5  Illustration of conservative vertical grid transformation. $q_k$ is a variable at the $k$-th vertical layer called “full level” and $p_{k-1/2}$ is a pressure at the boundary level called “half level” between the $(k-1)$-th full level and the $(k)$-th full level. $p_s$ is a surface pressure. $q_k$ and $p_{k-1/2}$ are the values before transformation and $\tilde{q}_k$ and $\tilde{p}_{k-1/2}$ are those after transformation. $q = q(p)$ is an interpolation function.
Figure 6  Simplified source code of the module for “Scup grid”, which is an example of object-oriented approach using Fortran 95 modules derived types and pointers.
Figure 1
character(len=16)::name_model='ATMOS' !! Model name
integer :: mpi_comm_atmos !! For intra-model comm.
integer :: imax, jmax !! Array size of data
integer:: map_12g(imax, jmax) !! Mapping table(local to global)
type(type_scup_grid):: obj_grid_atmos_2d !"Scup grid" object
type(type_scup_varp):: obj_varp_sens !"Scup put-variable" object
type(type_scup_varp):: obj_varp_tsea !"Scup get-variable" object
integer:: idstart(4), idend(4) !! Year, month, day, hour
logical:: leapyear !! Leapyear exist or not
real(8):: time, delt !! Current time, timestep
real(8):: tsea(imax, jmax), sens(imax, jmax) !! Tsea data, Sens data

(a) call scup_init( name_model ) !IN
(b) call scup_get_comm_local & & (mpi_comm_atmos ) !OUT
(c) call scup_create_grid & & ('ATMOS_2D', imax, jmax, map_12g, & !IN & obj_grid_atmos_2d ) !OUT
(d) call scup_grid_create_varp & & (obj_grid_atmos_2d, 'SENS', & !IN & obj_varp_sens ) !OUT
   call scup_grid_create_varp & & (obj_grid_atmos_2d, 'TSEA', & !IN & obj_varp_tsea ) !OUT
(e) call scup_set_date & & (idstart, idend, leapyear) !IN
(f) call scup_endini
(g) call scup_get_idstart( idstart ) !OUT
call scup_get_idend( idend ) !OUT
call scup_get_leapyear( leapyear ) !OUT

---

Time step loop

(h) call scup_set_time( time, delt ) !IN
(i) call scup_varp_get_value & & (obj_varp_tsea, tsea ) !INOUT, OUT
(j) call scup_varp_put_value & & (obj_varp_sens, sens, 1.0d0) !INOUT, IN
(k) call scup_send
(l) call scup_finalize

---

Figure 2
&nam_scup model_put='ATMOS', model_get='OCEAN',
    grid_put='ATMOS_2D', grid_get='OCEAN_2D',
    fl_remap='.inputs/rmp_8_6_toy.d'
&nam_scup var_put='SENS', var_get='SENS', intvl=3, lag=1, flag='AVR'
&nam_scup var_put='EVAP', var_get='EVAP', intvl=3, lag=1, flag='AVR'
&nam_scup model_put='OCEAN', model_get='ATMOS',
    grid_put='OCEAN_2D', grid_get='ATMOS_2D',
    fl_remap='.inputs/rmp_6_8_toy.d'
&nam_scup var_put='SST', var_get='TSEA', intvl=3, lag=-1, flag='SNP'
&nam_scup model_put='CHEMI', model_get='ATMOS',
    grid_put='CHEMI_3D', grid_get='ATMOS_3D',
    fl_remap=''
&nam_scup var_put='OZON', var_get='OZON', intvl=1, lag=-1, flag='AVR'
&nam_scup model_put='ATMOS', model_get='CHEMI',
    grid_put='ATMOS_3D', grid_get='CHEMI_3D',
    fl_remap=''
&nam_scup var_put='U', var_get='U', intvl=1, lag=-1, flag='SNP'
&nam_scup var_put='V', var_get='V', intvl=1, lag=-1, flag='SNP'
&nam_scup model_put='ATMOS', model_get='ATMOS',
    grid_put='ATMOS_2D', grid_get='LAND',
    fl_remap=''
&nam_scup var_put='EVAP', var_get='EVAP', intvl=-1, lag=0, flag='SNP'

&nam_scup restart model = 'ATMOS',
    fl_rs_scup_in = '.inputs/rs_in/RS_SCUP_ATMOS",
    fl_rs_scup_out = "rs_out/RS_SCUP_ATMOS"
&nam_scup restart model = 'OCEAN',
    fl_rs_scup_in = '.inputs/rs_in/RS_SCUP_OCEAN",
    fl_rs_scup_out = "rs_out/RS_SCUP_OCEAN"
&nam_scup restart model = 'CHEMI',
    fl_rs_scup_in = '.inputs/rs_in/RS_SCUP_CHEMI",
    fl_rs_scup_out = "rs_out/RS_SCUP_CHEMI"

Figure 3
Figure 4
Figure 5
module scup_grid
  use scup_varp, only : type_scup_varp, scup_varp__new, ...
  implicit none
private
public :: type_scup_grid, scup_grid__new, scup_grid__create_varp, ...
type :: type_scup_grid
private
  type(type_this),pointer :: this => null()
end type

type :: type_this
private
  character(len=16) :: name           !! Name of grid
  integer :: imax             !! Imax*jmax is number of horizontal grid
  integer :: kmax             !! Kmax is number of vertical grid
  integer :: jmax
  integer,pointer :: map_l2g(:)   !! Map ( Local array indices
                      !! to Global grid addresses )
  integer :: num_varp = 0        !! Number of used elements of obj_varp(:)
type(type_scup_varp) :: obj_varp(100) !! Varp (“Scup put-variable” object)
end type
contains
subroutine scup_grid__new(obj, name_grid, imax, kmax, jmax, map_l2g)
  !! --- Create and initialize new grid ---
type(type_scup_grid),intent(inout) :: obj !! Grid (“Scup grid” object)
character(len=*) , intent(in) :: name_grid       !! Name of this grid
integer,intent(in) :: imax             !! Imax*jmax is number of horizontal grid
integer,intent(in) :: kmax             !! Kmax is number of vertical grid
integer,intent(in) :: jmax
integer,intent(in) :: map_l2g(imax*jmax)   !! Map ( Local to Global )
type(type_this),pointer :: this
allocate(obj%this)          !! Allocate data of this grid
this => obj%this
this%name = name_grid       !! Initialize data of this grid
this%imax = imax
this%kmax = kmax
this%jmax = jmax
allocate( this%map_l2g(imax*jmax) )
this%map_l2g(:) = map_l2g(:)
end subroutine scup_grid__new

subroutine scup_grid__create_varp( obj, name_varp, obj_new_varp )
  !! --- Create and initialize new varp which belong to grid ---
type(type_scup_grid),intent(inout) :: obj     !! Grid (object)
character(len=*) , intent(in) :: name_varp     !! Name of varp
type(type_scup_grid),intent(inout) :: obj_new_varp !! New varp (object)
type(type_this),pointer :: this
this => obj%this
!! -- Create and initialize new varp --
call scup_varp__new( obj_new_varp, name_varp, this%name, &INOuin, &
                   this%imax, this%kmax, this%jmax ) !!IN
!! -- Register obj_new_varp with this grid --
this%num_varp = this%num_varp+1
this%obj_varp(this%num_varp) = obj_new_varp     !! Same meaning as
end subroutine scup_grid__create_varp
!! pointer assignment (=>)
subroutine scup_grid__set_time_recv( obj, time, delt )
end subroutine scup_grid__set_time_recv
end module scup_grid

Figure 6