Arakawa *et al. Progress in Earth and Planetary Science* (2020) 7:6
https://doi.org/10.1186/s40645-019-0320-z

Progress in Earth and Planetary Science

REVIEW Open Access

Coupling library Jcup3: its philosophy and application

Takashi Arakawa[1*], Takahiro Inoue[2], Hisashi Yashiro[3] and Masaki Satoh[4]

Abstract

In this paper, we describe the design of the coupling library, Jcup, and report its various applications, including the coupling between global atmospheric and oceanic models of different grid systems. Jcup is a software library mainly focused on weather/climate models and was developed for coupling the components of various models. Jcup has the flexibility to be applied to an unspecified number of components of earth system models. To achieve a high level of safety and versatility, we classified the processes of the general coupling software into processes that change the value of the data and those that do not and placed the former outside of the program and under the control of the user. Consequently, Jcup exhibits two features: (1) the correspondence relationship between grid indexes is used as input information, and (2) the user can implement an arbitrary interpolation code. Jcup was applied to atmosphere-ocean coupling, IO component coupling, and the coupling between the seismic model and structure model, and the validity and usefulness of the design were demonstrated.

Keywords: Earth system modeling, Coupler, Coupled simulation

Introduction

A typical weather/climate model is generally composed of several submodels, such as an atmospheric model, ocean model, and land surface model. However, as the required accuracy of reproduction increases, the number of submodels also increases. For example, according to the transition of the components of the NCAR CCSM (2009), at the beginning of its development, the model comprised two components, namely, the atmosphere (land surface) and the ocean. In the 1990s, the sea ice and aerosol components were added, and the vegetation and carbon cycle modules are currently being incorporated.

In the structure of such a model, it is necessary to exchange relevant information on the appropriate spatiotemporal scale, corresponding to each component, when executing the model. In this situation, a grid remapping is required according to the spatial scale of each component; however, it is not preferable from the viewpoint of development efficiency and maintainability to separately develop and implement such a remapping program for each component. For this reason, dedicated software responsible for the coupling between components has been developed and used.

In this paper, software that executes such tasks is called coupling software or couplers. Generally, there are two types of coupling software. One is a program that targets a specific model.

In this case, because target components are predetermined, coupling software specialized for a specific grid system, time scale, or coupling pattern may be sufficient. An example of this type of coupling software is the coupler used in NCAR CESM. The other type is the coupling software developed for general use. As the specific target is not assumed, the structures of both the interface and program are determined depending on the extent of support on the grid system, the coupling pattern, and the interpolation method. As a representative of this type of coupler, the OASIS coupler was developed mainly by CERFACS and is widely used among European meteorological research groups (Valcke et al. 2006; Valcke 2013).

YAC is also a coupling software developed by European research institutions (Hanke et al. 2016). YAC is developed as a lightweight library that supports various grid systems. YAC 1.2.0 is utilized in the coupling of the Icosahedral Nonhydrostatic general circulation model. As its name suggests, ESMF is a software framework for earth

*Correspondence: arakawa@rist.jp
[1]Research Organization for Information Science and Technology, 1-18-16 Hamamatsu-cho, Minato-ku, Tokyo 105-0013, Japan
Full list of author information is available at the end of the article

system modeling (Hill et al. 2004). ESMF provides several functions for various component models that make up the earth system model, and it can flexibly couple component models that have various grid systems.

C-Coupler2 has state-of-the-art functions supporting variable vertical coordinates in time integration (this function is also supported by ESMF) and a user-friendly interface using XML (XML is also supported by YAC) to deal with complex couplings (Liu et al. 2018).

Scup was developed to target models of the Japanese Meteorological Agency and Meteorological Research Institute but has a general-purpose interface that can be used for other models (Yoshimura and Yukimoto 2008). Furthermore, MCT was developed as a basic software library for constructing coupling software and cannot perform coupling alone. However, it can be thought of as general-purpose coupling software (Larson et al. 2005; Jacob et al. 2005). In addition to being used for CESM couplers, MCT is also utilized as a basic library for constructing the latest version of the OASIS coupler (OASIS3-MCT, Craig et al. 2017).

These couplers have developed various functions in response to the demands of component models that progress in larger and more complex directions. For example, OASIS3-MCT, ESMF, and YAC support concurrent and sequential coupling. OASIS3 and ESMF can generate interpolation weights offline. For many of these couplers, the diversity of grid systems that can be adapted has been discussed, and that diversity is particularly noticeable in YAC, ESMF, and OASIS3-MCT.

Jcup has also been developed for application in a variety of models, but its approach is different from the existing couplers.

These couplers support existing grid systems and interpolation methods and provide coupled computing environments; however, to deal with future grid systems and interpolation calculations that the software does not currently support, some kind of software modification will be required. Meanwhile, Jcup is a library developed to correspond to various grid systems and interpolation algorithms without the need for future modification of its code and enable coupled calculations of various patterns.

To achieve this goal, first, the general characteristics of weather/climate models need to be considered as a target of a coupled simulation. These may be summarized as follows:

- Each model has a grid structure suitable for the physical state expressed by the model. In addition, the optimum grid structure may change depending on external factors, such as computer architecture. For example, in the conventional global atmospheric model, latitude and longitude grids and spectral methods were used. However, to avoid an increase in the calculation cost of Legendre transformations, models using grid structures that differ from those in conventional models, such as icosahedral grids, Yinyang grids (Baba et al. 2010), and cubic grids (Adcroft et al. 2004), have recently been developed. Regarding the ocean model, grid point concentration in the polar region (Arctic Ocean) is a classical problem, which is typically resolved by the use of stretch and tri-polar grids. Furthermore, river models, for which an irregular grid system is used to express the catchment along the river channel (Yamazaki et al. 2014), might become a coupling target.
- The interpolation method between models varies depending on the physical requirements of each model and cannot be uniquely determined. The cases in which the integration period is relatively short or the system is not physically closed; it is unnecessary to strictly satisfy conservativity, whereby, simple linear interpolation might be sufficient. However, in simulations that require long integration times and have physically closed systems, such as climate simulations, the preservation of physical quantities is crucial, and interpolation algorithms that satisfy conservativity are required. Furthermore, it can be assumed that the physical quantity to be interpolated or the interpolation algorithm itself might change depending on the physical condition in the model, for example, the solar altitude and coverage degree of ice.
- In many cases, coupled simulations entail multiphysics and multiscale nonlinear calculations, making computational bugs extremely difficult to find. In complex-system simulations, results are difficult to predict because of the nonlinearity of the system and are complexly altered by slight changes in the parameters. Therefore, even if there are bugs in the program, their detection is generally difficult, requiring intensive labor, except fatal bugs for which the results would change radically.

To deal with such models, Jcup contains the following two features.

- Correspondence of grid points in an interpolation calculation is used as input information.
- Interpolation calculation code can be freely implemented by the user.

In this paper, the design, implementation, and application examples of Jcup are described. First, the structure and justification for the design of Jcup is described. Next, an implementation that realizes the above two features is described. And finally, we clarify the usefulness of the design adopted by Jcup in three cases: atmosphere-ocean coupling, coupling of IO components, and coupling between the seismic model and structure model.

Review

Overview of Jcup

First, the general features of Jcup are addressed. Like the other couplers mentioned in the introduction (OASIS3-MCT, ESMF, YAC), Jcup also supports concurrent and sequential coupling. Figure 1 shows an example of such a coupling pattern. There are three binaries, namely, A, B, and C, in the figure. Component A is executed in binary A, components B, C, and D are executed in binary B, and component E is executed in binary C. In binary B, component B is executed in all MPI processes. Subsequently, component C is executed in certain processes, while component D is executed in the other process in parallel. The solid line in the figure indicates parallel exchange, and the dotted line indicates sequential exchange. In reality, it seems that there is no case where such complicated execution and data exchange patterns are required, but Jcup is designed to deal with such complicated cases. The time interval for data exchange is constant for each record of data. The interval is set for each record of data through the Jcup API subroutine in the initialization process. The interface related to data exchange is detailed in Arakawa et al. (2011). In this paper, we refer to Jcup as a "coupling library," rather than a "coupler," because Jcup is delivered as subroutine libraries.

Design philosophy

Relationship between research community and development community

As mentioned in the introduction, coupling software is roughly categorized as those targeting specific models and those intended to be used generically without specifying a model. In the former case, the majority of the coupling software development community is close to or included in the model research and development community. The direction of model improvement is well organized, and the development community also works with limited partners. Consequently, the developers of coupling software can respond quickly and appropriately to changes in model components. However, in the case of coupling software developed for general purpose use, the relationship between the development entity and the user community varies. For example, they can belong to the same community, as in the case of Scup, they can develop software independently without belonging to a specific research community, as in the case of MCT, or development can be promoted by maintaining a relatively close relationship with multiple research institutions, as in the case of OASIS. In particular, when the development community does not have a relationship with a specific research community and develops coupling software targeting an unspecified number of models, the direction of model improvement is not determined. In such a case, the relationship between the research community and the development community tends to be distant. From the view of the research community, this means that the required improvement of the coupling software accompanying the improvement of their own model might not be performed promptly or at all. Moreover, for research communities, the coupler code is a black box, which means that it is difficult to detect and trace a non-fatal bug.

Essential functions of coupling software

The essential functions of coupling software in the large-scale parallel computing environment is as follows: (1) coupling between two or more component models, (2)

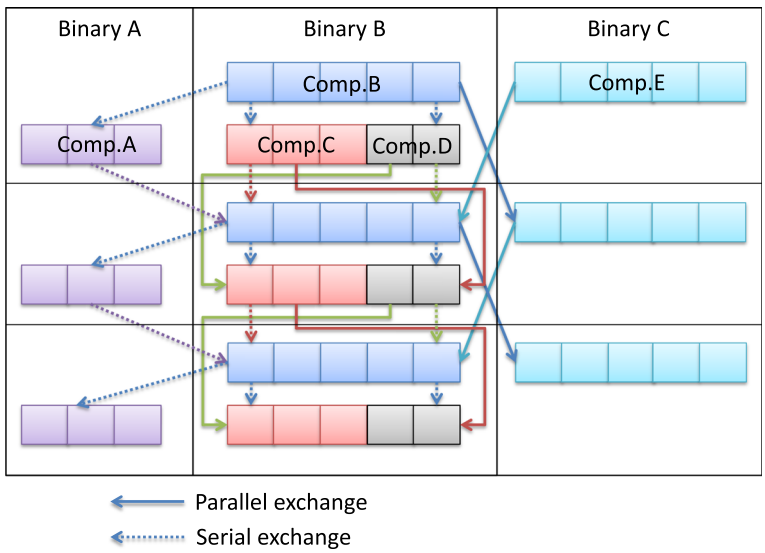

Fig. 1 Example of complex coupling pattern. The solid line indicates parallel exchange, and the dotted line indicates serial exchange

appropriate timing, (3) performing appropriate time interpolation according to the time scale of each model, (4) performing appropriate interpolation calculations according to the grid shape, and (5) exchanging data between appropriate nodes. Here, the condition that the time step of each model is a divisor of the data exchange interval is set. This condition seems to be reasonable in coupling components such as atmospheric and ocean models that involve large-scale phenomena and long- term, constant interaction. In this case, the essential functions of coupling software are to perform interpolation calculation and data exchange at appropriate times. Since the characteristics of Jcup are related to data exchange and interpolation calculation, the following sections focus on these two characteristics in the explanation of the design and implementation of Jcup.

Basic design of Jcup

Jcup is a general purpose coupler that does not cover specific models, and the relationship between the coupler developer and user community is one-to-many and not dense. In this case, the required design concepts can be summarized as follows:

- Since Jcup is a black box for users, it is desirable to minimize the possibility of bugs caused by the coupler.
- Since Jcup is used over a wide range, it must respond flexibly to various grid systems and interpolation methods that exist or will be developed.

The design philosophy of Jcup is to realize the above-mentioned two functions under these requirements.

- Data exchange Based on the implementation of Jcup, we assumed that the spatial interpolation calculation was performed by the receiving-side component. Furthermore, we assumed that each component was parallelized by region. In this case, data exchange means that the value of each grid point of the sending component is transmitted and received by an area that requires its value in the receiving component, that is, it uses the value of that point in the interpolation calculation. Therefore, to perform appropriate data exchange, information on the grid point index of the sending-side component used for calculating each grid point value of the receiving-side component was necessary. The correspondence between the grid point index of the sending-side component and that of the receiving-side component in the interpolation calculation is referred to as a mapping table here. Moreover, since each component is parallelized by region, it suffices that a point index for each grid point is assigned to each region of each component to perform the appropriate data exchange. The problem here is how to obtain the mapping table. In general coupling software, the position of the grid points and the grid shape are given as input information, and mapping tables and interpolation coefficients are calculated in the coupling software. This method, however, has the following problems:

  1. Coupling software can only deal with grid shapes and interpolation algorithms that are already installed, and portability/extensibility is poor.
  2. Depending on the grid shape, it might be difficult to eliminate the possibility of bug contamination, for example, requiring complicated calculations or accurate judgment considering the rounding error.

  This can be a serious disadvantage for general-purpose coupling software, for which there is a distance between the software developer and users and which is recognized as a black box by the user. Therefore, Jcup does not calculate the mapping table internally but uses the mapping table itself as input information. This enables the elimination of problems caused by the mapping table calculation from the coupling software, as described above. The idea of using the mapping table as input information is not unique to Jcup; there are other coupling software that can give a mapping table in addition to grid information, such as OASIS3-MCT or YAC coupler. What is significant about Jcup is not that the mapping table can “also” be used as input information but that input information is limited to the mapping table only.
  As a result, increases in the size and complexity of the code of the coupling software are suppressed, and the future extensibility and security of the software are "clearly" secured. This is considered to be an essential condition for an unspecified number of research communities that cannot necessarily uniquely control the tendency of a coupling software developer.
- Interpolation calculation
  As with data exchange, in interpolation calculation, it is important to ensure future extensibility and portability and to eliminate the possibility of bug contamination. For this purpose, the interpolation calculation program should not be implemented as a black box in Jcup but should be placed in a location such that it can be controlled by the user. The difference between the mapping table and the interpolation calculation is that the former can be calculated with a program independent of the model before model execution, whereas the interpolation calculation must be performed at each data exchange step during model execution. This means that the

calculation performed during model execution as part of the operation of the coupling software must be placed under user control. To realize this in the interpolation calculation, we ensured that the interface subroutine was provided by the coupling-software side, and the concrete interpolation calculation code was implemented in the interface subroutine by the user. The following conditions were considered:

1. Interpolation algorithms used for one set of coupling are not necessarily limited to one type but may vary depending on data or other conditions.
2. Interpolation calculation is executed not only for each type of data; it can also be computed by combining multiple types of data, such as in vector calculation.

Therefore, in the interface subroutine implementing the interpolation calculation, it is indispensable to have functions for identifying individual data points and for simultaneously processing multiple kinds of data. Conversely, if these functions are provided, arbitrary interpolation calculation code can be implemented with a high degree of freedom.

Implementation

As described in the previous section, the unique function of Jcup is that the user can implement the spatial interpolation calculation code. In this section, the data flow and code structure for realizing this is explained.

Data flow

First, the data flow related to data exchange and interpolation calculation is described. Here, in order to facilitate understanding, an explanation is provided based on the example of Fig. 2. In the figure, data are exchanged from component A to component B. Both components have a 6×6 grid, and the positions of the grid points are assumed to be the same. Therefore, the interpolation calculation is equal to a copy of the value for each grid point. The red lines drawn on each grid represent the dividing line of the region. In the figure, component A is divided into six parts of size 2×3, and component B is divided into three parts of size 1×3. The grid points of component B are color-coded. The gray circles represent the masked value. In this case, only the values of the grid points represented by the blue circle are meaningful and are to be sent and received. In the figure, the data exchange and interpolation calculation relating to rank 2 of component B are shown in detail. The grid point indexes, required by rank 2 of component B, are 33 on rank 3 of component A, 28, 33, and 34 on rank 4, and 29, 30, 35, and 36 on rank 5. Therefore, these grid point values are sent from ranks 3, 4, and 5 of component A to rank 2 of component B. These values are received by rank 2 of component B and passed to the interpolation subroutine in the order of receipt. The interpolation calculation is executed inside the interpolation subroutine(interpolated_data) called from Jcup in the receiving-side component. In this example, since the interpolation calculation is a data copy from the grid point to the grid point, indicated by the code below, it is sufficient to provide conversion tables *is* and *ir* of the array indexes from iteration index *i*.

```
do i = 1, N
 R(ir(i)) = S(is(i))
end do
```

Initialization

The information necessary for the data exchange and interpolation calculation is the following:

1. For the sending component, the rank numbers of the receiving component that receives the value of each grid point value held by a sending rank.
2. For the receiving component, the rank numbers of the sending component that holds the value of the sending-side grid point necessary for the interpolation calculation.
3. The conversion tables representing the conversion from the iteration index to the array indexes in the interpolation calculation for each rank.

The first and second items of information can be calculated from the correspondence table showing the relationship between the grid point indexes and the rank number to which they belong (index-rank table), along with the mapping table. The index-rank table can be easily constructed by gathering the grid point indexes of each rank into one rank. Figure 3 shows the example of how to calculate the rank of the target component. The left square indicates the grid point indexes of rank 1 of component A. The center square is a mapping table, and the right square is the index-rank table of target component B. As shown in the figure, the rank number of each grid point of the sending destination/receiving source can be obtained immediately from the mapping table and the index-rank table of the "opponent component."

The third piece of information corresponds to *is* and *ir*, described in the previous section. In Jcup, the interpolation calculation is performed on the receiving component. Thus, these tables can be calculated from the order of the grid point indexes of each receive rank, mapping table, rank number of the sending component, and order of the grid point indexes within the send rank. In addition, the last two of these four conditions to calculate the tables are obtained from the second of the three items of information discussed above.

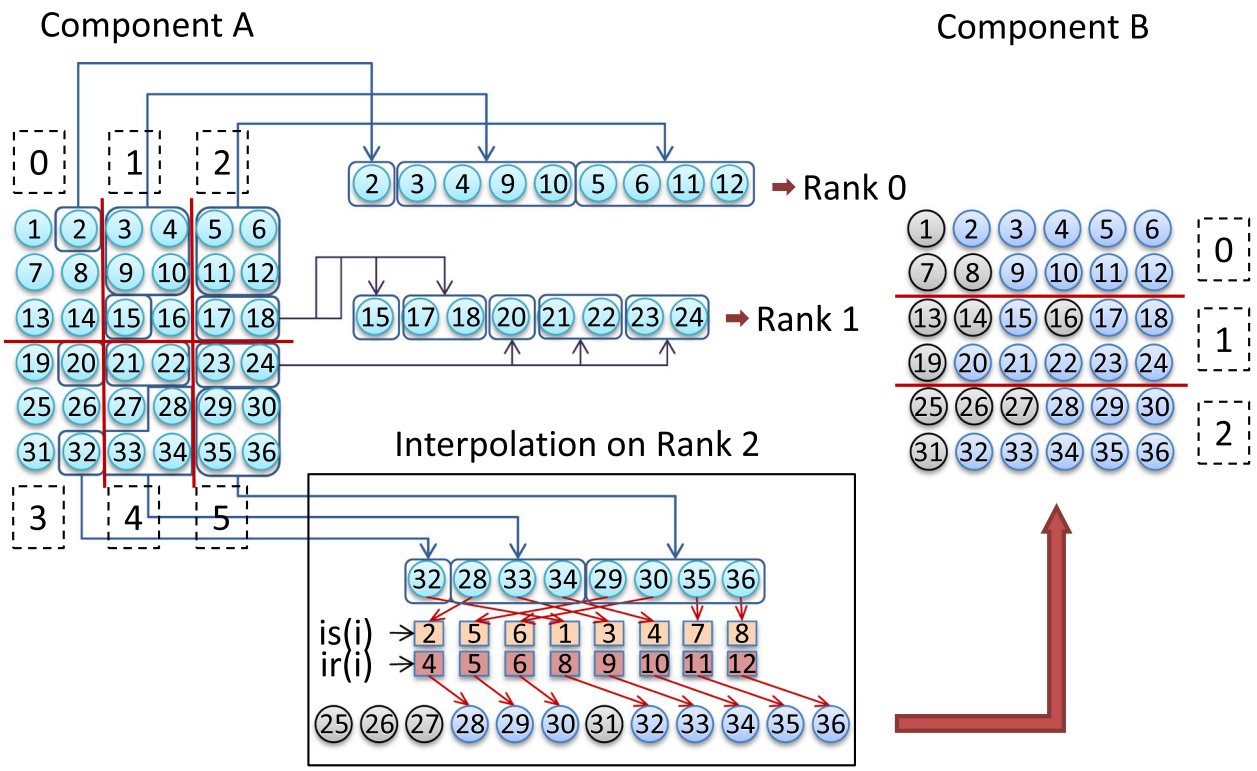

Fig. 2 Data exchange diagram. The gray circles represent masked value. Therefore, only the grid point value indicated by the blue circle is passed from component A to component B

Therefore, the information necessary to perform the data exchange and interpolation calculation can be constructed from the grid point indexes held by each rank and from the mapping table. Jcup provides APIs, jcup_def_grid and jcup_set_mapping_table, to obtain this information (Table 1). Users can provide information to Jcup by calling these subroutines at the appropriate steps in the initialization process.

Time integration

There are only three types of Jcup subroutines that a user must call during the time integration: jcup_set_time, jcup_put_data, and jcup_get_data.

Figure 4 shows an overview of these three routine calls and data processing within Jcup. In the figure, blue squares indicate the API routines. Green squares refer to the data processing in Jcup. The orange square represents the calling of the interpolation subroutine interpolate_data in which the interpolation code must be implemented by users.

Jcup_put_data is a subroutine for applying the data to Jcup. The argument "varp" is an identifier of the data, and a unique value is set in advance by the initialization subroutine jcup_def_varp. Jcup_get_data is a subroutine for obtaining the data from Jcup. Same as jcup_put_data, the argument "varg" is an identifier of the data. A unique value is set in advance by the initialization subroutine jcup_def_varg. The functions of both subroutines comprise only put/get data to/from data buffer in Jcup, and almost all functions for exchanging the data are performed inside the subroutine jcup_set_time.

Subroutine jcup_set_time is expected to be called at the beginning of each time step. The argument "time" refers to the current model time(YYYY/MO/DD/HH/MM/SS) and "deltaT" is the ΔT of the current time step. The left side of the figure shows the data sending process. At the beginning of the subroutine jcup_set_time, a judgment is made for each data record as to whether it is time to send. The data judged to be sent are extracted from the buffer and rearranged based upon the first piece of information listed above. After the rearrangement, the data are sent to the appropriate ranks of the receive-component.

The right side of the figure shows the data-receiving process. Similar to the sending side, at the beginning of the subroutine, a judgment is made for each data record as to whether it is time to receive. The data judged to be received are received based upon the second piece of information from the list and rearranged into one array. After the receive-rearranging loop, the data are passed to the interpolation subroutine interpolate_data, and after the interpolation based on user implementation, the result (iA) is stored in the data buffer.

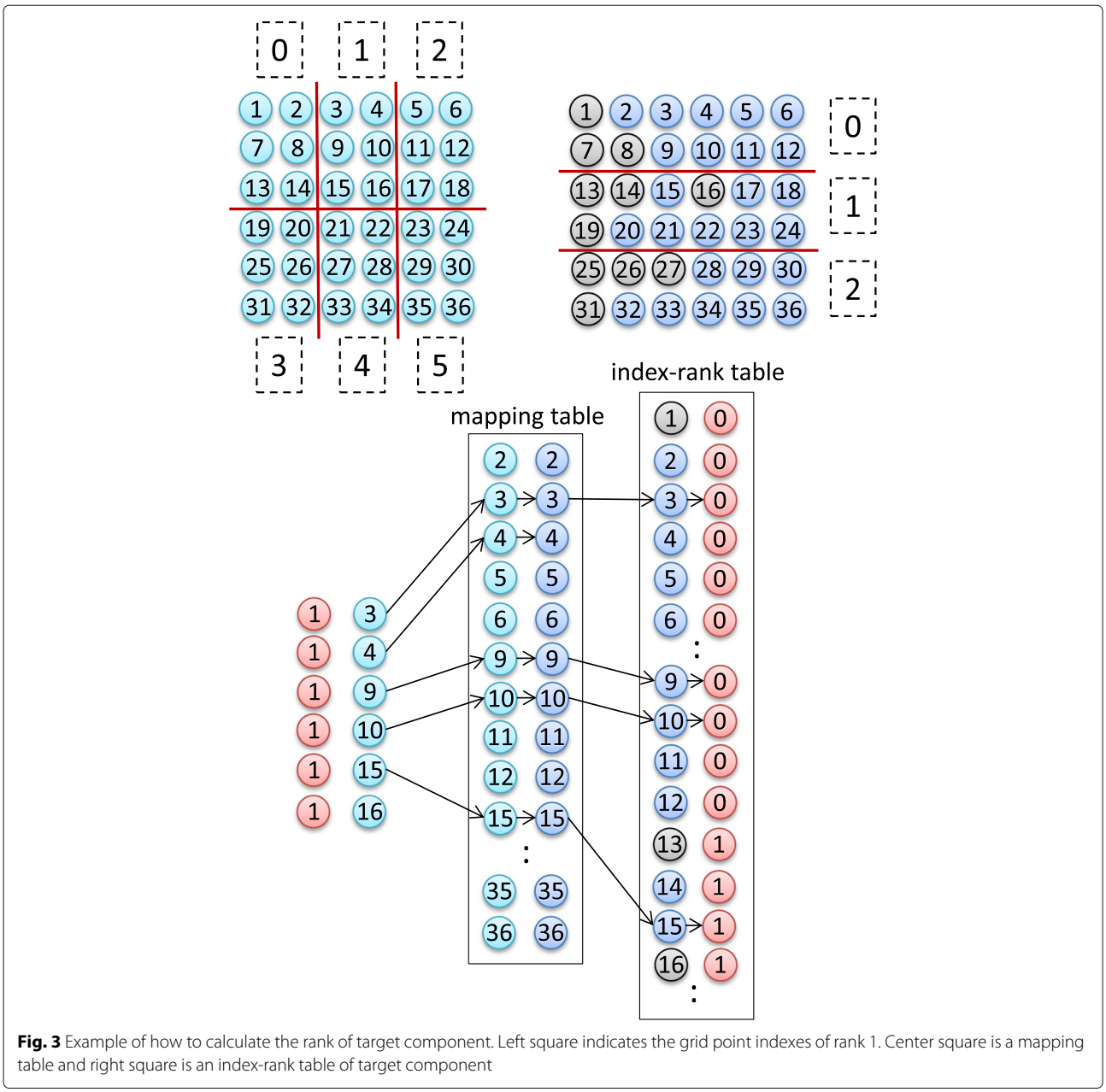

Fig. 3 Example of how to calculate the rank of target component. Left square indicates the grid point indexes of rank 1. Center square is a mapping table and right square is an index-rank table of target component

Interpolation

The subroutine interpolate_data is the realization of a key concept of Jcup; thus, its interface and implementation example are explained in this section. The arguments of this subroutine are listed in Table 2. Send_comp_name and recv_comp_name are the names of send and receive components, respectively. The argument mapping_tag is the identifier of the mapping table set by the initialization subroutine set_mapping_table. The arguments sn1 and sn2 and rn1 and rn2 indicate the size of arrays send_data and recv_data, respectively. Jcup treats the spatial dimension to be interpolated as one dimension. For example, assuming horizontal interpolation, sn1 and rn1 are the number of horizontal grid points of the send- and receive-components, respectively. The arguments, sn2 and rn2, representing the size of the array of the second dimension, ordinally refer to the number of the vertical layer. The last argument, data_tag, is an identifier for identifying the data. The value of this argument is set at the initialization subroutine jcup_def_varg by the user. For future extension, data_tag is to be given as an array, but each value of the array is the same.

Table 1 APIs for grid setting

Subroutine name	Argument type	Argument name	Description
jcup_def_grid	Integer, intent(IN)	grid_index(:)	Array of grid index
	Character(len=*), ,intent(IN)	comp_name	Name of component
	Character(len=*), intent(IN)	grid_name	Name of grid
	Integer, optional, intent(IN)	num_of_vgrid	Number of vertical layers
jcup_set_mapping_table	character(len=*), intent(IN)	my_comp_name	Name of my component
	Character(len=*), intent(IN)	send_comp_name	Name of send component
	Character(len=*), intent(IN)	send_grid_name	Name of send grid
	Character(len=*), intent(IN)	recv_comp_name	Name of receive component
	Character(len=*), intent(IN)	recv_grid_name	Name of receive grid
	Integer, intent(IN)	mapping_tag	Identifier of the mapping table
	Integer, optional, intent(IN)	send_grid_index(:)	Index of send grid
	Integer, optional, intent(IN)	recv_grid_index(:)	Index of receive grid

The interpolation calculation code will vary depending on the model, data, etc., but in a simple and typical case, the value of a grid point of receive-component R is calculated from the values of some grid points of send-component S and interpolation coefficients C, as expressed in the formula below.

$$R = \sum_{i=1}^{n} S_i * C_i \tag{1}$$

In this case, the calculation is expressed as follows.

```
Do i = 1, ni
 R (ir (i)) = R (ir (i)) + S (is (i))*Cl(i)
End do
```

A simple implementation of the subroutine interpolate_data on this example is shown in Fig. 5. Jcup_get_local_operation_index(Table 3) is an API subroutine for obtaining number of iterations *ni* and index conversion tables *is* and *ir* from arguments rname, sname, and mapping_tag.

The example shown by Fig. 2 was a data copy case, and an interpolation coefficient was not needed. However, in the case considered here, an interpolation coefficient should be taken into account. As for the interpolation coefficient, it is assumed that there is one coefficient for each correspondence of the grid points listed in the mapping table. In this case, it is only necessary to extract the interpolation coefficient used for the interpolation calculation of the local region from the array of interpolation coefficients of the global region, and the subroutine for this, jcup_get_local_coef, is provided by Jcup API (Table 4). In the example of Fig. 5, *Cg* means an array of interpolation coefficient for the global region that is calculated in advance

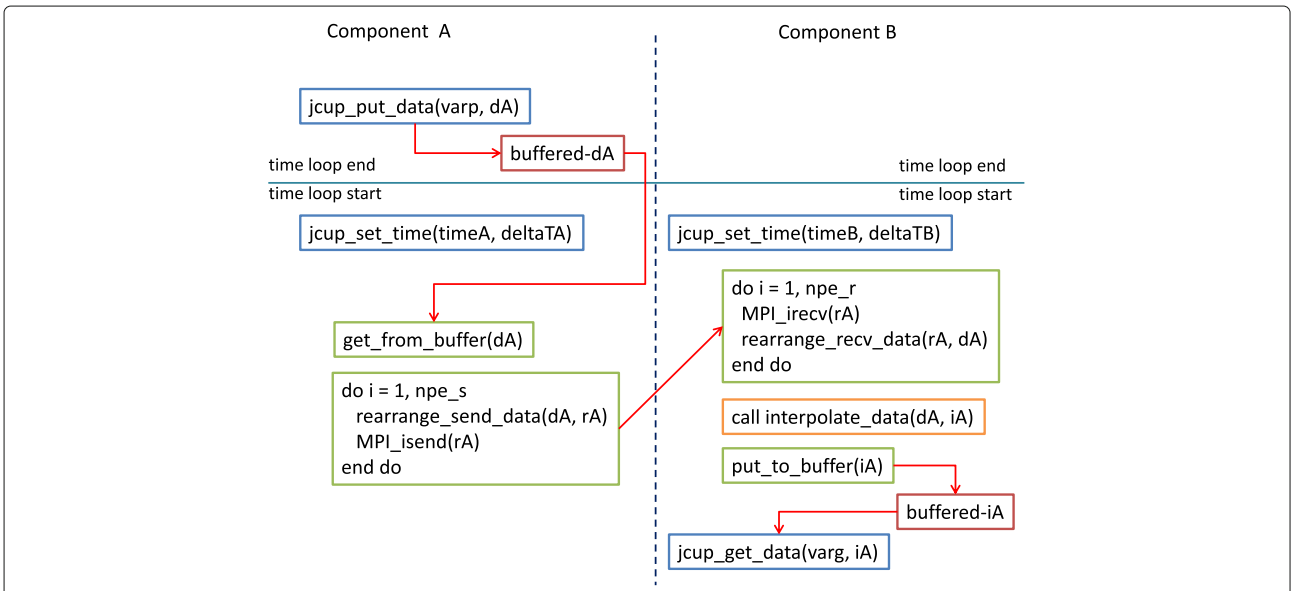

Fig. 4 Overview of API routine calls and data processing within Jcup. Blue squares indicate the API routines. Green squares mean data processing in Jcup. Orange square is a calling of interpolation subroutine interpolate_data in which the interpolation code must be implemented by users

Table 2 Interface of the interpolation subroutine interpolate_data

Subroutine name	Argument type	Argument name	Description
interpolate_data	Character(len=*), Intent(IN)	recv_comp_name	Name of receive component
	Character(len=*), intent(IN)	send_comp_name	Name of send component
	Integer, intent(IN)	mapping_tag	Identifier of mapping table
	Integer, intent(IN)	sn1, sn2	Array size of send_data
	Real(kind=8), ,intent(IN)	send_data(sn1, sn2)	Array of send data
	Integer, intent(IN)	rn1, rn2	Array size of receive data
	Real(kind=8), intent(INOUT)	recv_data(rn1, rn2)	Array of receive data
	Integer, intent(IN)	num_of_data	Number of data
	Integer, intent(IN)	tn	Number of data tag
	Integer, intent(IN)	data_tag(tn)	Array of data tag

and stored in the module "mod_common." By calling the subroutine jcup_set_local_coef, localized interpolation coefficient *Cl* is obtained. In the figure, for simplicity, these subroutines are called in interpolate_data. However, since these subroutines need only to be called once, they are usually called during the initialization process.

Application case studies

In this section, we describe cases where Jcup was applied, thereby showing that the Jcup features mentioned above are effective for coupling various models.

MIROC coupling

The first case is the coupling of the atmospheric and ocean models in the climate model MIROC. MIROC is a global climate model jointly developed by the University of Tokyo, Japan Agency for Marine-Earth Science and Technology and the Institute for Environmental Studies. It is a representative climate model in Japan, and its results have been referenced in the Intergovernmental Panel on Climate Change report(Watanabe et al. 2010). MIROC consists of four subcomponents: atmosphere, ocean, land surface, and rivers, and the atmospheric model and ocean model are coupled by MIROC's proprietary coupling

```
subroutine interpolate_data(rname, sname, mapping_tag, &
                            sn1, sn2, send_data,
                            rn1, rn2, recv_data,
                            num_of_data,
                            tn, data_tag)
  use mod_common, only : Cg ! global interpolation coefficient
  integer :: ni
  integer, pointer :: is(:), ir(:)
  real(kind=8), pointer :: Cl(:)
  integer :: i, j

  call jcup_get_local_operation_index(rname, sname, mapping_tag, &
                                      ni, is, ir)
  call jcup_set_local_coef(rname, sname, mapping_tag, Cg, Cl)

  recv_data = 0.d0
  do j = 1, num_of_data
   do i = 1, ni
     recv_data(ir(i), j) = recv_data(ir(i), j) + send_data(is(i), j)*Cl(i)
   end do
  end do

end subroutine interpolate_data
```

Fig. 5 A simple implementation example of the subroutine interpolate_data

Table 3 Interface of jcup_get_local_operation_index

Subroutine name	Argument type	Argument name	Description
jcup_get_local_operation_index	character(len=*), intent(IN)	recv_comp_name	Name of receive component
	Character(len=*), intent(IN)	send_comp_name	Name of send component
	Integer, intent(IN)	mapping_tag	Mapping table indentifier
	Integer, intent(OUT)	ni	Number of local iterations
	Integer, pointer	is(:)	Conversion table of send grid
	Integer, pointer	ir(:)	Conversion table of receive grid

software. The first test case in the development of Jcup is to replace the original coupling code of MIROC with Jcup. Importantly, when replacing the coupling code with Jcup, the calculation results do not change before or after the replacement, that is, the results of both are identical at the binary level. By guaranteeing matching at the binary level, there will be no bug in Jcup internally or in the coupling code using Jcup.

In the coupling of the original MIROC code, the coupling procedure consists of three steps: (1) collecting the data on the root processor and transmitting it to the root processor of the other component, (2) executing the interpolation calculation on the root processor of the receiving-side component, and (3) distributing the result to each rank. Meanwhile, in Jcup, data exchange is performed local-to-local, and the interpolation calculation is individually performed for each rank on the receiving side.

Since there is no change in the values during data transfer, the interpolation calculation must at least be the same, including the calculation order, to match the results at the binary level.

This was made possible by the design of Jcup in that users can arbitrarily implement an interpolation calculation code. A part of the interpolation calculation code implemented in MIROC is shown in Fig. 6.

The interpolation code is composed of calculation formulas, including branching by IF statement and the addition and multiplication of constants. This code was ported to Jcup's interpolation calculation subroutine interpolate_data with only minor modifications, such as changing the name of the variable. As a result, it was possible to replace the coupling code of MIROC with Jcup while maintaining binary compatibility. (This coupling was conducted as a Jcup test, and Jcup is not used in the current MIROC (MIROC6) atmosphere-ocean coupling.)

NICAM-COCO coupling

As a second example, we describe coupling of another atmospheric model and ocean model. The target atmospheric model is NICAM. NICAM is a global atmospheric model that uses a non-hydrostatic equation system (Tomita and Satoh 2004; Satoh et al. 2008; Satoh et al. 2014). The discretization method is a finite volume method, and the grid system employs an icosahedral grid. This model has been mainly utilized for research on phenomena with time scales of several days to months, such as typhoons and MJO, and in such research, simulation is carried out with the atmosphere-only model by specifying the sea surface temperature (SST) or assuming a slab ocean model (Miyakawa et al. 2014). However, NICAM is expected to be applied not only to such short- to medium- term simulations but also to climate simulations with time scales of several decades to centuries (Kodama et al. 2015; Haarsma et al. 2016). In such long-term simulations, although the current simulations are conducted with the atmosphere-only model under the specified sea surface temperature (SST) condition, it is natural to extend NICAM to be coupled with an ocean model for use as an atmosphere-ocean coupled model to internally reproduce SST in the model. The ocean model coupled to NICAM in this study was COCO (Hasumi 2006). COCO is a global ocean model that uses the Boussinesq approximate hydrostatic equation and a general orthogonal coordinate system. The version of COCO used for this coupling adopts the tri-polar grid. The tri-polar grid has an irregular grid shape, such that the North

Table 4 Interface of jcup_set_local_coef

Subroutine name	Argument type	Argument name	Description
jcup_set_local_coef	character(len=*), intent(IN)	recv_comp_name	Name of receive component
	Character(len=*), intent(IN)	send_comp_name	Name of send component
	Integer, intent(IN)	mapping_tag	Mapping table indentifier
	Real(kind=8), intent(IN)	Cg(:)	Global coefficient
	Real(kind=8), pointer	Cl(:)	Local coefficient

```
DO N = 1, IJ_OMAX
   DO M = IJ_OHEAD(N), IJ_OHEAD(N+1) - 1
         LA = IJRECOV_O2A(M)
         IF (BFSOCNG(LA).GT.0.0D0
  $          .and.BFOxxG(LA,2).GT.0.0D0) THEN
           LO   = IJO2C(LA,N)
           AOCN2 = SOCN(LA,N) * BFOyyG(LO,2)
  &                / (BFSOCNG(LA)* BFOxxG(LA, 2))
           BFOxxG(LA,3) = BFOxxG(LA,3)
  &           + ( BFOyyG (LO,3) * 1.D-2 ) *AOCN2
           BFOxxG(LA,4) = BFOxxG(LA,4)
  &           + ( BFOyyG (LO,4) * 1.D-2 ) *AOCN2
          BFOxxG(LA,5) = BFOxxG(LA,5)
  &           + ( BFOyyG (LO,5) + kelvin )*AOCN2
           BFOxxG(LA,6) = BFOxxG(LA,6)
  &           + ( BFOyyG (LO,6) + kelvin )*AOCN2
          ENDIF
    END DO
END DO
```

Fig. 6 A part of MIROC interpolation code. Owing to the design of Jcup, this code could be ported directly to the Jcup interpolation subroutine

pole is stretched to the North American and Eurasian continents. In fact, a study using the NICAM-COCO coupled model with Jcup has already been conducted by Miyakawa et al. (2017).

In this way, both NICAM and COCO are irregular grid-system models covering a spherical surface, and a large number of calculations are required to determine the grid point correspondence and interpolation coefficients necessary for coupling. This is because it is necessary to search for polygons, including individual grid points, in the calculation of the grid correspondence. In addition, the calculation of the interpolation coefficients must account for the preservation of water, energy balance, etc., during the time integration. An interpolation algorithm that satisfies the preservations is formulated by Jones (1999). According to Jones, the interpolation coefficient is represented by the area ratio of the overlapping portion of the polygon formed by each grid point. The calculation algorithm and execution performance are reported in Arakawa et al. (2014). As reported in that paper, a large number of calculations are required to determine the mapping table and coefficients. Meanwhile, in meteorological simulations, the grid shape is generally constant, irrespective of time, and the pattern of resolution is also generally limited. For example, in NICAM, there is a constraint condition in which the number of grid points is specified by the power of two, and the pattern of resolution that can normally be employed is limited to five to six patterns.

In summary, in NICAM-COCO coupling, it can be said that the mapping table calculation might be computationally expensive compared with processing the pre-calculated table, and the required pattern is limited. For these reasons, the design of Jcup in which the mapping table is calculated in advance and used as input information is well adapted to NICAM-COCO coupling.

In the experiment, three cases were considered according to the number of horizontal grid points of NICAM, which could be calculated by the function $10 * (2^{glevel})^2$. The correspondence between the cases and grid points is shown in Table 5

Experimental conditions are listed in Table 6. The conditions of COCO were fixed in all experiments. Size m52 corresponded to the number of horizontal grid points 360 (West– East) × 256 (South–North). The number of the process was 16, and ΔT was 15 min. The number of NICAM processes was represented by the r-level as equation $10 * 4^{rlevel}$. ΔT and duration time are shown in the table. The data exchange interval is 1 h.

The results of the performance measurement are shown in Fig. 7. The bar graph indicates the number of days that can be simulated by the one day calculation. The line graph is a scaling factor on the basis of 10 (or 40) processes.

The most notable difference between the cases was the state of the change in the scaling factor. In case 1, the scaling factor was reduced according to the number of processors, but it was almost constant in case 3.

Two reasons can be postulated for this difference:

1. Efficiency of NICAM itself
   The number of grid points per processor was smaller in case 1 than in case 2 or 3, as shown in Table 7. Therefore, the calculation time, which was scalable, became relatively short, and the non-parallelized process became longer. This could have caused the low scalability.
2. Load imbalance between NICAM and COCO
   In case 1, the time step calculation of NICAM was faster, and NICAM had to wait for data from COCO. Therefore, the execution time did not decrease with an increase in the number of processors. In contrast, in case 3, NICAM did not need to wait for the data. Therefore, the execution time was determined by the number of processors assigned to NICAM.

Table 5 Correspondence between the cases and grid points

Case	glevel	Grid points
Case 1	5	10240
Case 2	7	163840
Case 3	9	2621440

Table 6 Experimental conditions of NICAM–COCO coupling

		Case 1			Case 2			Case 3		
COCO	Size	m52	m52	m52	m52	m52	m52	m52	m52	m52
	Process	16	16	16	16	16	16	16	16	16
	ΔT[min]	15	15	15	15	15	15	15	15	15
NICAM	glevel	5	5	5	7	7	7	9	9	9
	rlevel	0	1	2	0	1	2	1	2	3
	Process	10	40	160	10	40	160	40	160	640
	ΔT[min]	15	15	15	4	4	4	1	1	1
	Duration time [day]	10	10	10	2	2	2	1	1	1

To confirm these assumptions, the execution time of NICAM's time integration loop is listed in Table 8. "Main ALL" is the execution time of the entire time-integration loop. "Coupler Put" is the time required to move the data from NICAM to the coupler, and "Coupler Get" is the time required for the reverse process. "Coupler Exchange" is the time required for the data exchange; the load imbalance is included in this time. "Atmos" is the calculation time without coupling.

To examine the scalability of NICAM itself, we calculated the scalability factor from the execution time of "Atmos". The result is shown in Fig. 8. In case 1, the scalability factor decreased, even though there was no data exchange.

However, it should be noted that with the examination of the load imbalance, the time "Coupler Exchange" in case 1 was significantly longer than that in the other cases. This suggests that the reception waiting occurred on the NICAM side during the data exchange. The latency became larger as the execution time was reduced so as to keep the time of "Main All" constant. This constant time was determined by the execution time of COCO, and it can be concluded that the decrease in the parallel efficiency was mainly caused by the load imbalance.

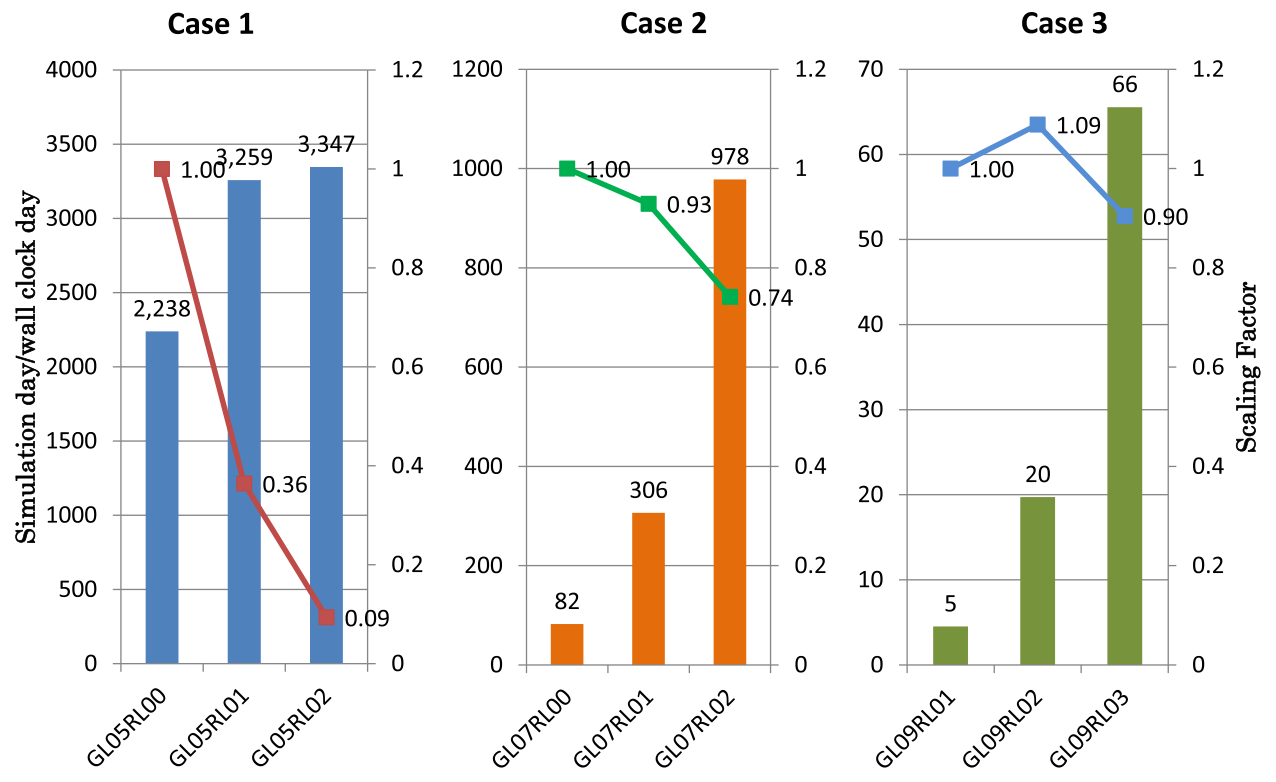

Fig. 7 Results of performance measurement on NICAM–COCO coupling. Bar graph is the number of days that can be simulated in 1 day's calculation. Line graph is a scaling factor

Table 7 Number of horizontal grid points per processor

	Case 1			Case 2			Case 3		
glevel	5	5	5	7	7	7	9	9	9
rlevel	0	1	2	0	1	2	1	2	3
The number of grid points	1024	256	64	16384	4096	1024	65536	16384	4096

In contrast to case 1, the scaling factor in high-resolution case 3 remained mostly constant. Therefore, it can be concluded that the effect of coupling was minimal.

NICAM-IO coupling

The coupling of NICAM and IO components shows that "users can implement their own interpolation code," which is one of the features of Jcup features that worked effectively.

As mentioned above, NICAM is an icosahedral grid model. This grid system may be inconvenient for post-process visualization and analysis. This is because many of the visualization and analysis tools that are currently widely available are predicated on the latitude-longitude grid (although the situation is improving).

For this reason, a program was developed to convert the results of NICAM into a latitude-longitude grid. This program is called NICAMIO. The overview of NICAMIO is shown in Fig. 9. In the figure, the area surrounded by the blue square represents NICAMIO.

NICAMIO operates in parallel with NICAM in multiple processes, simultaneously converting the result to the latitude-longitude grid with the calculations of NICAM and outputting them to files. Jcup is utilized to couple NICAM and NICAMIO, and the grid conversion code is implemented in the Jcup interpolation subroutine interpolate_data.

The interpolation algorithms implemented in NICAMIO are the following:

1. Area weighting method
2. Distance weighting method by three grid points
3. Nearest neighbor method

Among these three methods, the nearest neighbor approximation is a method that was later added owing to user request, and the interpolation code implementation is an example in which Jcup's feature that interpolation code can be freely implemented worked effectively. Figure 10 illustrates the outline of the distance weighting method and the nearest neighbor method used in NICAMIO. The values of the latitude-longitude grid represented by blue circles are calculated from the three points of the NICAM grid represented by orange squares and the coefficients that are inversely proportional to the distances. Here, the grid point indexes and coefficients are calculated and given in advance. For the nearest neighbor approximation method, the value with the largest coefficient among the three coefficients in the distance weight method is adopted. To be precise, this algorithm does not necessarily obtain the nearest value. This is because when the grid is irregular, there may be points closer to the three points used in the calculation. Nevertheless, this three-point method was adopted because it is not necessary to search for the nearest point exactly, and the accuracy required by the user is sufficient. In this way, the design that the interpolation code can be freely implemented has made it possible to flexibly respond to user needs.

Seismic model-structure model coupling

The coupling of the seismic model and the structure model can be easily implemented, depending on the conditions, although the model grid is complex. The structural model used here is FrontISTR ++, which has an unstructured grid employing the finite element method (Okuda 2019), and the seismic model is Seism3D, which is a finite difference model with a regular rectangular grid (Furumura 2005). The coupled calculation of these models involves the conversion of the ground motion speed calculated by Seism3D into the displacement by Jcup and transmission to FrontISTR++, where it is considered to

Table 8 Execution time of NICAM (s)

	Case 1			Case 2			Case 3		
glevel	5	5	5	7	7	7	9	9	9
rlevel	0	1	2	0	1	2	1	2	3
Main ALL	386	265	258	2096	564	177	19070	4379	1318
Coupler Put	2	3	3	3	12	6	305	127	106
Coupler Exchange	36	114	172	4	1	1	18	17	3
Coupler Get	1	2	3	1	1	1	2	6	19
Atmos	338	141	75	2039	538	163	17613	3969	1078

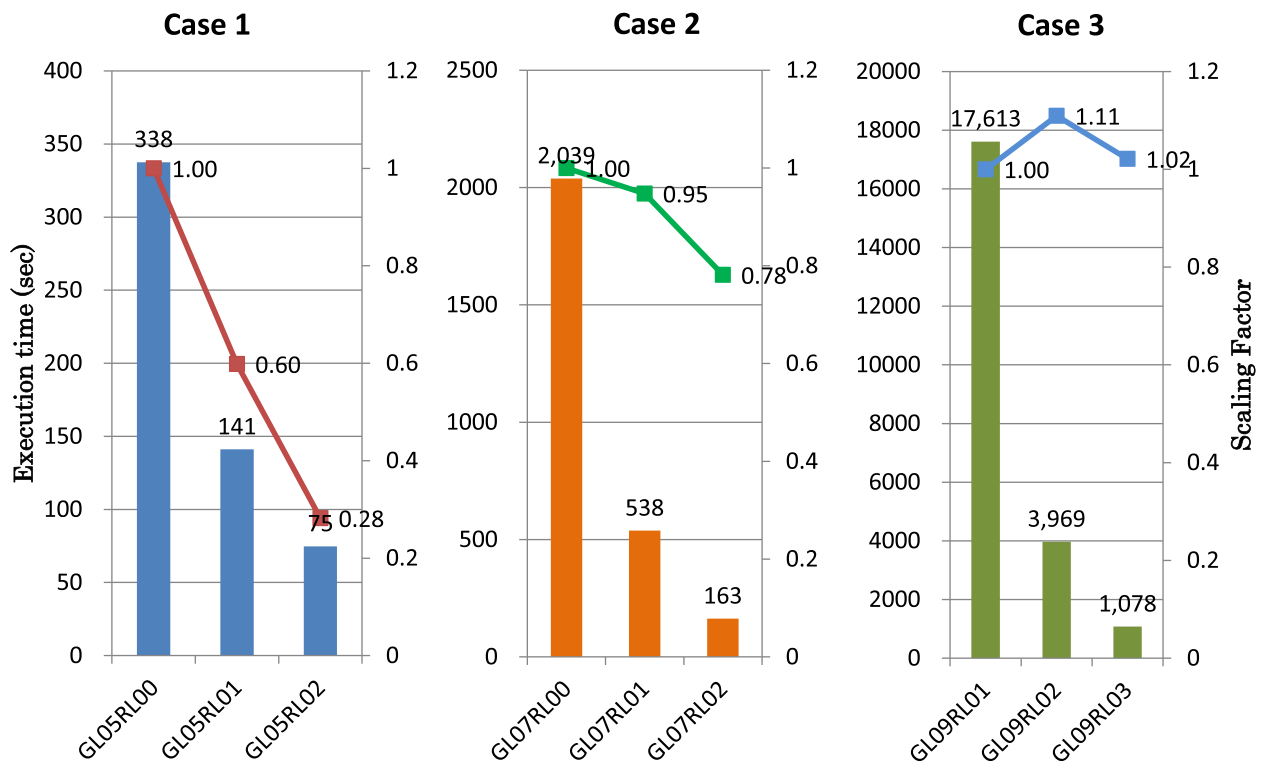

Fig. 8 Execution time and scaling factor of "Atmos." Bar graph illustrates execution time (s). Line graph represents scaling factor

vibrate the building (Matsumoto et al. 2015). Figure 11 shows the grid used by the structure model for this calculation. The grid is largely divided into a part representing the building and a part representing the ground, and the ground grid is embedded in the seismic model grid. In this case, the data exchange is one-way, from the seismic model to the structure model, and physical quantity conservation in the interpolation calculation does not have to be strict, in contrast to the NICAM-COCO case. The seismic model has a regular rectangular grid in which

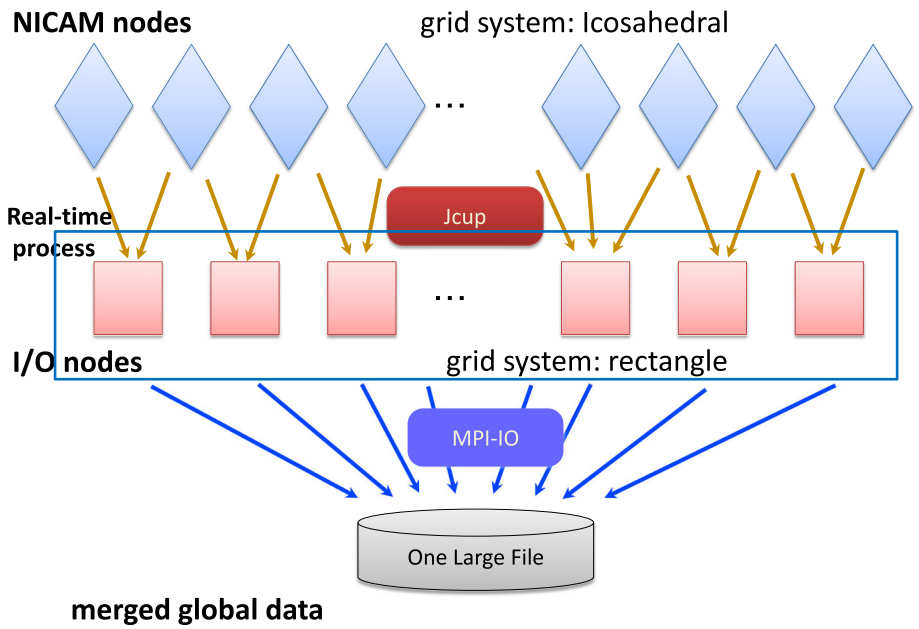

Fig. 9 The overview of IO component for NICAM. A blue square indicates NICAMIO process

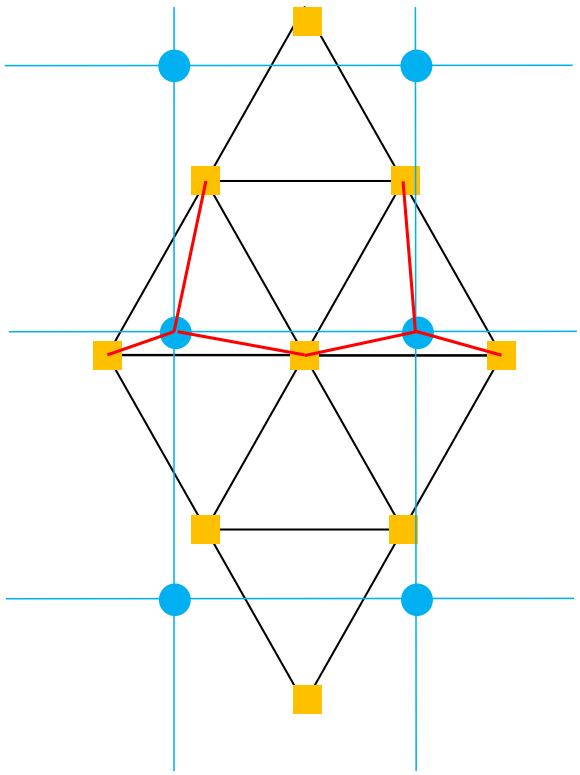

Fig. 10 The interpolation method from NICAM grid to lat-lon grid. Orange squares indicate NICAM grid points and blue circles indicate lat-lon grid points. The lat-lon grid point value is calculated from the values of the vertices of the triangle of the NICAM grid in which the point is included

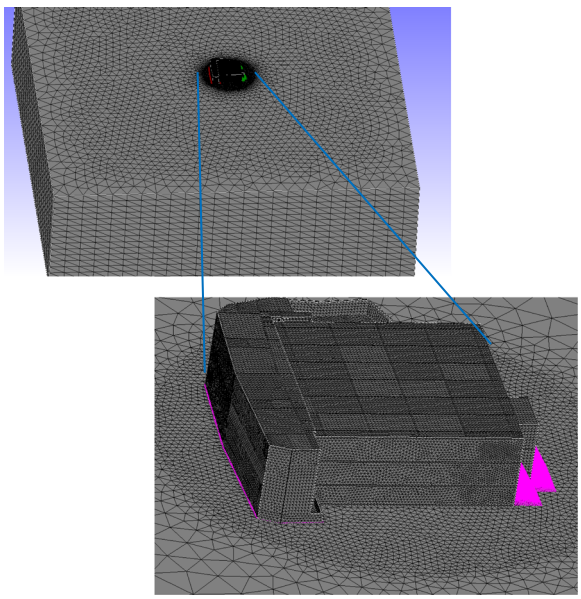

Fig. 11 Grid shape of the structure model FrontISTR++ used in the seismic model-structure model coupling. A grid representing a building is set at the center of the grid representing the ground

ΔX, ΔY, and ΔZ are constant. Therefore, the grid point indexes of the seismic model surrounding the individual grid points of the structure model can be easily calculated, and the interpolation coefficient can be calculated by tri-linear interpolation from the eight grid points of the seismic model surrounding the grid points of the structure model. As shown in this case, by using a mapping table as an input, even if the model grid is unstructured and has a complicated shape, it is possible to easily couple them depending on the conditions.

Conclusions

Features and future of a coupling library Jcup

In this paper, we discussed the design concept and implementation of a coupling library, Jcup, to indicate how coupling software should be implemented under the conditions listed below. The conditions are as follows: (1) developer does not belong to a specific research community, and the software is used generically. (2)Coupling software must adapt to changes in the grid system and interpolation algorithm, and the possibility of bug contamination should be as low as possible.

Conceptual answers to these issues can be summarized as follows: (1) dividing the function of the coupling software into those that change the values and those that do not change the values. (2) Enabling users to manage and implement the code in which values are changed as a glass box.

Based upon this basic concept, Jcup is constructed so that (1) correspondence relations of grid points in the interpolation calculation (mapping table) are utilized as input information, and (2) interpolation calculation codes can be freely implemented by the user.

Through these features, Jcup has high flexibility with respect to coupling various components, but there are restrictions on the timing of data exchange. That is, data exchange is only performed at predetermined time intervals for each element of data, and the model time must match each exchange time. This constraint might not be a problem when the time constant of interaction is long, such as in atmosphere-ocean coupling. However, when component Δ_T varies irregularly, and interaction is required on such a time interval, the current Jcup cannot cope. Therefore, the next modification for Jcup is to remove this restriction on the data exchange interval and enable the coupling at an arbitrary time interval.

Concluding remark

Jcup is not software for easily coupling specific models but is designed and implemented as a library that provides a wide range of users with limited functions. Such a minimal approach would not necessarily match the direction of development of many modern coupling software. However, considering the continuity and safety of usage

under a specific relationship between the developer and user communities, we think that the design philosophy of Jcup has a certain generality and usefulness.

Acknowledgements
This research was supported by the Integrated Research Program for Advancing Climate Models (TOUGOU program) of the Ministry of Education, Culture, Sports, Science and Technology, Japan, and is supported by the Japan Science and Technology Agency/Core Research for Evolutional Science and Technology, and the Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning. This research used the computational resources of the High Performance Computing Infrastructure (HPCI) system provided by the Information Technology Center of the University of Tokyo, through the HPCI System Research Project (Project ID:hp120190).

Authors' contributions
T.A is in charge of designing and coding Jcup and has written this paper. T.I is a code reviewer of Jcup and also implemented a coupling code of NICAM-COCO and NICAM-IO. Y.H conducted the coupled calculation of NICAM and COCO and measured and evaluated its performance. M.S was the development manager of NICAM and led the development of a coupled system between NICAM and other components. All authors read and approved the final manuscript.

Funding
This research was supported by the Integrated Research Program for Advancing Climate Models (TOUGOU program) of the Ministry of Education, Culture, Sports, Science and Technology, Japan, and is supported by the Japan Science and Technology Agency/Core Research for Evolutional Science and Technology, and the Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning.

Availability of data and materials
The version of Jcup described in this paper is v.3.150100. Jcup code (doi:10.5281/zenodo.1297240) is available from github website. In addition, for readers who want to tray Jcup, sample programs (doi:10.5281/zenodo.1297250) also available from github website. The urls of github are as follows. https://github.com/Jcuplib/jcup/releases/tag/3.150100. https://github.com/Jcuplib/jcup_sample/releases/tag/3.150100.

Competing interests
The authors declare that they have no competing interest.

Author details
[1]Research Organization for Information Science and Technology, 1-18-16 Hamamatsu-cho, Minato-ku, Tokyo 105-0013, Japan. [2]Japan Agency for Marine-Earth Science and Technology, 3173-25 Showa-machi, Kanazawa-ku, Yokohama 236-0001, Japan. [3]National Institute for Environmental Studies, 16-2 Onogawa, Tsukuba, Ibaraki 305-8506, Japan. [4]Atmosphere and Ocean Research Institute, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba 277-8564, Japan.

Received: 23 May 2019 Accepted: 30 December 2019
Published online: 06 February 2020

References

Adcroft A., Campin J.-M., Hill C., Marshall J. (2004) Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube. Mon. Weather Rev. 132:2845–2863

Arakawa T., Yoshimura H., Saito F., Ogochi K. (2011) Data exchange algorithm and software design of KAKUSHIN coupler Jcup. Procedia Comput. Sci. 4:1516–1525

Arakawa T., Inoue T., Satoh M. (2014) Performance evaluation and case study of a coupling software ppopen-math/mp. Procedia Comput. Sci. 29:924–935. https://doi.org/10.1016/j.procs.2014.05.083

Baba Y., Takahashi K., Sugimura T., Goto K. (2010) Dynamical core of an atmospheric general circulation model on a yin-yang grid. Mon. Weather Rev. https://doi.org/10.1175/2010MWR3375.1

Craig A., Valcke S., Coquart L. (2017) Development and performance of a new version of the OASIS coupler OASIS3mct3.0. Geosci. Model Dev. 10:3297–3308. https://doi.org/10.5194/gmd-10-3297-2017

Furumura T. (2005) Large-scale parallel simulation of seismic wave propagation and string ground motion for the past and future earthquakes in Japan. J. Earth Simul. 3:29–38

Haarsma R. J., Roberts M., Vidale P. L., Senior C., Bellucci A., Corti S., Fuckar N., Guemas V., von Hardenberg J., Hazeleger W., Kodama C., Koenigk T., Leung R., Lu J., Luo J.-J., Mao J., Mizielinsky M., Mizuta R., Nobre P., Satoh M., Scoccimarro E., Semmler T., Small J., von Storch J.-S. (2016) High resolution model intercomparison project (highresmip v1.0) for cmip6. Geosci. Model Dev. 9:4185–4208. https://doi.org/10.5194/gmd-2016-66

Hanke M., Redler R., Holfeld T., Yastremsky M. (2016) Yac 1.2.0: new aspects for coupling software in earth system modelling. Geosci. Model Dev. 9:2755–2769. https://doi.org/10.5194/gmd-9-2755-2016

Hasumi H. (2006) CCSR Ocean Component Model (COCO) Version 4.0. Center for Climate System Research Report. vol 25:103. https://ccsr.aori.u-tokyo.ac.jp/~hasumi/COCO/coco4.pdf

Hill C., DeLuca C., Balaji V., Suarez M., DaSilva A., ESMFJointSpecificationTeam (2004) The architecture of the earth system modeling framework. Comp. Sci. Eng. 6:12–28

Jacob R., Larson J., Ong E. (2005) Mxn communication and parallel interpolation in community climate system model version 3 using the model coupling toolkit. Int. J. High Perform. Comput. Appl. 19(3):293–307. https://doi.org/10.1177/1094342005056116

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

Kodama C., Yamada Y., Noda A. T., Kikuchi K., Kajikawa Y., Nasuno T., Tomita T., Yamaura T., Takahashi T. G., Hara M., Kawatani Y., Satoh M., Sugi M. (2015) A 20-year climatology of a NICAM AMIP-type simulation. J. Meteor. Soc. Japan 93:393–424. https://doi.org/10.2151/jmsj.2015-024

Larson J., Jacob R., Ong E. (2005) The model coupling toolkit: a new Fortran90 toolkit for building multiphysics parallel coupled models. Int. J. High Perform. Comput. Appl. 19(3). https://doi.org/10.1177/1094342005056115

Liu L., Zhang C., Li R., Wang B., Yang G. (2018) C-coupler2: a flexible and user-friendly community coupler for model coupling and nesting. Geosci. Model Dev. 11:3557–3586. https://doi.org/10.5194/gmd-11-3557-2018

Matsumoto M., Arakawa T., Kitayama T., Mori F., Okuda H., Furumura T., Nakajima K. (2015) Multi-scale coupling simulation of seismic waves and building vibrations using ppopen-hpc. Procedia Comput. Sci. 52:1514–1523. https://doi.org/10.1016/j.procs.2015.05.341

Miyakawa T., Satoh M., Miura H., Tomita H., Yashiro H., Noda A. T., Yamada Y., Kodama C., Kimoto M., Yoneyama K. (2014) Madden-Julian Oscillation prediction skill of a new-generation global model. Nat. Commun. 5:3769. https://doi.org/10.1038/ncomms4769

Miyakawa T., Yashiro H., Suzuki T., Tatebe H., Satoh M. (2017) A Madden-Julian Oscillation event remotely accelerates ocean upwelling to abruptly terminate the 1997/1998 super El Nino. Geophys. Res. Lett. 44:9489–9495. https://doi.org/10.1002/2017GL074683

Okuda H. (2019) Nonlinear structual analysis open software FrontISTR. https://frontistr-commons.gitlab.io/FrontISTR/manual_en/index.html

Satoh M., Matsuno T., Tomita H., Miura H., Nasuno T., Iga S. (2008) J. Comput. Phys. Spec. Issue Predicting Weather Clim. Extreme Events 227:3486–3514. https://doi.org/10.1016/j.jcp.2007.02.006

Satoh M., Tomita H., Yashiro H., Miura H., Kodama C., Seiki T., Noda A. T., Yamada Y., Goto D., Sawada M., Miyoshi T., Niwa Y., Hara M., Ohno T., Iga S-i., Arakawa T., Inoue T., Kubokawa H. (2014) The non-hydrostatic icosahedral atmospheric model: description and development. Prog. Earth Planet. Sci.:1–18. https://doi.org/10.1186/s40645-014-0018-1

Tomita H., Satoh M. (2004) A new dynamical framework of nonhydrostatic global model using the icosahedral grid. Fluid Dyn. Res. 34:357–400

Valcke S., Budich R., Carter M., Guilyardi E., Foujols M.-A., Lautenschlager M., Redler R., Steenman-Clark L., Wedi N (2006) The PRISM Software Framework and the OASIS Coupler. In: Hollies A. J., Kariko A. P. (eds). (ACCESS) - Changes and Opportunities, BMRC Research Report. Bur. Met., Australia. pp 132–140

Valcke S. (2013) The OASIS3 coupler: a European climate modelling community software. Geosci. Model Dev. 6:373–388. https://doi.org/10.5194/gmd-6-373-2013

Washington W. M., Buja L., Craig A. (2009) The computational future for climate and Earth system models: on the path to petaflop and beyond. Phil. Trans. R. Soc. A 367:833–846

Watanabe M., Suzuki T., O'ishi R., Komuro Y., Watanabe S., Emori S., Takemura T., Chikira M., Ogura T., Sekiguchi M., Takata K., Yamazaki D., Yokohata T., Nozawa T., Hasumi H., Tatebe H., Kimoto M. (2010) Improved climate simulation by MIROC5: mean states, variability, and climate sensitivity. J. Clim. 23:6312–6335. https://doi.org/10.1175/2010JCLI3679.1

Yamazaki D., Sato T., Kanae S., Hirabayashi Y., Bates P. D. (2014) Regional flood dynamics in a bifurcating mega delta simulated in a global river model. Geophys. Res. Lett. 41:3127–3135. https://doi.org/10.1002/2014GL059774

Yoshimura H., Yukimoto S. (2008) Development of a simple coupler (Scup) for earth system modeling. Pap. Met. Geophys. 59:19–29

Publisher's Note