Design and Research of CAS-CIG for Earth System Models

T. Wang1,2, J. Jiang1, M. Zhang3,4, H. Zhang3, J. He3, H. Hao1, and X. Chi1

1High Performance Computing Department, Computer Network Information Center, CAS, Beijing, China, 2University of Chinese Academy of Sciences, Beijing, China, 3International Center for Climate and Environment Science, CAS, Beijing, China, 4School of Marine and Atmospheric Sciences, State University of New York at Stony Brook, Stony Brook, NY, USA

Abstract The Chinese Academy of Sciences Coupling Interface Generator (CAS-CIG) is designed to address the complexities of the development and coupling of different component models in Earth System Models based on the Coupler 7 of the Community Earth System Model (CESM). Its application in the Chinese Academy of Sciences Earth System Model (CAS-ESM) is described. The CAS-CIG automatically generates the coupler code through a simple configuration file when a component model is accessed, enabling different component models to be easily ported to CPL7 to create simulation cases. Combined with the automatic generation of compile scripts, the precompilation and run directories are directly formed. The component model integration, model selection, experimental setup, and platform migration can be all accomplished in the CAS-CIG. Verification of the CAS-CIG is presented to show that the automatically generated codes can identically reproduce the simulation results of CAS-ESM. CAS-CIG presents a software tool for modeling centers to investigate the impact of component model selections on simulations of climate and weather.

1. Introduction

Earth System Model is the cornerstone of the study on global climate change. A coupler, which links component models together and handles data transfer between different models and different grids, is an important tool for Earth System Models. Currently, there are several couplers available from major modeling institutions around the world, for example, the CPL6 coupler (Craig et al., 2005) designed by NCAR (National Center for Atmospheric Research) for the Community Climate System Model Version 3 (CCSM3) (Collins et al., 2006), the CPL7 (Craig et al., 2012) designed by NCAR for the Community Climate System Model Version 4 (CCSM4) (Gent et al., 2011) and the Community Earth System Model (CESM) (Hurrell et al., 2013), the OASIS (Ocean Atmosphere Sea Ice Soil) coupler (Craig et al., 2017; Redler et al., 2010; Valcke, 2013) designed by CERFACS, the Model Coupling Toolkit (MCT; Jacob et al., 2005; Larson et al., 2005), the Earth System Modeling Framework (ESMF; Hill et al., 2004), the Flexible Modeling System (FMS) coupler (Balaji et al., 2006), the Yet Another Coupler (YAC; Hanke et al., 2016), and the Community Coupler (C-Coupler; Y. Liu et al., 2014, 2018). OASIS and C-Coupler support flexible and pluggable access in component models. National Centers for Environmental Prediction (NCEP) proposed a concept of component model standardization called NUOPC (Theurich et al., 2016). NCEP implements ESMF according to NUOPC. However, significant effort is still needed by scientists outside these modeling centers to implement a new component using these couplers and framework. For example, when using ESMF to integrate a component model, a cap needs to be written by the user. Implementing a cap still requires a lot of work, even though the ESMF provides many standard libraries. CAS-CIG has a similar goal to Unified Forecast System (Tallapragada, 2018). We both hope to unify and integrate various component models through a top-level framework, simplify the development and use process, and form a more widely used earth system simulation platform.

The purpose of this paper is to describe a new coupling interface generator on top of the CPL7 coupler of the CESM. Unlike ESMF, in CAS-CIG, the user can use the configuration file to achieve automatic generation of coupling interface instead of writing code. We believe that using configuration files can reduce the time for users to understand the framework and reduce the amount of coding effort. The code of coupler is no longer fixed or needs to be manually modified but can be automatically generated according to certain rules and adapted to different component models. By using such a coupling generator, we hope to reduce the secondary development effort on the coupler during development process of component models. The wide use of
CESM by the community motivates the design of a flexible plug-and-play generator so that researchers can explore the effects of multiple combinations of different component models without spending significant efforts in software engineering. In this paper, we will introduce the general design of the CAS-ESM coupling interface generator (hereinafter referred to as the CAS-CIG) and provide the details of the interface design. The remainder of this paper is organized as follows: Section 2 describes the new features of CAS-CIG over the CPL7 coupler. Section 3 presents the general design of the CAS-CIG. Section 3.4 introduces the main module of the CAS-CIG. Section 3.5 reviews the technology of coupling interface generation. Section 3.6 details the experiments and provides the results. Section 4 summarizes this paper and discusses the future needs for the development of the CAS-CIG.

2. CAS-CIG Features and Its General Design

The CPL7 coupler has been used in CCSM4 and CESM1 which are two Earth System Models of the CMIP5 (Phase 5 of the Coupled Model Intercomparison Project). It inherits most of the characteristics from the CPL6 coupler. The CPL6 coupler includes a centralized coupler component, integration of MCT, and flux computation. It uses multiple executables for a coupled simulation case, where the coupler component forms a separate executable. Coupled models assembled with CPL7 form a single executable into which a top-level driver achieves various processor layout and time sequencing. Based on the CPL7 coupler, a variety of component sets can be used to simulate various simulation cases. Because CPL7 is a coupler designed for the component models in CESM, users need to do extra work to understand CPL7 code and do a lot of code modification when they integrate and develop other component models. In order to further improve the efficiency of collaborative development of Earth System Model, CAS-CIG is designed and implemented at the top level of CPL7.

In this section, we will introduce the general design of the CAS-CIG, which will also be used in follow-up development. We first describe the simulation case in CAS-CIG, then introduce its architecture, and explain the general software architecture. Finally, three different features of CAS-CIG and CPL7 are introduced.

2.1. A Simulation Case for the CAS-CIG

A simulation case for the CAS-CIG is a combination of component models and configurations that can be simulated on the CAS-CIG platform. It includes a series of configuration files, a certain set of component models, and some functional usage rules. All of these combined together can run and produce simulation results. Generally speaking, a simulation case can be, for example, a single active component model with other data models, a regional model coupled with an atmosphere model, all active and coupled models, etc.

2.2. Architecture of the Simulation Cases With the CAS-CIG

To achieve the target of simulating various simulation cases on the same software platform and integrating new pluggable component models, we have designed an architecture for the simulation cases with the CAS-CIG. Figure 1 shows an example of the CAS-CIG architecture with a typical simulation case to implement the integration of different component models, using the CAS-ESM as an example. In the CAS-ESM, atmosphere (ATM), land (LND), ocean (OCN), and sea ice (ICE) are the types of component models. "IAP AGCM" (Zhang et al., 2009, 2013) and "CAM" (Meehl et al., 2013) are the component models of atmosphere (ATM). "ColM" (Meng et al., 2009) and "CLM" (Dai et al., 2003; Lawrence & Chase, 2007) are the component models of land (LND). "LICOM" (Huang et al., 2014a, 2014b; Liu et al., 2013) and "POP" (Kerbyson & Jones, 2005; Leasure et al., 2011; Smith et al., 2010) are the component models of ocean (OCN). "CICE" (Hunke et al., 2013; Urrego-Blanco et al., 2016) is the component model of sea ice (ICE). "IAP AGCM," "ColM," and "LICOM" are independently developed by IAP-CAS. "CAM," "CLM," "POP," and "CICE" are inherited from CESM. In Figure 1, “WRF” (Hines & Bromwich, 2008), “DGVM,” “OBM,” and “CO2” are other new component models integrated into CAS-ESM. Each simulation case uses arrows of one color. The main ideas of this architecture are as follows.

1. All simulation cases that represent different component models are freely selectable. Orange arrow, purple arrow, and gray arrow represent the automatic generation process of each simulation case, respectively. Given the names, types, and codes of the component models required for coupling, the CAS-CIG can automatically generate the coupler interface and connect this simulation case to the software platform based on the information described in the configuration file. The blue and green parts
of Figure 1 indicate the component model codes that need to be connected to the platform. In addition, the red part of Figure 1, which is the configuration files, represents a series of required coupling information, and the basic information is the names and types of the component models shown in the blue and green parts. The component models in the blue part can be substituted by any other model, such as CAM, POP, and CLM.

2. The CAS-CIG Toolkits are the cores of the architecture. They analyze the configuration files, translate the template files, and generate the coupler codes while combining the component model codes with the coupler codes to generate a simulation case running directory that can be compiled and run.

3. The new component models developed by scientists and inherited from other platforms can also be integrated into the software platform seamlessly. The integration of the new component model may require more configuration information than that of the existing component model, such as coupling variables, mapping relationships, and merging processes. The CAS-CIG provides configuration file interfaces for all modules at all levels of the coupler that are required for the integration of the new component model.

2.3. General Software Architecture of CAS-CIG

Based on the above main ideas, we designed the software architecture of the CAS-CIG, which is shown in Figure 2. It consists of a configuration file parsing component (top left panel), an interface generation component (top right panel), and a compile script generation component (lower left panel). In the following, we introduce these three components.

Configuration file parsing component. The configuration files can be divided into three layers, labeled in yellow in Figure 2, which are the process configuration file, module configuration file and component configuration file. The process configuration file is mainly used for the automatic generation of the coupler driver code, which controls the coupling process. The coupling sequence is written into the process configuration file. The module configuration files are actually a series of files, including mapping, merging, and other sharing codes. The component configuration file controls the automatic generation of the component model coupling interface, including the declaration and transfer of coupled variables. A common configuration file is added in the upper layer, as shown in the red part of Figure 2.

There may be two typical types of parsing processes in different simulation cases when parsing configuration files. The first case is that the component models used in the simulation case are existing active or data
models that are supported by the coupler. In this case, when running a model scenario, only the common configuration file is required. Based on the component model names and types provided by the common configuration files, the configuration file parsing component automatically generates the intermediate parameters and variables that are required by the other layers of configuration files. The second case is that there is a new component model that needs to be connected to the software platform and coupled to run. Since the new component model may be a new component type, the configuration file parsing component may not be able to give default intermediate parameters and variables. The configuration files of the other layers must be provided by the user. Then, the configuration file parsing component will parse the configuration files of each layer in turn.

Interface generation component. The interface generation component is built on the top of the configuration file parsing component, which uses the result after parsing. It automatically generates the coupling interface based on each module template file and the results of the configuration file parsing of the coupler. The template files consist of many common code segments extracted from the coupler codes, where coupler codes are split into common codes and specific codes. The common code is the code that can be shared by various component models, and the specific codes are set based on the parsing result. The interface generation component can combine the template files and the parsing results, replace the renamable words of common codes, insert the specific codes, and generate the coupling interface codes.

Compile script generation component. In addition to the coupling interface code, compilation scripts and configurations are implemented by the script system in the CAS-CIG. In view of the usage habits and convenience of the script system, we have added a build script generation component in the CAS-CIG to automatically generate a scripting system to unify the usage of a simulation case. The compile script
configuration file mainly provides information such as the resolution, machine information, and case name. The compile script generator generates the script code while parsing the compile script configuration.

Based on the general design and architecture of CAS-CIG mentioned above, we can conclude that the CAS-CIG is developed to expand CPL7 by including the following three new features:

* **Automatic generation of the coupler codes and building scripts.** In the CAS-CIG, the coupler codes and the build scripts are dynamically generated instead of statically. By setting the configuration files, the CAS-CIG can automatically generate the coupler codes and the build scripts. Different coupling models, resolutions, coupling sequences, and other functional components can be selected on demand, and coupling variables and coupling interfaces can be inserted as needed. The implementation details of the algorithm also provide several optional generation methods and a default method. For example, given a case of all active models (Each type of models has an active component model that participates in coupling.), if there are several different algorithms for merging data from other component models, users can select one of them in the simulation case. To integrate an external component model, the model experts do not need to spend much time improving the coupler codes and the build scripts. They just set a few configuration files instead, and the CAS-CIG can automatically generate the coupler codes and the build scripts to adapt to the external component model.

* **Flexible and pluggable component models.** The CAS-ESM supports a wider range of component models than the CESM and can easily integrate new component models. In that case, the CAS-CIG extracts the main functions of the coupler and the building scripts such as mapping and merging between component models, coupling sequences, coupling interfaces and variables of component models, and resolutions and machine environment information of simulation cases. These are the functional modules in CESM and CPL7, which will be described in detail in section 3. CAS-CIG can generate the codes of these modules automatically. In fact, after including the above functions, CAS-CIG is a system that can be run independently on top of CAS-ESM. The models exist as a pluggable and independent system in CAS-CIG. When a user needs to use some component models for simulation experiments, the component models can be inserted to the simulation case, and they can be pulled out after the simulation experiment. Component models can be flexibly integrated to the CAS-CIG.

* **Simple and convenient operation.** The format of the configuration files used to generate the codes is the list format of the Python language. Compared to the XML format, the list format is simpler and clearer. In addition, due to the application of the intermediate variable generation technology, the configuration process of the configuration file is simplified. For the insertion of coupling variables and interfaces, the template structure is refined to facilitate configuration. Originally, users needed to read, modify, and add thousands of lines of codes to integrate a new component model; now they only need to modify dozens of lines of configuration files and run their simulation cases. Therefore, the operation of multiple simulation experiments and models integration is very simple and convenient.

### 3. Main Modules of CAS-CIG

CPL7 adopts a modular software architecture, and each module implements certain types of specific functions. In this section, we will introduce which main modules of the coupler are automatically generated in the CAS-CIG. In addition, we will briefly cover the automatic generation of build scripts and build directories. From this section, we will introduce some examples to help readers understand CAS-CIG. In order to ensure the logic and the continuity of the paper, all the examples mentioned below are taken from the coupling experiment of AGCM+COLM+LICOM+CICE.

#### 3.1. Coupling Process Module

The coupling process module is the top module of the coupling interface. This module controls the coupling relationship between the component models and the coupling process. When a component model needs to be integrated into the software platform, it is necessary to register variables and add the function calls for the data exchange with other component models in the coupling process module.

To realize the automatic generation of the coupling sequence module, it is necessary to analyze and extract the commonality of the coupling sequence module. The coupling sequence module includes the following main processes: pre-init, init, run, and finalize. For each component model, the pre-init and finalize
processes are similar, and the commonality is strong. The common codes can be directly extracted to form a template. The init process is different. In addition to processing the same initialization process of each component model, the mapping relationship between component models is also managed, and the mapping relationships of different component models may be different. For example, ATM needs to perform a two-way mapping with LND and OCN; ICE needs a one-way mapping with ATM, etc. Therefore, in addition to the simple common code extraction, this part also needs to retain the specific code generation interface of the mapping relationship. This involves the content of the mapping module, which will be introduced in the next subsection. Other than this, the run process is the most complicated process since it involves the coupling process and the sequence of the different component models.

Figure 3 shows the data flow between component models through CPL7. Due to the limitations of the component models supported by the CESM, CPL7 has some restrictions on the coupling order of the component models when executing the run process. For example, the ATM model must be run after other models. To maintain the flexibility of the CAS-CIG, we redesigned the run process. The new run process uses the coupling process configuration file to set the coupling order. Except that the GLC model must be coupled after the LND model, the other component models can be freely coupled at users' own run sequential choice.

3.2. Mapping Module
As mentioned in the previous subsection, the component model needs to configure the mapping relationship between component models when calling the init process. This function is mainly realized by calling the mapping module. The reason for mapping between component models is that component models use different resolutions and grids. Moreover, since the data transfers direction between the component model and the coupler is either one way or two way, there are also both one-way and two-way transfers in the mapping.

To implement the generation of the mapping module, we analyze the mapping code of each component model in the coupler and find that although there are different directions of mapping (For example, atmosphere and land need to be mapped bidirectional, while runoff and ocean need to be mapped one way.), the codes of mapping module in one of these directions are similar even if the component
models are different. We therefore decided to use the mapping code of the one-way process as a mapping module template, which usually includes the mapping_init and mapping_mct functions. The mapping relationship between component models is provided by the mapping module's configuration file. If the two component models need a two-way mapping, it can be achieved by the automatic generation of four one-way mapping codes.

When a new component model is coupled with the existing component models, some special algorithms and function calls may need to be written in the mapping module, so we set some insertion points in the template

```
!FORTRAN code templates
L1: !<list> key="cother"
L2: subroutine mrg_x2(c) _init_mct( cdata_{c} ,{cother}_{c} )
L3: type(seq_cdata) ,intent(in) :: cdata_{c}
L4: type(mct_aVect), intent(inout) :: {cother}_{c}
L5: type(mct_GsMap), pointer :: GSMap_{ccc}
L6: integer :: mpicom
L7: !</list>
......

L8: !<list> key="cother"
L9: call mct_aVect_init( {cother}_{c} , rlist=seq_flds_{cother} fields, lsize=lsize)
L10: call mct_aVect_zero( {cother}_{c} )
L11: !</list>
```

Figure 5. An example of merge templates.

```
models are different. We therefore decided to use the mapping code of the one-way process as a mapping module template, which usually includes the mapping_init and mapping_mct functions. The mapping relationship between component models is provided by the mapping module's configuration file. If the two component models need a two-way mapping, it can be achieved by the automatic generation of four one-way mapping codes.

When a new component model is coupled with the existing component models, some special algorithms and function calls may need to be written in the mapping module, so we set some insertion points in the template

```
L1: [case]
L2: casename=['B1850C5X_test']
L3: machine=['yuan']
L4: grids=['[ATM','fH4'], ['ATM','fH4'], ['OCN','licom'], ['ICE','licom']]
L5: [create_newcase]
L6: srcfile="create_newcase_template"
L7: desfile="create_newcase"
L8: allicomp = ['ATM','WRF','GEA','SRD','LND','ICE','OCN','CPL','GLC']
L9: comp = ['ATM','LND','ICE','OCN','CPL','GLC']
L10: everycomp = ['cam','wrf','gsea','datm','atmi','sarm','clm','dtas','gsn','cice','dice','xice','sice', 'po p2','licom','dcmn','xocn','socn','camdom','sgle','xgle','cism']
L11: onecomp = ['atm','ind','ocn','ice','glc']
......
```

Figure 6. An example of compile script configuration file.
of the mapping module for the user to add the specific codes. The insertion of specific codes is discussed in section 4.3. When the new component model is mapped with other component models, the coupling coefficient file needs to be provided, which is not automatically generated in the current version of CAS-CIG. We will implement automatic generation of the coupling coefficient file in the next release of CAS-CIG, as mentioned in section 3.6.

3.3. Merge Module

The role of the merge module is to merge data from the coupler. As seen from the data flow between component models through CPL7 shown in Figure 3, the data that need to be merged in different component
models are quite different. For example, the ICE model merges the data of the two component models of ATM and OCN, while the LND model only merges the data of the ATM model. In this case, the variables and process of the merge module are different from each component model, although the main functions of the merge module are the same: init, run, and final.

When implementing the automatic generation of the merge module, we adopted two solutions according to different situations. For an existing component model, a selection interface for each type of specific codes is provided, and the component model name to be merged is set in the configuration file of the merge module. For a new component model, the insertion interface of the specific codes in different functions and procedures is provided to support the free insertion of custom code. Figure 4 is an example of Atmosphere model in merge configuration file. The “atm” (L1 in Figure 4) means this is the configuration of the Atmosphere model. The “cother” (L3 in Figure 4) variable sets that model or submodel the atmosphere model needs to collect data from. The other variables (from L2 to L9 in Figure 4) correspond to the other code segments of the template, selected by the numbers. The numbers on the right hand (L2, L4, L6, L7, L8, and L9 in Figure 7) determine the ID (the unique flag), which corresponds to the ID in the template described in section 4.1. In the template example in Figure 5, the code segment affected by the “cother” (L1 and L8 in Figure 5) variable is found by the corresponding relationship of “key,” and the variables needed to collect the data are defined and initialized through the circular substitution of {cother} and {c} (from L2 to L6 and from L9 to L10 in Figure 5). When the user needs to insert specific codes, he can insert the corresponding code into the template as shown in Figure 5, which will then take effect when generated.

3.4. Share Module

The share module is a collective term for a series of functional modules in CPL7. It includes all the variables, common functions, and information required by the coupling process module. Table S1 in the supporting information shows the main module information of the share module. The time function, coupling process variables, index, and other functions involved in the share module play important roles in the coupling process control. Since common functions are basically common codes, you can use the template to generate the code directly. Data and information need to use the configuration file to set the coupling variables, indices, and other information of the component model. The configuration file is unified with the component interface module.

3.5. Component Interface Module

The component interface module is used for controlling the independent operation of a component model. Since each component model is a subsystem that can run independently, the actual calling process of each component model interface is different. The valuable common codes are the flow functions in the

| Table 1 | Variables Sending to Coupler of Atmosphere Model and Ice Model |
|---------|---------------------------------------------------------------|
| Component model | States | Fluxes |
| Atmosphere | z, u, v, tbot, ptem, shum, dens, pbot, psiv, and test | lwdn, rainc, rainl, snowc, snowl, swndr, swvdr, swdf, swdf, swnet, bchphdry, bchphwet, bchphdry, bchphwet, dswet1, dswet2, dswet3, dswet4, dsdry1, dsdry2, dsdry3, and dsdry4 |
| Ice | t, tref, qref, ifrac, avsdr, anidr, avsdf, anidf, snowh, and u10 | taux, tasy, lat, sen, lwp, evap, swnet, swpen, melth, and salt |
component interface module such as init, run, and final. We have made a template file based on these common functions. The template itself is simple. In addition, for different functions and locations, we also created some interfaces that can insert coupling variable definitions and specific codes.

3.6. Generation of Build Scripts and Build Directories

Currently, CAS-CIG automatically generates only the generic scripts needed to set up the compile directory. In fact, for each component model, the compilation of its own code also requires a compilation script. Due to the strong specificity of such scripts, CAS-CIG does not support automatic generation of component compilation scripts at present.

The compilation system in CAS-ESM consists of many script files. Since this paper focuses on the automatic generation of coupling interfaces, we will cover this briefly with an example.

As shown in Figure 6, when generating a compile directory, case name “casename” (default value: default), machine name “machine” and grid name “grids” (from L2 to L4 in Figure 6) need to be filled in the “case” field of the compile script configuration file, respectively. It is important to note that the grid names are entered in order according to the types of component models. The following “create_newcase” (L5 in Figure 6) field is used to generate the “create_newcase” script, which is automatically filled in when the intermediate value is generated. The example in Figure 6 will generate a compile directory named B1850C5X_test.

4. Automatic Generation Methodology of Coupling Interface

4.1. Methodology of Common Code Multiplexing

The concept of common code has been mentioned many times in the previous section. The common code is mostly the same or even exactly the same code segment that summarizes and refines all the component model codes and the coupler codes. This part of the code can be reused multiple times when generating the coupling code for the simulation case.

Figure 7 shows a standard example of the common code. Here, we will briefly explain the meaning of each mark in Figure 7. The exclamation mark before each line indicates that these are the template codes. The “key” (L1 in Figure 7) represents the type of common code. The “number” (L1 in Figure 7) determines the ID (the unique flag) of the current code segment in this type of common code. “Key” and “number” are mainly combined with the parsing result of the configuration file to identify and replace the code segment. Sometimes, these two fields are also needed when inserting a specific code. The code generated by this template is used to rearrange the variables when mapping between two component models is
needed. However, for different component models, the variables that need to be rearranged may be different, for example, the atmosphere model requires additional rearrangement of “ka” and “km” (L5 and L6 in Figure 7) variables. “<list>” (L1 in Figure 7) and “</list>” (L17 in Figure 7) represent the starting and ending ranges of the common code segment, respectively. “[c1],” “[ccc1],” “[c2],” and “[ccc2]” (from L2 to L16 and from L19 to L26 in Figure 7), which are called renamable words, are used for regular replacement, which we will detail in the subsection below.

What is illustrated on Figure 7 is a template of common code and that these templates are used to create coupler codes of component models. When creating these templates, the templates will be classified according to the modules described in the previous section. Furthermore, when extracting the common code segments, the starting and ending positions of the code segments should be divided according to functions and the segments should be kept related to the same function together as much as possible. As a result, there are fewer code segments separated in a template and fewer variables in the configuration file when each segment is represented by a “key.” For the template, only using the common code segment is insufficient. We will explain the insertion of the specific codes in section 4.3.

4.2. Regular Expression Replacement

As mentioned above, the start and end positions are marked with <list> and </list>, respectively, and the type and ID are marked with “key” and “number,” respectively. During the replacement process, we are ready to match the parsing result of the configuration file when the start mark is encountered. “Number” is an integer variable that can be directly matched. “key,” however, needs to be matched using a regular expression. We set the regular expression as “key\ = \([^\(\*\)]\)*”. After matching the key and number (or the type and the ID), the renamable words within the segment will be replaced by the corresponding component model names.

As shown in Figure 7, there are many types of renamable words. In fact, “[c1]” represents the short name of the first component model and “[c2]” represents the short name of the second component model. “[ccc1]” represents the long name of the first component model. Sometimes, the full name of a component model is needed. For example, the atmosphere model requires additional rearrangement of “ka” and “km” (L5 and L6 in Figure 7) variables. “<list>” (L1 in Figure 7) and “</list>” (L17 in Figure 7) represent the starting and ending ranges of the common code segment, respectively. “[c1],” “[ccc1],” “[c2],” and “[ccc2]” (from L2 to L16 and from L19 to L26 in Figure 7), which are called renamable words, are used for regular replacement, which we will detail in the subsection below.

What is illustrated on Figure 7 is a template of common code and that these templates are used to create coupler codes of component models. When creating these templates, the templates will be classified according to the modules described in the previous section. Furthermore, when extracting the common code segments, the starting and ending positions of the code segments should be divided according to functions and the segments should be kept related to the same function together as much as possible. As a result, there are fewer code segments separated in a template and fewer variables in the configuration file when each segment is represented by a “key.” For the template, only using the common code segment is insufficient. We will explain the insertion of the specific codes in section 4.3.

4.2. Regular Expression Replacement

As mentioned above, the start and end positions are marked with <list> and </list>, respectively, and the type and ID are marked with “key” and “number,” respectively. During the replacement process, we are ready to match the parsing result of the configuration file when the start mark is encountered. “Number” is an integer variable that can be directly matched. “key,” however, needs to be matched using a regular expression. We set the regular expression as “key\ = \([^\(\*\)]\)*”. After matching the key and number (or the type and the ID), the renamable words within the segment will be replaced by the corresponding component model names.

As shown in Figure 7, there are many types of renamable words. In fact, “[c1]” represents the short name of the first component model and “[c2]” represents the short name of the second component model. “[ccc1]” represents the long name of the first component model. Sometimes, the full name of a component model

| Template          | seqFlds_mod.template | seqFlds_indices.template | [ccc]_comp_mct.template | [ccc]_cpl_indices.F90 |
|-------------------|----------------------|-------------------------|------------------------|-----------------------|
| Illustration      | Registration of states and fluxes variables | Registration of index of variables | Numerical transfer from component variables to coupling variables | Index variables definition, index variables assignment and correlation |

Table 2

Coupling Variables Automatic Generation Templates

L1: [OCN]

......

L2: to_cpl_state=[t:integer, u:integer, v:integer, s:integer, dllx:integer, dldy:integer]

L3: to_cpl_fluxes=[ioo_q:integer, aoo_fco2_ocn:integer, aoo_fhans_ocn:integer]

L4: cpl_to_state=[i_fic:integer, o_dm10n:integer, a_pslv:integer, a_co2prog:integer, a_co2diag:integer]

L5: cpl_to_fluxes=[oxx_taux:integer, oxx_tny:integer, oxx_swmet:integer, oxx_sen:integer, oxx_lat:integer, oxx_lwvd:integer, oxx_lwvp:integer, oxx_melt:integer, oxx_salt:integer, oxx_prec:integer, oxx_snow:integer, oxx_rain:integer, oxx_evap:integer, oxx_rhmet:integer, orr_rof:integer, orr_riff:integer, orr_ifff:integer, oxx_bcpdhdy:integer, oxx_bcpdhy:integer, oxx_bcphwet:integer, oxx_ocphdy:integer, oxx_ocphwy:integer, oxx_ocphwt:integer, oxx_dstwt:integer]

......

Figure 11. Configuration file for coupling variables in ocean component.
is also used. We use the different marks to determine which form of name or type to use for replacement. Because the name and type of the different forms of the component model are used here, it is necessary to generate intermediate values. This process will be introduced in section 4.5.

Figure 8 is a python program that shows the process of replacing common code and generating coupling code. When both type and ID are matched by a component model (from L1 to L6 in Figure 8), the code segment from the start and end tags is added to the variable to be replaced. Replace the short and long names in the variable to be replaced row by row (from L11 to L15 in Figure 8). Finally, the replaced code segment is printed into the coupling code.

4.3. Specific Code Insertion

Specific codes are a class of code segment that varies considerably from one component model to another. The number of specific codes is small, but the insertion positions and code forms vary. The specific codes can be divided into two categories, which are coupling variable declarations and data processing.

As shown in Figure 3, there are two relationships between component models: direct coupling and indirect coupling. Therefore, the coupling variables received from the coupler and sending to coupler are different in different component models. Table 1 is a comparison of variables sending to coupler between atmosphere
model and ice model. Coupling variables can be divided into two categories, states and fluxes. State variables can be used but cannot be updated in the coupler, and flux variables are quantities that can be changed by calculation. As can be seen from Table 1, states and fluxes of different component models are quite different.

Figure 9 is an example of specific codes for data processing. This code is a piece of code in the ocean model coupling interface. As you can see in Figure 3, the ocean model requires data from the atmosphere model. In the process of receiving data transmitted by atmospheric model, since variables related to “CO2” are not necessary, ocean model needs to judge whether these variables need to be registered (from L1 to L8 in Figure 9). However, other component models do not require this code.

For automatic generation of coupling variables, cyclic substitution through templates is still possible. Figure 10 is a flow chart which shows generation for coupling variables when two component models are coupled. First, the coupling variables are declared in the coupling interfaces of Component Model 1 and Component Model 2. Second, the coupling variable is registered in the coupler, including what is the use of the coupling variable, whether the coupling variable is an input or output variable, and whether the coupling variable is a state or a flux. Finally, the coupling variables are initialized. Due to the complexity of generating a coupling variable, we list the template files associated with automatic generation of coupling variables in Table 2. When coupling variables are automatically generated, the four templates shown in Table 2 are used simultaneously for traversal substitution.

We use the configuration file shown in Figure 11 to configure the coupling variables. Figure 11 is an example of an ocean component configuration file. Where “to_cpl_state” (L2 in Figure 11) indicates that the listed variables are input to the coupler, while “cpl_to_fluxes” (L5 in Figure 11) indicates that the listed variables are output from the coupler, and so on. To add a coupling variable, simply write the name and type of the coupling variable in the configuration file, separated by a colon, such as “tinteger,” “ioo_q:integer,” and “oxx_taux:integer”. The variable type can be replaced with “character,” “real,” “logical,” and other variable types supported by FORTRAN90. Figure 12 is an example of a specific code template. Some component models require simple calculations and processing before variable values are transmitted to the coupler (from L3 to L17 in Figure 12). Coupling variables are primarily related to "<list>" section (L2 in Figure 12). Where this part of the code is highlighted in red (L10 and L11 in Figure 12) is where it needs to be replaced, and the rest can be generated directly by printing. When generating, the generate script replaces {c} with the short name of a component model. The generate script then replaces [to_cpl_states] with each state variable sending to coupler to generate a new line of code. Similarly it is done for each flux variable sending to the coupler in place of [to_cpl_fluxes]. In this way, we can declare, identify, index, register, initialize and pass all coupling variables.

For the automatic generation of the data processing part, we combine the insertion point with component model configuration file. In Figure 12 “"!Xmlinsert" (L1 and L19 in Figure 12) represents an insertion point. Information in parentheses such as “before_init,” “seq_infodata” are keys. We then add the specific codes using the data processing configuration file of each component model as shown in Figure 13. “[seq_infodata]” (L1 in Figure 13) is the key that writes the data processing code behind it. The generate script will determine whether the code needs to be inserted at each insertion point according to the data processing configuration file and then generate the complete component models interface code. For the cross-platform component model, procedures calling other submodules can also be written to the data processing configuration file. The user variable is an input or output variable, and whether the coupling variable is a state or a flux. Finally, the coupling variables are initialized. Due to the complexity of generating a coupling variable, we list the template files associated with automatic generation of coupling variables in Table 2. When coupling variables are automatically generated, the four templates shown in Table 2 are used simultaneously for traversal substitution.

We use the configuration file shown in Figure 11 to configure the coupling variables. Figure 11 is an example of an ocean component configuration file. Where “to_cpl_state” (L2 in Figure 11) indicates that the listed variables are input to the coupler, while “cpl_to_fluxes” (L5 in Figure 11) indicates that the listed variables are output from the coupler, and so on. To add a coupling variable, simply write the name and type of the coupling variable in the configuration file, separated by a colon, such as “tinteger,” “ioo_q:integer,” and “oxx_taux:integer”. The variable type can be replaced with “character,” “real,” “logical,” and other variable types supported by FORTRAN90. Figure 12 is an example of a specific code template. Some component models require simple calculations and processing before variable values are transmitted to the coupler (from L3 to L17 in Figure 12). Coupling variables are primarily related to "<list>" section (L2 in Figure 12). Where this part of the code is highlighted in red (L10 and L11 in Figure 12) is where it needs to be replaced, and the rest can be generated directly by printing. When generating, the generate script replaces {c} with the short name of a component model. The generate script then replaces [to_cpl_states] with each state variable sending to coupler to generate a new line of code. Similarly it is done for each flux variable sending to the coupler in place of [to_cpl_fluxes]. In this way, we can declare, identify, index, register, initialize and pass all coupling variables.

For the automatic generation of the data processing part, we combine the insertion point with component model configuration file. In Figure 12 “"!Xmlinsert" (L1 and L19 in Figure 12) represents an insertion point. Information in parentheses such as “before_init,” “seq_infodata” are keys. We then add the specific codes using the data processing configuration file of each component model as shown in Figure 13. “[seq_infodata]” (L1 in Figure 13) is the key that writes the data processing code behind it. The generate script will determine whether the code needs to be inserted at each insertion point according to the data processing configuration file and then generate the complete component models interface code. For the cross-platform component model, procedures calling other submodules can also be written to the data processing configuration file. The user variable is an input or output variable, and whether the coupling variable is a state or a flux. Finally, the coupling variables are initialized. Due to the complexity of generating a coupling variable, we list the template files associated with automatic generation of coupling variables in Table 2. When coupling variables are automatically generated, the four templates shown in Table 2 are used simultaneously for traversal substitution.

We use the configuration file shown in Figure 11 to configure the coupling variables. Figure 11 is an example of an ocean component configuration file. Where “to_cpl_state” (L2 in Figure 11) indicates that the listed variables are input to the coupler, while “cpl_to_fluxes” (L5 in Figure 11) indicates that the listed variables are output from the coupler, and so on. To add a coupling variable, simply write the name and type of the coupling variable in the configuration file, separated by a colon, such as “tinteger,” “ioo_q:integer,” and “oxx_taux:integer”. The variable type can be replaced with “character,” “real,” “logical,” and other variable types supported by FORTRAN90. Figure 12 is an example of a specific code template. Some component models require simple calculations and processing before variable values are transmitted to the coupler (from L3 to L17 in Figure 12). Coupling variables are primarily related to "<list>" section (L2 in Figure 12). Where this part of the code is highlighted in red (L10 and L11 in Figure 12) is where it needs to be replaced, and the rest can be generated directly by printing. When generating, the generate script replaces {c} with the short name of a component model. The generate script then replaces [to_cpl_states] with each state variable sending to coupler to generate a new line of code. Similarly it is done for each flux variable sending to the coupler in place of [to_cpl_fluxes]. In this way, we can declare, identify, index, register, initialize and pass all coupling variables.

For the automatic generation of the data processing part, we combine the insertion point with component model configuration file. In Figure 12 “"!Xmlinsert" (L1 and L19 in Figure 12) represents an insertion point. Information in parentheses such as “before_init,” “seq_infodata” are keys. We then add the specific codes using the data processing configuration file of each component model as shown in Figure 13. “[seq_infodata]” (L1 in Figure 13) is the key that writes the data processing code behind it. The generate script will determine whether the code needs to be inserted at each insertion point according to the data processing configuration file and then generate the complete component models interface code. For the cross-platform component model, procedures calling other submodules can also be written to the data processing configuration file. The user
Figure 15. An example of the process configuration file.

```plaintext
L1: [proc]
L2: seq = [atm, lnd, ocn, ice, glc]
L3: [atm]
L4: components=[1]
L5: merge=[1]
L6: mapping=['lnd', 'ice', 'ocn', 'atm']
L7: gsMap_components=[1]
......
L8: [lnd]
L9: components=[2]
L10: merge=[1]
L11: mapping=['lnd']
L12: gsMap_components=[1]
......
```

4.4. Configuration File Format

The configuration files are an important part of the CAS-CIG. In many couplers, the xml file format is often used as the configuration file format, including the C-Coupler, OASIS, and BFG. The advantage of XML files is that the format is neat and easily readable, and many programming languages can parse XML files. However, the CAS-CIG is designed and written based on the Python language. Multiple data structures are defined using Python. Lists and dictionaries are common data structures in Python, and their forms are simple and clean. In addition, there are many unnecessary characters in the xml format file. In terms of parsing, Python’s configparser module provides powerful configuration file parsing. Therefore, we use a list and dictionary as the format of the configuration file in the CAS-CIG. Below we will illustrate various configuration file formats combined with different modules and functions.

The common configuration file. Figure 14 is an example of the common configuration file. The user needs to set the component model type and the name of the simulation case in the common configuration file. The type and name here are as described in section 2.2. As shown in Figure 14, the configuration file can have a “[common]” (L1 in Figure 14) section and an “[other]” (L7 in Figure 14) section. Then, the user sets the existing component model type and name after the “[common]” tag and sets the new component model type and name after the “[other]” tag. “atm,” “lnd,” “ice,” “ocn,” “glc” (From L2 to L6 in Figure 14), “chem,” and “wrf” (from L8 to L9 in Figure 14) are the types of the component models. “cam,” “clm,” “cice,” “pop,” “sglc,” “gea,” and “wrf” are the names of the component models in this example. We can also use “iap-agcm” for the “atm” type, “colcma” for the “lnd” type, and so on.

The process configuration file. Figure 15 is an example of the process configuration file. The “[proc],” “[atm],” “[lnd]” (L1, L3 and L8 in Figure 15) are the section labels. The “Seq,” “components,” “merge,” and so on are the keys corresponding to the template. The key “seq” in Figure 15 shows the coupling order of component models, which generates the driver code for each component model in turn. The “[proc]” label is followed by the labels of component models. The key “mapping” (L6 in Figure 15) is special, which is used to identify which component models the current field needs to map with. Other keys can be selected using numbers. These numbers also correspond to templates:

The module configuration file. Module configuration files include mapping module configuration files, merging module configuration files, sharing module configuration files, and sequence module configuration files. Below we discuss the module configuration file with the mapping configuration file as an example. See supporting information Figures S1 to S4 for other module configuration files.

Figure 16 is an example of the mapping module configuration file. The section label “[atmlnd]” (L1 in Figure 16) represents the mapping relationship of “ATM” to “LND.” Because the code segments of that mapping module of the coupler code can be distinguished by their IDs, the “key” and “number” are used in the configuration file. The names of the “key” labels are “use,” “public_interface,” “private_data,” “init1,” etc. (From L2 to L15 in Figure 16). In addition, the “number” values are “1,” “2,” “3,” etc. The “key” and “number” in the configuration file correspond to those in the mapping template. If you do not need a piece of code, you can set it to empty without writing the corresponding “key” tag.

Since there are many component pattern mapping relationships in the simulation cases, it is troublesome to modify the configuration file line by line. Therefore, after configuring the common configuration file, executing a python script called “create_defaults.py,” the CAS-CIG will automatically generate the intermediate value, thereby giving the default value of the mapping configuration file. These intermediate values are the default values for some configuration files and information such as the component model name.
needed to generate the coupling code. The generation of the intermediate value will be explained in the next subsection.

**The component configuration file.** Figure 11 is an example of the component configuration file, which was covered earlier and will not be repeated here.

### 4.5. Methodology of Intermediate Value Generation

Intermediate value generation technology is used to simplify configuration files. As mentioned earlier, there are many default values in the configuration file. In the replacement of the template file, information such as the short name, long name, and full name of the component model is also used. Therefore, it is necessary to generate intermediate values to avoid multiple iterations of such information.

The information needed to generate the intermediate values is in the common configuration file. After modifying the common configuration file, the CAS-CIG can automatically generate intermediate values, including prewriting the configuration files and the generation of other information.

Figure 17 shows an example of the intermediate values generated through the common configuration file. “c1” (L1 in Figure 17) can be thought of as the short names for component models. “ccc2” and “ccc3” (L2 and L3 in Figure 17) are the long names for component models for substitution. The main difference is that long names in ccc3 require pairwise combinations, such as “wrf,” “cam” being one group and “atm,” “lnnd” being another.

### 5. Validation and Performance of CAS-CIG

To evaluate the CAS-CIG, we used it to construct several simulation cases and tested them on a common performance computing platform. The main purpose of these experiments is to verify the correctness of the CAS-CIG generated codes. Indeed, the CAS-CIG is a high-performance computing application that requires performance testing.

All the experiments conducted in this paper are carried out on the high-performance computer “Yuan” of the Chinese Academy of Sciences. The computing node processor version of the “Yuan” Supercomputer Phase II is an Intel(R) CPU E5-2680 V3: 2.5GHz. The memory is 128 GB DDR3 ECC,1866 MHz on each node. The system version is the Linux CentOS release 6.4 (Final), and the compilation environment is the Intel composer_xe_2013_sp1.0.080 and Intelmpi 4.1.3.049.

Here we present one example of a preindustrial coupled model simulation in Figure 18 (a case of B1850, which is a case with all active component models and data of Year 1850). The component models of this case include IAP AGCM, CoLM, LICOM, and CICE. The horizontal resolution of IAP AGCM and CoLM model is 1.4° in both latitude and longitude, and the horizontal resolution of LICOM and CICE is 1°. We conducted simulation tests on the CAS-ESM using CAS-CIG and CAS-ESM using CPL7, respectively. The number of processors used here is 128 cores. Figure 18a shows the global distribution of surface air temperature by CAS-ESM with CAS-CIG (top panel). (b) The difference of simulated temperature of CAS-ESM using CAS-CIG and CPL7 (bottom panel).
Figure 18. Global distribution of simulated surface air temperature by CAS-ESM with CAS-CIG (top panel), and the difference of simulated temperature of CAS-ESM using CAS-CIG and CPL7 (bottom panel).

Figure 19. Simulation time in CAS-CIG.
simulations by using the two versions. Figure 18 is generated by AMWG (Atmosphere Model Working Group) Diagnostics Package and NCL (NCAR Command Language), which read the results of the earth surface air temperature data using two simulation tests. We used `cprnc`, which is a tool shared by both CAM and CLM to compare two NetCDF history files, to compare the results of the two simulation experiments and found that the results were bit-for-bit consistent. We also compared other output fields and got the same conclusion.

We also conducted performance tests of the B1850 simulation by using different number of processors. Figure 19 gives wall-clock time per model day in simulations with different processors in CAS-CIG. As we expected, the computation time scales almost linearly with the number of processors increased from 16 to 256. However, when the number of processors continues to increase from 256 to 512, the total running time does not change significantly. Figure 20 is the figure of speedup in CAS-CIG. It is easier to see the trend of performance changes in Figure 20. Table 3 is the comparison of the running time of CAS-CIG and CAS-ESM. The percentage difference in Table 3, calculated by (CIG-ESM)/ESM, shows that the simulation time using CAS-CIG and CAS-ESM has little difference. When using different processors, there will always be some difference in simulation cost, which is normal. These parallel characteristics are very similar to that of CAS-ESM with CPL7, indicating that the code generated by CAS-CIG does not affect the computational efficiency.

In addition to the B1850 experiment described above, we also designed and conducted the following other experiment, such as “IAP AGCM+CLM+LICOM+CICE,”, “IAP AGCM+CLM+POP+CICE,” and “CAM +CLM+POP+CICE.” The simulation results are also consistent with those based on CAS-ESM with CPL7. Figures S5 to S7 of the supplement show the simulation results of these three experiments.

### 6. Summary and Discussion

The CAS-CIG is a coupled interface generator designed and developed based on the CPL7 of CESM1 to add new component models. CAS-CIG automatically generates the coupler interface code required by the simulation case, thereby simplifying the tedious work of adding component models. We have described the design, key modules, and the methodologies of automatic generation of coupled interfaces.

Future work to improve CAS-CIG includes the following.

- **Integration tests of other component models.** Currently, completed experiments (as shown in section 3.5) have tested the validity of the CAS-CIG with the existing component models of the CAS-ESM. Other problems may arise when performing coupling interface generation tests on the component models of other Earth System Models.

- **Automatic generation of coupling coefficients.** The CAS-CIG currently focuses on the automatic generation of coupling interfaces. When the simulation case experiment is performed, interpolation coefficients are required for mapping and coupling with different model resolutions. Therefore, the automatic generation of the coupling coefficient is also an important functional component of the CAS-CIG.

- **Three-dimensional coupling.** The CAS-CIG is a coupled interface generator based on CPL7. Since CPL7 only supports 2-D coupling, the CAS-CIG does not have 3-D coupling function. The 3-D coupling in CPL7 is implemented by invoking 2-D coupling. However, with the development of component models and the improved accuracy, 3-D coupling is a function that is being used in CAS-ESM. In the process of integrating the regional model WRF, CAS-ESM has made some

| Number of processes | CAS-CIG  | CAS-ESM | Percentage difference |
|---------------------|----------|---------|-----------------------|
| 16                  | 1,1471.59| 1,2564.76| −8.7%                |
| 32                  | 7,046.80 | 6,471.21| 8.9%                 |
| 64                  | 3,503.09 | 3,735.31| −6.2%                |
| 128                 | 2,075.16 | 2,155.94| −3.8%                |
| 256                 | 1,261.51 | 1,191.85| 5.8%                 |
| 512                 | 866.45   | 830.82  | 4.3%                 |
improvements to CPL7 to support 3-D coupling. Therefore, the CAS-CIG should also support 3-D coupling access and automatic generation.

More performance tests. In section 3.5, we tested the detailed performance of the BI850 experiments using CPL7 and CAS-CIG. It is only tested for the consistency of results in other experiments. CAS-CIG can serve as a software platform for the future research of Earth System Model, so the detailed performance of other experiments needs to be tested in the follow-up work.

It is hoped that CAS-CIG can be a community software tool for other modeling groups to substitute model components, who use CESM with CPL7 to explore their impact and related science questions on modeling weather and climate. Since CAS-CIG can generate coupling interface automatically, the cross-platform component models can also be easily accessible. We have made the CAS-CIG freely available for others to download and use and welcome improvement to it from other contributors.

Data Availability Statement

The CAS-CIG software is available at Wang Tianyi (24 January 2020) (CAS-CIG. Zenodo, http://doi.org/10.5281/zenodo.3626263)

References

Balaji, V., Anderson, J., Held, I., Winton, M., Durachta, J., Malyshnev, S., & Stouffer, R. J. (2006). The exchange grid. Parallel Computational Fluid Dynamics, 2005, 179–186. https://doi.org/10.1016/b978-044452206-1.50021-5

Collins, W. D., Blitz, C. M., Blackmon, M. L., Bonan, G. B., Bretherton, C. S., Carton, J. A., et al. (2006). The Community Climate System Model Version 3 (CCSM3). Journal of Climate, 19(11), 2122–2143. https://doi.org/10.1175/jcli3761.1

Craig, A. P., Jacob, R., Kauffman, B., Bette, T., Larson, J., Ong, E., et al. (2005). CPL6: The new extensible, high performance parallel coupler for the community climate system model. The International Journal of High Performance Computing Applications, 19(3), 309–327. https://doi.org/10.1177/1094342005056117

Craig, A., Valcke, S., & Coquart, L. (2017). Development and performance of a new version of the OASIS coupler, OASIS3-MCT 3.0. Geoscientific Model Development, 10, 3297–3308. 2017. https://doi.org/10.5194/gmd-10-3297-2017

Craig, A., Vertenstein, M., & Jacob, R. (2012). A new flexible coupler for earth system modeling developed for CCSM4 and CESM1. The International Journal of High Performance Computing Applications, 26(1), 31–42. https://doi.org/10.1177/1094342011428141

Dai, Y., Zeng, X., Dickinson, R. E., Baker, I., Bonan, G. B., Bosilovich, M. G., et al. (2003). The common land model. Bulletin of the American Meteorological Society, 84(8), 1013–1024. https://doi.org/10.1175/bams-84-8-1013

Gent, P. R., Danabasoglu, G., Donner, L. J., Holland, M. M., Hunke, E. C., Jayne, S. R., et al. (2011). The Community Climate System Model Version 4. Journal of Climate, 24(19), 4973–4991. https://doi.org/10.1175/2011jcli4083.1

Hill, C., DeLuca, C., Balaji Suarez, M., & Da Silva, A. (2004). The architecture of the earth system modeling framework. Computing in Science & Engineering, 6(1), 18–28. https://doi.org/10.1109/mcite.2004.1255817

Hines, K. M., & Bronwich, D. H. (2008). Development and testing of polar weather research and forecasting (WRF) model. Part I: Greenland Ice Sheet Meteorology. Monthly Weather Review, 136(6), 1971–1989. https://doi.org/10.1175/2007mwr2112.1

Huang, W.-Y., Wang, B., Li, L.-L., & Yu, Y.-Q. (2014a). Improvements in LICOM2. Part II: Arctic circulation. Journal of Atmospheric and Oceanic Technology, 31(1), 233–245. https://doi.org/10.1175/jtech-d-13-00064.1

Huang, W.-Y., Wang, B., Yu, Y.-Q., & Li, L.-J. (2014b). Improvements in LICOM2. Part I: Vertical mixing. Journal of Atmospheric and Oceanic Technology, 31(2), 531–544. https://doi.org/10.1175/jtech-d-13-00065.1

Hunke, E. C., Hebert, D. A., & Lecomte, O. (2013). Level-ice melt ponds in the Los Alamos sea ice model, CICE. Ocean Modelling, 71, 26–24. https://doi.org/10.1016/j.ocemod.2012.11.008

Hurrell, J. W., Holland, M. M., Gent, P. R., Ghan, S., Kay, J. E., Kushner, P. J., et al. (2013). The community Earth system model: A framework for collaborative research. Bulletin of the American Meteorological Society, 94(9), 1339–1360. https://doi.org/10.1175/bams-d-12-00121.1

Jacob, R., Larson, J., & Ong, E. (2005). M X N communication and parallel interpolation in Community Climate System Model Version 3 using the model coupling toolkit. The International Journal of High Performance Computing Applications, 19(3), 293–307. https://doi.org/10.1177/1094342005056116

Kebbyson, D. J., & Jones, P. W. (2005). A performance model of the parallel ocean program. The International Journal of High Performance Computing Applications, 19(3), 261–276. https://doi.org/10.1177/1094342005056114

Larson, J., Jacob, R., & Ong, E. (2005). The model coupling toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models. The International Journal of High Performance Computing Applications, 19(3), 277–292. https://doi.org/10.1177/1094342005056115

Lawrence, P. J., & Chase, T. N. (2007). Representing a new MODIS consistent land surface in the Community Land Model (CLM 3.0). Journal of Geophysical Research, 112(G1). https://doi.org/10.1029/2006jg000168

Leasure, B., Kuck, D. I., Gorlatch, S., Cole, M., Watson, G. R., Darte, A., et al. (2011). Parallel Ocean Program (POP). Encyclopedia of Parallel Computing, 1416–1416. https://doi.org/10.1007/978-0-387-97664-4_2394

Liu, H., Lin, P., Yu, Y., Wang, F., Liu, X., & Zhang, X. (2013). LASIC/IAP Climate System Ocean Model Version 2: LICOM2. In T. Zhou, Y. Yu, Y. Liu, & B. Wang (Eds.), Springer Earth System Sciences Flexible Global Ocean-Atmosphere-Land System Model (pp. 15–26). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-41801-3_3

Liu, L., Yang, G., Wang, B., Zhang, C., Li, R., Zhang, Z., et al. (2014). C-Coupler1: A Chinese community coupler for Earth system modeling. Geoscientific Model Development, 7(5), 2281–2302. https://doi.org/10.5194/gmd-7-2281-2014
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. *Geoscientific Model Development, 11*(9), 3557–3586. https://doi.org/10.5194/gmd-11-3557-2018

Meehl, G. A., Washington, W. M., Arblaster, J. M., Hu, A., Teng, H., Kay, J. E., et al. (2013). Climate change projections in CESM1(CAM5) compared to CCSM4. *Journal of Climate, 26*(17), 6287–6308. https://doi.org/10.1175/jcli-d-12-00572.1

Meng, C. L., Li, Z.-L., Zhan, X., Shi, J. C., & Liu, C. Y. (2009). Land surface temperature data assimilation and its impact on evapotranspiration estimates from the common land model. *Water Resources Research, 45*, W02421. https://doi.org/10.1029/2008wr006971

Redler, R., Valcke, S., & Ritzdorf, H. (2010). OASIS4—A coupling software for next generation Earth system modelling. *Geoscientific Model Development, 3*(1), 87–104. https://doi.org/10.5194/gmd-3-87-2010

Smith, R., Jones, P., Briegleb, B., Bryan, F., Danabaoglu, G., Dennis, J., et al. (2010). The Parallel Ocean Program (POP) reference manual ocean component of the Community Climate System Model (CCSM) and Community Earth System Model (CESM). Rep. LAUR-01853, 141, 1–140.

Tallapragada, V. (2018, April). Development and implementation strategies for NOAA’s Unified Forecast System for operational NWP at NCEP. In EGU general assembly conference abstracts (Vol. 20, p. 10325).

Theurich, G., DeLuca, C., Campbell, T., Liu, F., Saint, K., Vertenstein, M., et al. (2016). The Earth system prediction suite: Toward a coordinated US modeling capability. *Bulletin of the American Meteorological Society, 97*(7), 1229–1247. https://doi.org/10.1175/BAMS-D-14-00164.1

Urrego-Blanco, J. R., Urban, N. M., Hunke, E. C., Turner, A. K., & Jeffery, N. (2016). Uncertainty quantification and global sensitivity analysis of the Los Alamos sea ice model. *Journal of Geophysical Research: Oceans, 121*, 2709–2732. https://doi.org/10.1002/2015jc011558

Valcke, S. (2013). The OASIS3 coupler: A European climate modelling community software. *Geoscientific Model Development, 6*(2), 373–388. https://doi.org/10.5194/gmd-6-373-2013

Zhang, H., Lin, Z. H., & Zeng, Q. C. (2009). The computational scheme and the test for dynamical framework of IAP AGCM-4. *Chinese Journal Of Atmospheric Sciences-Chinese Edition, 33*, 1267–1285. https://doi.org/10.3878/j.issn.1006-9895

Zhang, H., Zhang, M., & Zeng, Q. (2013). Sensitivity of simulated climate to two atmospheric models: Interpretation of differences between dry models and moist models. *Monthly Weather Review, 141*(5), 1558–1576. https://doi.org/10.1175/mwr-d-11-00367.1