We thank all reviewers a lot for the comments and suggestions.

Reply to Referee1

1. Grammar and syntax needs to be considered much more carefully.

Response: We carefully improved the grammar and syntax in the revised version.

2. The software does not contain any version number.

Response: The version number has been added into the software.

3. In the introduction the authors raise the impression that they address high-resolution climate model applications running on modern high-performance compute systems where a single model employs several thousand processors or cores. Later on the algorithmic approach is investigated with test cases at very coarse resolution o(2 degrees) on a comparatively low number of cores (192) and leave the reader alone with any guess about the scalability of their approach.

Response: The performance of data transfer between high-resolution toy models has been evaluated, where each model employs about one thousand processor cores (please refer to P12 L29 – P13 L9 and Fig. 19).

4. P8983,L1: Is it the number of coupled models or the number of coupled model configurations the authors have in mind?

Response: It means the number of coupled model configurations (please refer to P2 L10 - L11).

5. P8984,L27: Do you believe or are you convinced?

Response: The sentence is modified as "We believe that other coupler versions can also benefit from it" (please refer to P3 L28 – L29).

6. The main - if not the only - purpose of section 2 is to provide the reader with an overview about the communication algorithms which are used in existing coupling software. In essence this section is telling us that all existing coupling software products use P2P communication. I wonder why I have to read approx. 85 lines to arrive at this. An overview of existing coupler software has already been published elsewhere – among the GMD - and the author should be able to reference those rather than providing another overview.

Response: The overview about the data transfer in existing coupler is shrunk (please refer to P4 L4 – L16). We have merged it and the original Section 3 into Section 2 of the revised version (please refer to P4 L3 – P5 L30).

7. In section 4 the headline raises the expectation that we can learn how the butterfly algorithm works. The reader is not really guided through this section. Is the numbered list in sec 4.1 based on findings

by the authors? In this case some piece of information is missing which guides the reader to this statement. In case it is not based on the authors findings a reference is missing. Fig. 6 (and likewise Fig 8) does not help me at all to learn how the butterfly algorithm works. If each of the 8 processes P0 to P8 already has all data D0 to D8 I cannot see any necessity for communication. What is the information that shall be transported to the reader with the colours?

Response: The butterfly algorithm is more clearly explained in Section 3 (please refer to P6 L1 – P9 L12). An example is given to explain the numbered list in Section 3.1 (please refer to P7 L21 – L30, Fig. 9, Fig. 10, Fig. 11). More information is added to help the understanding of the butterfly implementation (please refer to P7 L2 – L20, Fig. 7, Fig. 13).

8. I would have loved to be guided through Fig 7 in the text a little bit. If this Figure is not important at all it should be removed.

Response: Please refer to P8 L30 – P9 L12 and Fig. 12.

9. In section 5 it remains unclear (to me at least) how the adaptive process works and I would appreciate if this was clarified in a revised version. Does this work as a kind of self-learning algorithm where the optimal path is determined of the first n data exchanges of a model integration or is this part of the initialisation procedure beforehand and made available already for the first data exchange?

Response: Please refer to P10 L22 – L25.

10. The first sentence of section 6 does not make sense to me. Having read the previous sections the authors put the focus of the reader to the adaptive transfer library. Now the authors propose the butterfly implementation as well. Later we learn that the butterfly approach can be outperformed. At the end of the section the authors show that for coupled climate models the P2P communication is as good as the adaptive transfer library, probably because the adaptive transfer library completely switches to P2P in the latter case. I think that this is an important finding and should be emphasized. It tells us that the P2P which is used in existing coupler software is not that bad. But is also tell me that the paper is severely suffering from a clear structure. If my conclusion (P2P is sufficient) is wrong the authors will need to put more effort in getting the reader onto the correct track.

Response: The first sentence of the original Section 6 has been removed in the revised version. The manuscript is restructure and the finding is emphasized (please refer to P9 L30 – L31, P12 L13 – L14, P13 L19 – L20, P14 L14 - L15).

11. Table 1 and Fig 10 are not really addressed. Are they required to understand the adaptive data transfer library? These can be removed of shifted to the user guide.

Response: They are removed in the revised version and added to the user guide.

12. Could Fig 9 be replaced by a real flow chart rather than providing pseudo code?

Response: Please refer to Fig. 14 and P10 L3 – L25.

13. In section 6 the performance of the data transfer is evaluated by using a coupled climate model with roughly 2 degree grid horizontal grid spacing using 192 processes. As there are 8400 cores available Tansuo100 I would have expected to see an evaluation of the performance at least with a toy model and exploring the scalability of the adaptive data transfer library up to several thousand cores. Unless there are sound arguments why this cannot be done this raises the impression that the authors are trying to hide something. The dynamical core sets an upper limit to the number of cores that can reasonably be employed - when the communication starts dominating over the computing part (MPI messages required for the boundary exchange required for advection and diffusion operators versus the time for the forward integration of the less and less points left on a single core). With roughly 2 degree resolution we have probably reached this point with 192 processes. Here it would be nice to know how much percentage of the overall compute time is consumed by the data exchange, and how much wall clock time can be gained for a single run of the coupled model. Last but not least, how important is the load imbalance between the processes as the boundary exchange between the model components (atmosphere and ocean) provides a synchronisation point, either explicitly or implicitly, where the components have to wait for each others.

Response: The performance of data transfer between high-resolution toy models has been evaluated, where each model employs about one thousand processor cores (please refer to P12 L29 – P13 L9 and Fig. 19). As shown in Fig. 23, we use GAMIL2-CLM3 to measure the performance improvement resulted from the adaptive data transfer library for one realistic coupled model (Please refer to P14 L17 – L28 and Fig. 23). In the evaluation, the maximum core number of each component model is 128, because two component models will not achieved better performance when using more 128 CPU cores.

14. The conclusions are weak if not misleading. Fig. 17 does not really confirm the last statement, that "the adaptive transfer library can effectively improve the performance of data transfer in model coupling. What can we conclude or expect for model with higher resolution than those investigated in this study?

Response: The performance of data transfer between high-resolution toy models has been evaluated, where each model employs about one thousand processor cores (please refer to P12 L29 – P13 L9 and Fig. 19).

Reply to Referee2

1. They show a good understanding of the current state-of-the-art in climate models but are missing some of the history of butterfly networks in parallel computer design.

Response: Some related works about the butterfly networks and algorithms are introduced in section 3 of the revised version (P6 L6 – L13).

2. In their performance testing, the results can also be affected by the decomposition strategy (decomposing the domain by lat-lon blocks or by latitude stripes). It's not clear if the two land and atmosphere domains have different decomposition strategies which would impact performance. Please clarify.

Response: Parallel decompositions of component models can affect the performance of data transfer. For example, GAMIL and CLM3 has different parallel decompositions, so data transfer between them has big communication depth, and the adaptive data transfer library can significantly improve the performance of data transfer (please refer to P13 L25 – P14 L3); For the data rearrangement in parallel interpolation, the source parallel decomposition is similar to the target parallel decomposition, so the communication depth is small and the performance of data transfer will not be improved because the adaptive data transfer library will switch to the P2P implementation in this case (please refer to P14 L4 – L16). As component models have different computation characteristics, their parallel decompositions are usually different.

3. Overall this algorithm appears to be most useful on medium-sized grids and modest processor counts. That's ok but these limitations should be mentioned or data for larger cases presented.

Response: The performance of data transfer between high-resolution toy models has been evaluated, where each model employs about one thousand processor cores (please refer to P12 L29 – P13 L9 and Fig. 19).

Specific Comments

4. The decrease in time at the end of the graph in Figure 1 should be remarked upon. Will it continue to go down?

Response: Figure 1 is measured from the benchmark derived from GAMIL2-CLM3. The component models GAMIL2 and CLM3 can only scale to 128 processor cores, so we did not measure the time for more cores.

5. It's not clear what generated the data in Figure 2. Is that a P2P test program from an MPI distribution? And was it on the same machine?

Response: Please refer to Fig. 2.

6. The initialization overhead for the adaptive library could become to expensive at 1K and larger processor counts even if its only run once. It might be better to run it offline and read in the results when the climate model starts. Again a large case would help.

Response: Thanks a lot for this suggestion. It will be our future work. Please refer to P15 L13 – L16.

7. For Figure 15, are the "P2P" results from the unaltered CPL7 coupler or from the P2P option in their library? Please clarify.

Response: The P2P results are measured from the adaptive data transfer library which switches to the P2P implementation. Please refer to Fig. 20.

8. Technical Corrections: "network contention" is the preferred phrase instead of "jam of network communication" or "jams in communication".

Response: "jam of network communication" and "jams in communication" has been replaced with "network contention" in the revised version (P1 L20, P5 L28, P6 L4, and P7 L30).

9. There is more odd English phrasing throughout.

Response: We carefully improved the grammar and syntax in the revised version.

Improving Data Transfer for Model Coupling

C. Zhang[2,1], L. Liu[1,3], G. Yang[2,1,3], R. Li[2,1], and B. Wang[1,3,4]

[1]{Ministry of Education Key Laboratory for Earth System Modeling, Center for Earth System Science (CESS), Tsinghua University, Beijing, China}

[2]{Department of Computer Science and Technology, Tsinghua University, Beijing, China}

[3]{Joint Center for Global Change Studies (JCGCS), Beijing, China}

[4]{State Key Laboratory of Numerical Modelling for Atmospheric Sciences and Geophysical Fluid Dynamics (LASG), Institute of Atmospheric Physics, Chinese Academy of Sciences, Beijing, China.}

Correspondence to: L. Liu (liuli-cess@tsinghua.edu.cn), G. Yang (ygw@tsinghua.edu.cn)

Abstract

Data transfer~~, which~~ means transferring data fields between two component models or rearranging data fields among processes of the same component model~~,~~. It is a fundamental and most frequently used operation of a coupler. Most versions of state-of-the-art ~~coupler versions~~couplers currently use an implementation based on the point-to-point (P2P) communication of the Message Passing Interface (MPI) (~~call~~refer such an implementation as "P2P implementation" for short). In this paper, we reveal the drawbacks of the P2P implementation, including low communication bandwidth due to small message size, variable and big number of MPI messages, ~~and jams during communication.~~as well as network contention. To overcome these drawbacks, we propose a butterfly implementation for data transfer. Although the butterfly implementation can outperform the P2P implementation in many cases, it degrades the performance in some cases because the total message size transferred by the butterfly implementation is larger than ~~that~~the total message size transferred by the P2P implementation. To ~~make the~~further improve data transfer~~ completely improved~~, we design and implement an adaptive data transfer library that combines the advantages of both butterfly implementation and P2P implementation. Performance evaluation shows that the adaptive data transfer library significantly improves the performance of data transfer in most cases, and does not decrease the performance in any cases. Now, the adaptive data transfer

library is open to the public and has been imported into a coupler version C-Coupler1 for performance improvement of data transfer. We believe that ~~it can also improve~~ other coupler versions can also benefit from it.

1 Introduction

Climate System Models (CSMs) and Earth System Models (ESMs) are fundamental tools for simulating, predicting and projecting ~~the~~ climate. A CSM or an ESM generally integrates several component models, such as an atmosphere model, a land surface model, an ocean model~~,~~ and a sea-ice model, into a coupled system~~,~~ to simulate the ~~behaviors~~behaviours of ~~and~~of the climate system, including the interactions between components of the climate system. More and more ~~ESMs~~coupled models have sprung up in the world. For example, the number of coupled model ~~versions~~configurations in the Coupled Model Intercomparison Project (CMIP) has increased from less than 30 (used for CMIP3) to more than 50 (used for CMIP5).

High-performance computing is~~ an~~ essential technical support for model development, especially for higher and higher resolutions of models. Modern high-performance computers integrate an increasing number of processor cores for higher and higher computation performance. Therefore, efficient parallelization, which enables a model to utilize more processor cores for acceleration, becomes a technical focus in model development~~,~~; and a number of component models with efficient parallelization have sprung up. For example, the Community Ice CodE (CICE; Hunke et al., 2008, 2013) at 0.1° horizontal resolution can scale to 30,000 processor cores on the IBM Blue Gene/L (Dennis et al., 2008); the Parallel Ocean Program (POP; Kerbyson, 2005; Smith et al., 2010) at 0.1° horizontal resolution can also scale to 30,000 processor cores on the IBM Blue Gene/L and ~~to~~ 10,000 processor cores on a Cray XT3 (Dennis, 2007); the Community Atmosphere Model (CAM; Morrison et al., 2008; Neale et al., 2010, 2012) with ~~the~~a spectral element dynamical core (CAM-SE) at 0.25° horizontal resolution can scale to 86,000 processor cores on a Cray XT5 (Dennis et al., 2012). ~~To achieve an efficient parallelization of a coupled model, each component model requires to be efficiently parallelized.~~

A coupler is an important component in a coupled system. It links component models together to construct a coupled model, and controls the integration of the whole coupled model~~.~~ (Valcke, 2012). A number of couplers now are available~~ for model coupling~~, e.g., the Model Coupling Toolkit (MCT; Jacob et al., ~~2015~~2005), the Ocean Atmosphere Sea Ice Soil coupling software

(OASIS) coupler (Redler et al., 2010; Valcke, 2013), the Earth System Modelling Framework (ESMF; Hill et al., 2004), the CPL6 coupler (Craig et al., 2005), the CPL7 coupler (Craig et al., 2012), the Flexible Modelling System (FMS) coupler (Balaji et al., 2006), the Bespoke Framework Generator (BFG; Ford et al., 2006; Armstrong et al., 2009~~),~~) and the community coupler version 1 (C-Coupler1; Liu et al., ~~), among others. Most of the existing couplers provide fundamental coupling functions that include data transfer between component models and data interpolation between different model grids (Valcke et al., 2012~~).

A coupler generally has much smaller overhead than ~~other~~the component models~~.~~ in a coupled system. However, it is potentially a time-consuming component ~~in an ESM~~of a coupled model in future. This is because ~~there will be~~ more and more component models (such as land-ice model, chemistry model and biogeochemical model) will be coupled into ~~an ESM~~a coupled model, and the coupling frequency between component models will be ~~more~~higher and ~~more frequent~~higher. Data transfer is a fundamental and most frequently used operation in a coupler. It is responsible for transferring data fields between the processes of two component models and ~~responsible~~ for rearranging data fields among~~ various~~ processes of the same component model for parallel data interpolation.

A coupler may become a bottleneck for efficient parallelization of future coupled models. The most obvious reason is that the current implementation of data transfer in a state-of-the-art coupler is not efficient enough~~.~~ for transferring data fields between component models. For example, the data transfer from a component with a logically rectangular grid (of 1021×1442 grid points) to a component with a Gaussian Reduced T799 grid (with 843,000 grid points) can only scale to about 100 processor cores when using OASIS3 (Valcke, 2013) and to about 1000 processor cores when using OASIS3-MCT (Valcke et al., 2013); the data transfer from a component model with a horizontal grid (of 576×384 grid points) to another component model with another horizontal grid (of 3600×2400 grid points) can only scale to about 500 processor cores when using the CPL7 coupler (Craig et al., 2012). Therefore, it is highly desirable to improve the parallelization of couplers.

In this study, we first propose a butterfly implementation of data transfer. Since the P2P implementation and ~~then~~the butterfly implementation can outperform each other in different cases (Section 5), we next develop an adaptive data transfer library that ~~is open to the public~~includes both implementations and can adaptively use the better one for data transfer. Performance evaluation demonstrates that such a library significantly improves the performance of data transfer in most cases and does not ~~decrease~~degrade the performance in any ~~cases~~case. This library has been imported into C-Coupler1 with slight code modification. We believe ~~it can be easily imported into~~that other coupler versions ~~for better performance of data transfer~~can also benefit from it.

The reminder of this paper is organized as follows. We briefly introduce the implementation of data transfer in existing couplers in Section 2. ~~We analyze performance bottlenecks of the existing implementation in Section 3.~~ Details of the butterfly implementation and the adaptive data transfer library are presented in Sections ~~4~~3 and ~~5~~4, respectively. The ~~performance~~performances of ~~the butterfly implementation and the adaptive~~ data transfer ~~library is~~implementations are evaluated in Section ~~6. Conclusion is~~5. Conclusions are given in Section ~~7~~6.

2 ~~Implementation of~~ Data transfer implementations in existing couplers

~~In this section, we focus on the implementation of data transfer in existing couplers, including MCT (Jacob et al., 2015), the OASIS coupler (Redler et al., 2010; Valcke, 2013; Valcke et al., 2013), ESMF (Hill et al., 2004), the FMS coupler (Balaji et al., 2006), the CPL6 coupler (Craig et al., 2005), the CPL7 coupler (Craig et al., 2012)), and C-Coupler1 (Liu et al., 2014). More details of these couplers can be found in the citations given.~~

~~2.1 MCT~~

~~MCT works as a library for model coupling. It can be directly used to construct a coupled model with different component models, and can also be used to develop other couplers, such as OASIS3-MCT, the CPL6 coupler and the CPL7 coupler. It provides fundamental coupling functions, i.e., data transfer and data interpolation, in parallel. To achieve a parallel data transfer, MCT first generates a communication router (known as the data mapping between processes) according to the parallel decompositions of the two component models, and next uses the point-to-point (P2P) communication of the Message Passing Interface (MPI) to transfer data. A data field will be transferred from a process of the source component model to a process of the target component model, only when the two processes have common grid points. A data transfer can serve multiple data fields that will be packed into one MPI message for better communication performance.~~

~~On the other hand, parallel interpolation can also introduce data exchange among processes of the same component model. Interpolation is generally performed by the calculation of matrix-~~

vector multiplication. To achieve efficient parallelization of interpolation, MCT can rearrange the layout of the data field among processes, to enable the matrix-vector multiplication to be performed locally on each process. The data rearrangement is essentially a data transfer.

2.2 The OASIS coupler

The OASIS coupler is mainly developed by the European Centre for Research and Advanced Training in Scientific Computing (CERFACS) since 1991. OASIS3 (Valcke, 2013a) is a 2-D version of the OASIS coupler with broad usage. To transfer a field from one component model to another, a process of OASIS3 first gathers the field from the processes of the source component model and then scatters the field to the processes of the target component model. Each process of OASIS3 can transfer one model field, so that multiple model fields can be transferred in parallel. However, the parallelism of such an implementation is limited by the number of coupling fields. To solve this problem, MCT has been used to develop the latest version of the OASIS coupler (OASIS3-MCT).

OASIS4 is a 3-D version of the OASIS coupler. The data exchange library in the PRISM System Model Interface Library (PSMILe; Redler, 2010), which performs communication with MPI, is used to perform the data transfer in OSIS4. Similar to MCT, each process only needs to send or receive the data of its local decomposition.

In OASIS3, the interpolation of a field is carried out by only one process. Like the implementation of data transfer in OASIS3, the data needed interpolation will be gathered from all processes of the corresponding component model before the interpolation, and will be scattered to all processes after the interpolation. In OASIS4 and OASIS3-MCT, the interpolation is performed in parallel, where all processes of the corresponding component model cooperatively perform the interpolation at the same time. The data rearrangement for the parallel interpolation is implemented by PSMILe in OASIS4 and by MCT in OASIS3-MCT.

2.3 ESMF

Earth System Modeling Framework (ESMF) is a widely used software framework for model development, which defines a superstructure for the architecture of component models and an infrastructure with common coupling functions for model coupling. In ESMF, the coupler components are responsible for regridding and transferring data among component models. The coupler components build the corresponding relationship between the data of the source model and the data of the target model according to their parallel decomposition. Then, the data are transferred in parallel according to the corresponding relationship.

2.4 The FMS coupler

FMS is a software framework developed by the Geophysical Fluid Dynamics Laboratory (GFDL). It supports the development, construction, execution, and scientific interpretation of models. The FMS coupler deploys an exchange grid to perform the coupling. Given the grids of two component models, their exchange grid is generated by all the vertices in the two grids. The coupling fields from a source component model to a target component model, are first interpolated onto the exchange grid, and then averaged onto the target grid. Data transfer among different processors is performed with MPI P2P communications.

2.5 The CPL6 coupler

The CPL6 coupler is a centralized coupler for the Community Climate System Model version 3 (CCSM3; Collins et al., 2006) developed at the National Center for Atmospheric Research (NCAR). The data transfer between component models must go through the coupler. The CPL6 coupler integrates MCT for data transfer and data interpolation. Therefore, the data transfer between component models is processed in parallel with MPI P2P communications and can serve multiple model fields at the same time for better communication performance.

2.6 The CPL7 coupler

The CPL7 coupler is the latest coupler version from the NCAR. It has been used for the ESMs of the Community Climate System Model version 4 (CCSM4; Gent et al., 2011) and the Community Earth System Model (CESM; Hurrell et al., 2013). Similar to the CPL6 coupler, the CPL7 coupler is also a centralized coupler, where the data transfer between component models must go through the coupler. The CPL7 coupler also integrates MCT for data transfer and data interpolation. Moreover, the CPL7 coupler supports the coupling interface based on ESMF and can use the coupling functions in ESMF for data transfer and data interpolation.

2.7 C-Coupler1

C-Coupler1 is a Chinese community coupler for Earth system modeling. It achieves 3D coupling with flexible 3D interpolation, and supports direct coupling without a specific coupler

~~component to improve the parallel performance. Its implementation of data transfer is derived from the corresponding implementation in MCT. In other words, C-Coupler1 first generates a communication router according to the parallel decompositions of the component models, and then uses the MPI P2P communication to transfer the coupling fields in parallel. To further improve the communication performance, model fields with different data types, different model grids, or different parallel decompositions can be served by the same data transfer.~~

2.1 P2P implementation

Almost all state-of-the-art couplers use a similar implementation for data transfer. To achieve parallel data transfer, MCT first generates a communication router (known as the data mapping between processes) according to the parallel decompositions (the distribution of grid points among the processes) of two component models, and then uses the point-to-point (P2P) communication of the Message Passing Interface (MPI) to transfer the data. A data field will be transferred from a process of the source component model to a process of the target component model, only when the two processes have common grid points. In the following context, we call this “P2P implementation” for short.

Since MCT has already been imported into OASIS3-MCT, the CPL6 coupler and the CPL7 coupler, these couplers also use the P2P implementation for data transfer. Although the other couplers such as ESMF, OASIS4, the FMS coupler and C-Coupler1 do not directly import MCT, they also use the P2P implementation for data transfer.

2.2 Performance bottlenecks of the P2P implementation

~~3 Performance bottlenecks of existing implementations~~

~~The implementations of data transfer in~~ Although the ~~state-of-the-art couplers are similar, which can be concluded as the MPI~~ P2P ~~communication that transfers data among the processes according to the two corresponding parallel decompositions. In the following context, we call such an~~ implementation ~~“P2P implementation”~~can achieve good performance when rearranging data fields for ~~short.~~a parallel interpolation in a component model, it is not efficient enough when transferring data between component models (Craig et al., 2012; Valcke, 2013; Valcke et al., 2013; Liu et al., 2014). To reveal why the P2P implementation is ~~inefficient~~not efficient enough, we first derive a benchmark from a real coupled model ~~version~~ GAMIL2-CLM3, ~~where~~ which includes GAMIL2 (Li et al., 2013) that is an atmosphere model and CLM3 (Oleson et al., 2004~~)~~; Dickinson et al., 2006) that is a land surface model. GAMIL2 and CLM3 share the same horizontal grid of 7,680 (128×60) grid points~~.~~, but have different parallel decompositions: GAMIL2 uses a regular 2-D parallel decomposition, while CLM3 uses an irregular 2-D parallel decomposition where the grid points are assigned to the processes in a round-robin fashion.

In this benchmark, there is only the data transfer with P2P implementation between two data models with the same ~~grid as the~~ horizontal grid of GAMIL2-CLM3. The parallel ~~decompositions~~decomposition of the source data model is derived from CLM3, and the parallel decomposition of target data ~~models are the same as those of CLM3 and~~model is derived from GAMIL2~~, respectively~~. A high-performance computer named Tansuo100 at Tsinghua University, China is used for the performance ~~testing~~tests. It has 700 computing nodes, each of which contains two six-core Intel Xeon X5670 CPUs and 32 GB main memory. All computing nodes are connected by a high-speed InfiniBand network with peak communication bandwidth of 5 GB/s.

To evaluate the parallel performance of the P2P implementation, 14 2-D coupling fields are transferred between the two data models. In each test, the two data models ~~have~~use the same number of processes. ~~As~~Since there are 12 CPU cores on each computing node, the number of processes is set to be an integral multiple of 12. When the process number is less than 12, the two data models are located on two different computing nodes. The two data models do not share the same computing node, so the communication of the P2P implementation must go through the InfiniBand network.

Figure 1 demonstrates the poor performance of the P2P implementation. It is well known that the ~~performance of~~ communication performance heavily depends on message size. As shown in ~~Figure~~Fig. 2, the P2P communication bandwidth achieved generally increases with message size~~;~~. So when the message size is small (for example, smaller than 4 KB), the communication bandwidth achieved is very low. The message size in the P2P implementation decreases with increment of process number of models (~~Figure~~Fig. 3), indicating that the communication bandwidth ~~gets~~becomes lower with ~~increase~~the increment of process number. The performance of ~~a~~ data transfer also heavily depends on the MPI message number~~ of MPI messages.~~. As shown in ~~Figure~~Fig. 4, the message number~~ of MPI messages~~ in the P2P implementation increases with increment of process number. Here, we may conclude that the decrease of message size and the increase of message number~~ of MPI messages~~ are primary reasons for the poor performance of the P2P implementation when increasing the process number. However, the ideal performance shown in ~~Figure~~Fig. 5 is much better than the actual performance. The ratio between the ideal performance and the actual performance significantly increases with the increment of processor number. The significant gap between the ideal performance and the actual performance is due to ~~the jam of~~ network ~~communication.~~contention. For example, when multiple P2P communications share the same source process or target process~~,~~ (Fig. 6), they must wait in an order.

3 Butterfly implementation for better performance of data transfer

~~To improve the performance of data transfer, a new implementation should be able to overcome~~ The drawbacks of the P2P implementation~~, which~~ can be concluded as low communication bandwidth due to small message size, variable and big number of MPI messages, ~~and jams in communications. We therefore propose~~as well as network contention. To overcome these drawbacks, a prospective solution is to organize the communication for data transfer using a ~~new implementation called the butterfly implementation. As shown in Figure 6, it is similar to the~~ better structure, so that we investigate the butterfly structure (Fig. 7), which has already been used in the field of computer (Chong et al., 1994; Foster, 1995; Heckbert et al., 1995; Hemmert et al., 2005; Kim et al., 2007; Jan et al., 2013; Petagon et al, 2016). For example, in hardware aspect, the traditional butterfly ~~diagram~~structure and its transformation have been used to design networks (Chong et al., 1994; Kim et al., 2007); in software aspect, the butterfly structure has been used to improve the parallel algorithms with all-to-all communications (Foster, 1995), e.g., Fast Fourier Transform (FFT; Heckbert~~, 1995). The most significant challenge to the butterfly implementation is that the process number needs to be 2^n, where n is a non-negative integer, while the process number of data transfer generally can be any positive integer. To resolve this challenge, we investigated how to efficiently map processes between~~ et al., 1995; Hemmert et al., 2005), matrix transposition (Petagon et al, 2016) and sorting (Jan et al., 2013).

Unfortunately, the improved all-to-all communication with the butterfly ~~implementation and the sender/receiver. Next, we will introduce~~structure cannot be used to improve data transfer, because it requires that one process must communicate with every other process, that the communication load among processes is balanced and that the number of processes must be a power of 2, while the data transfer for model coupling has different charateristics, i.e., one process needs to communicate with a part of other processes (Fig. 6), the communication load among processes is always unbalanced (Fig. 3) and the process number cannot be restricted to a power of 2. Therefore, to benefit from the butterfly structure, we should design a new implementation ~~and~~of data transfer, which is called the butterfly implementation hereafter.

The butterfly implementation uses a butterfly structure to transfer data from the sender with the source parallel decomposition to the receiver with the target parallel decomposition. We call the communication following the butterfly structure "the butterfly kernel". As the process number of the butterfly kernel must be a power of 2, while the process number of the sender or the receiver need not be a power of 2, the butterfly implementation (see Fig. 8) has a process mapping~~.~~ from the sender onto the butterfly kernel and a process mapping from the butterfly kernel onto the receiver, and the butterfly kernel has its own source parallel decomposition and target parallel decomposition, which are determined by the process mappings. Next, we will present the butterfly kernel and the process mappings, respectively.

3.1 ~~The~~ Butterfly ~~implementation~~kernel

~~The butterfly implementation aims to rearrange the data~~The first question for the butterfly kernel is how to decide its process number. Any process of the sender or the receiver can be used as a process of the butterfly kernel. Given that the total number of unique processes of the sender and receiver is N_T, the process number of the butterfly kernel (N_B) can be any power of 2, which is no larger than N_T. We propose to select the maximum number in order for maximum utilization of resources. We prefer to pick out unique processes first from the sender, and then from the receiver if the sender does not have enough processes.

The butterfly kernel is responsible for rearranging the distribution of data among the processes from the source parallel decomposition to the target parallel decomposition. ~~As shown in Figure 6, there are multiple stages in the butterfly implementation. Given~~ Given the process number $N=2^n$, ~~the number of stages is~~ there are n~~+1. Each~~ stages in the butterfly kernel. In a stage ~~has a unique parallel decomposition. The parallel decompositions of the first stage and last stage are determined by the source and target parallel decompositions, respectively, while the parallel decompositions of the other stages are determined by the first and last stages. Between any two successive stages~~, all processes are ~~split~~divided into a number of pairs and the two processes of ~~each~~a pair ~~exchange data according to the corresponding parallel decompositions using~~uses MPI P2P communication to exchange data. Given a process P in the butterfly kernel, after each stage, the number of the processes that may have the data of P on the target parallel decomposition will become a half. Figure 7 is an example for further illustration, where D^i_j

means the data is originally in process P_i according to the source parallel decomposition and is finally in process P_j according to the target parallel decomposition. Before the first stage, all processes (P_0~P_7) may have the data of P_0 on the target parallel decomposition. After the first stage, only four processes (P_0, P_2, P_4 and P_6) may have; and after the second stage, only two processes (P_0 and P_4) may have.

~~Compared to the existing~~To reveal the advantages and disadvantages of the two implementations~~ of data transfer~~, we measure the characteristics of the two implementations based on the benchmark introduced in Section 2.2. The results show the total message size transferred by the butterfly implantation is larger than that by the P2P implementation (Fig. 9), which is the major disadvantage of the butterfly implementation. Meanwhile, comparing with the P2P implementation, the butterfly implementation has the following advantages:

1) bigger message size for better communication bandwidth~~. The message size is M/(2N) on average, where M is the total size of data to be transferred and N is the process number.~~ (Fig. 10);

2) balanced number of MPI messages among processes~~. Each process performs $\log_2 N$ times of MPI communication.~~ (Fig. 11);

3) ordered communications among processes and fewer communications operated concurrently~~. The jam of network communication~~ (Fig. 11), which can ~~be~~ dramatically ~~reduced~~reduce network contention.

3.2 Process mapping

~~Process number of the butterfly kernel must be 2^n, where n is a non-negative integer, while process number of sender or receiver can be any positive integer. The first question is how to decide the number of processes of the butterfly kernel? Any process of the sender or receiver can be used as a process of the butterfly kernel. Given that the total number of unique processes of the sender and receiver is N_T, the process number of the butterfly kernel (N_B) can be any power of 2, which is no larger than N_T. For example, we can select the maximum number in order for maximum utilization of resources. When N_B<N_T, we prefer to pick out processes first from the sender, and then from the receiver if the sender does not have enough processes, in order to save the overhead of process mapping from the sender to the butterfly kernel.~~

~~The second question is how to decide process mapping from the sender to the butterfly kernel and from the butterfly kernel to the receiver.~~In this subsection, we will introduce the process mappings from the sender to the butterfly kernel and from the butterfly kernel to the receiver. To minimize the overhead of process mapping from the butterfly kernel to the receiver, we ~~make~~map one or multiple processes of the butterfly kernel ~~map to~~onto a process of the receiver if the butterfly kernel has more processes than the receiver; otherwise, we ~~make~~map a process of the butterfly kernel ~~map to~~onto one or multiple processes of the receiver. In other words, there is no multiple-to-multiple process mapping between the butterfly kernel and the receiver. Similarly, there is no multiple-to-multiple process mapping between the sender and the butterfly kernel. ~~Processes of the sender or receiver may be unbalanced in terms of size of the data transferred, which may result in unbalanced communications between processes of the butterfly kernel.~~

Processes of the sender or the receiver may be unbalanced in terms of the size of the data transferred, which may result in unbalanced communications among processes of the butterfly kernel. As mentioned in Section ~~4~~3.1, at each stage of the butterfly kernel, all processes are ~~split~~divided into a number of pairs, each of which is involved in P2P communications. To improve the balance of communications among the processes in the butterfly kernel, one solution is to try to make the process pairs at each stage more balanced in terms of data size of P2P communications~~. To achieve balanced data size among process pairs,~~, so we propose to ~~take consideration of the sorting order of~~ reorder the processes ~~in terms~~ of the sender or the receiver according to data size. ~~For example, for the remaining processes that have not been paired,~~At the first stage, each time we ~~can pair~~pick out the process with the largest data size and the process with the smallest data size~~. The pairing of the processes should be conducted iteratively among stages of the butterfly kernel. All processes are taken as the input for the first stage, while output of the pairing for one stage will be the input~~ from the remaining processes that have not been paired, to generate a process group. For the next stage~~.~~, the outputs of two process groups from the previous stage are paired into a bigger process groups in a similar way. After finishing the iterative pairing ~~through~~throughout all stages, all processes of the sender or the receiver are reordered.

The iterative pairing also requires the number of processes to be a power of 2. Given that the process number of ~~processes of~~ the sender (or receiver) is N_C and the process number of the butterfly kernel is N_B, we ~~propose to~~ first pad empty processes (~~the~~whose data size is ~~0~~zero) before the iterative pairing to make the process number of~~ the processes for~~ the sender (or receiver) be a power of 2 (donated N_P), which is no smaller than N_B. Therefore, the reordered N_P processes after the iterative pairing can be divided into N_B groups, each of which contains N_P/N_B processes with consecutive reordered indexes and maps ~~to~~onto a unique process of the butterfly kernel.

Figure ~~7~~12 shows an example ~~for further illustration ~~of the process mapping~~.~~, where the sender has five processes (S_0-S_4 in Fig. 12a), the receiver has 10 processes (R_0-R_9 in Fig. 12b), and the butterfly kernel uses eight processes (B_0-B_7 in Fig. 12c). At the first, empty processes are padded to the sender (S_5-S_7 in Fig. 12a) and the receiver (R_{10}-R_{15} in Fig. 12b). Next, the iterative pairing is conducted for the sender and the receiver, respectively. The iterative pairing has three stages for the sender. At the first stage, the eight processes of the sender are divided into four groups $\{S_1,S_7\}$, $\{S_0,S_6\}$, $\{S_2,S_5\}$ and $\{S_4,S_3\}$ (Fig. 12a), according to the data size corresponding to each process. These four process groups are divided into two bigger groups ($\{\{S_4,S_3\},\{S_2,S_5\}\}$ and $\{\{S_1,S_7\}, \{S_0,S_6\}\}$ at the second stage (Fig. 12a). Finally, one process group $\{\{\{S_4,S_3\},\{S_2,S_5\}\}, \{\{S_1,S_7\}, \{S_0,S_6\}\}\}$ is obtained at the third stage (Fig. 12a), and the eight processes of the sender are reordered as S_4, S_3, S_2, S_5, S_1, S_7, S_0 and S_6, each one of which is mapped onto one process of the butterfly kernel (Fig. 12c). Similarly, the iterative pairing has four stages for the receiver, and the 16 processes of the receiver are reordered as R_9, R_{15}, R_7, R_{12}, R_4, R_8, R_3, R_{10}, R_1, R_{14}, R_5, R_{13}, R_0, R_6, R_2 and R_{11} finally, each two of which are mapped onto one process of the butterfly kernel (Fig. 12c).

4 Adaptive data transfer library

Now, we have two kinds of implementations (the P2P implementation and the butterfly implementation) for data transfer. Although the butterfly implementation can effectively improve the performance of data transfer~~, it still~~ in many cases (examples are given in Section 5), it has some drawbacks: 1) it generally has a larger total message size of communications than the P2P implementation; 2) its stage number is $\log_2 N$ (where N is the number of processes for the butterfly kernel) (Foster, 1995), which may be bigger than the average number of MPI messages per process in the P2P implementation~~.~~ in some cases (for example, the data rearrangement for parallel interpolation). Therefore, it is possible that the P2P implementation outperforms the butterfly implementation in some cases (examples are given in Section ~~6~~5). To achieve optimal performance for data transfer, we propose an adaptive data transfer library that can ~~keep~~take the advantages of the two implementations in all cases.

As introduced in Section ~~4~~3.1, the butterfly implementation is divided into multiple stages. ~~Each stage has a unique intermediate parallel decomposition.~~ Actually, the data transfer ~~between two successive stages~~in one stage can be viewed as a P2P implementation with only one MPI message per process. Inspired by this fact, we try to design an adaptive approach that can combine the butterfly and P2P implementations, where some stages in the butterfly implementation are skipped with the P2P implementations of more MPI messages per process. If all stages of the butterfly implementation are skipped, the adaptive data transfer library will switch to the P2P implementation. Figure ~~8~~13 shows an example of the adaptive data transfer library with ~~8~~eight processes, where Stage ~~1~~2 of the butterfly implementation is skipped with the P2P implementation of ~~3~~three MPI messages per process.

The most significant challenge to such an adaptive approach is how to determine which stage(s) of the butterfly implementation should be skipped. The first ~~solution is~~attempt was to design a cost model that can accurately predict the performance of data transfer in various implementations. We eventually gave up this ~~solution~~ because it was almost impossible to accurately predict the performance of the communications on a high-performance computer, especially when a lot of users share the computer to run various applications. Performance profiling which means directly measuring the performance of data transfer is more practical to determine an appropriate implementation, because the simulation ~~for~~of Earth system ~~modeling~~modelling always takes a long time to run. ~~To obtain an appropriate implementation of~~Figure 14 shows our flowchart of how the adaptive data transfer library~~, we try to successively skip~~ determines an appropriate implementation. It consists of an initialization segment and a profiling segment. The initialization segment generates the ~~stages of the~~process mappings and a candidate implementation that is a butterfly implementation~~. If skipping one stage can achieve better performance, this~~ with no skipped stages. The profiling segment iterates through each stage of the butterfly implementation to determine whether the current stage ~~will~~should be skipped~~; otherwise, it will be~~ or kept. ~~Figure 9 shows a flowchart for determining an~~ In an iteration, the profiling segment first generates a temporary implementation based on the candidate implementation where the current stage is skipped, and then runs the temporary implementation to get the time the data transfer takes. When the temporary implementation is more efficient than the candidate implementation, the current stage is skipped and the temporary implementation replaces the candidate implementation. When the profiling segment finishes, the appropriate implementation ~~of~~is set to be the candidate implementation. To reduce the overhead introduced by the adaptive data transfer library~~. In the algorithm, a stage mask array~~

~~(Stage_mask in the flowchart) specifies which stages are skipped.~~, the profiling segment truly transfers the data for model coupling. In ~~detail, each array element corresponds to a stage of the butterfly implementation. If the value of an array element is false, its corresponding stage is skipped with a P2P implementation. Otherwise, its corresponding stage is kept.~~other words, before obtaining an appropriate implementation, the data is transferred by the profiling segment.

~~The source code of the adaptive data transfer library is mainly written in C++, while the application programming interfaces (APIs) are written in Fortran because most couplers and models are programmed in Fortran. Table 1 lists the APIs, and Figure 10 shows an example of how to use these APIs. The adaptive data transfer library can transfer 2-D and 3-D fields at the same time. Now, it is publicly available at a website (see the code availability section).~~

5 Performance evaluation

~~In order to improve the performance of data transfer for model coupling, we propose the butterfly implementation and an adaptive data transfer library that combines the butterfly implementation and the traditional P2P implementation.~~ In this section, we empirically evaluate the adaptive data transfer library, through comparing it to the butterfly implementation and the P2P implementation. Both toy models and realistic models (GAMIL2-CLM3 and CESM) are used for the performance evaluation. GAMIL2-CLM3 has been introduced in Section ~~3~~2.2. CESM (Hurrell et al., 2013) is a state-of-the-art ESM developed by the National Center for Atmospheric Research (NCAR~~.~~). All the experiments are run on the high performance computer Tansuo100~~ that has been introduced in Section 3~~.

~~In the following context~~Next, we will~~ respectively~~ evaluate the overhead of initialization, the performance in data transfer and the performance in data rearrangement for parallel interpolation.

5.1 Overhead of initialization

We first evaluate the ~~overhead of~~ initialization overhead of ~~different~~data transfer implementations~~ of data transfer.~~. As shown in ~~Figure 11~~Fig. 15, the ~~overheads of~~ initialization ~~of all the three implementations increase~~overhead of each implementation increases with the increment of core number. The initialization overhead of the butterfly implementation is a little higher than that of the P2P implementation, while the initialization overhead of the adaptive data transfer library is ~~4-5~~2-3 folds higher than that of the P2P implementation, because the adaptive data transfer library uses extra time on the performance profiling~~.~~ (please refer to

Section 4). Considering that one data transfer instance should only be initialized ~~only one time~~ at the beginning and executed many times in ~~an ESM~~a coupled model, we can conclude that the initialization overhead of the adaptive data transfer library is reasonable, especially when the simulation is executed for a very long time.

5.2 Performance of data transfer between toy models

~~In this~~ As mentioned in Section 3, the butterfly implementation has different characterizations compared to the P2P implementation. Many factors can impact the performance of a data transfer implementation including MPI message number, the size of data to be transferred (also known as the number of fields in this evaluation) and the number of cores used. In this subsection, we evaluate the performance of data transfer ~~(excluding the initialization overhead) with~~affected by each of these factors. We first build two toy models that use the same logically rectangular grid (of 192×96 grid points). Coupling fields are transferred between the two toy models. In each test, the two toy models ~~have~~use the same ~~process~~ number of cores, and each process has the same MPI message number. ~~The MPI message number of one process can be modified through adjusting the parallel decompositions of the toy models. The factors that impact the performance of a data transfer implementation include the commutation number, the size of the data to be transferred (also known as the number of fields in this evaluation) and the number of processes.~~ Next, we evaluate the performance of data transfer through varying ~~these~~one factor and fixing the other factors.

~~Given a fixed process~~In the first experiment, we fix the number of cores to be 192 and ~~a fixed 2-D~~the coupling field number ~~of~~to be 10, and vary MPI message number per process. Figure ~~12~~16 shows the execution time of one data transfer ~~of~~with different implementations when varying the MPI message number ~~of each~~per process from 1 to 96. The P2P implementation can outperform the butterfly implementation when the MPI message number is small (say, smaller than 12 in ~~Figure 12~~Fig. 16), while the butterfly implementation can outperform the P2P implementation when the MPI message number is big (say, bigger than 12 in ~~Figure 12). Our~~Fig. 16). The adaptive data transfer library ~~can completely keep~~has the best performance ~~of the P2P and butterfly implementations.~~. Moreover, it ~~further~~ improves the performance based on the butterfly implementation when the MPI message number is big, because some butterfly stages in the adaptive data transfer library have been skipped with the P2P implementation. When the MPI message number per process is 96, the adaptive data transfer library can achieve a 13.9-fold performance speedup compared to the P2P implementation.

~~Given different numbers of processes~~In the second experiment, we fix the number of cores and ~~different numbers of~~ MPI ~~messages~~message number per process, and vary the coupling field number transferred. Figure ~~13~~17 shows the execution time of one data transfer ~~in~~with different implementations ~~when varying the number of 2-D coupling fields to be transferred.~~in this experiment. The results show that the execution time of each implementation increases with the increment of data size. When ~~the~~ MPI message number per process is small (~~Figures 13a~~Figs. 17a and ~~13b~~17b), the performance of the butterfly implementation is poorer than that of the P2P implementation, especially when the number of 2-D coupling fields gets bigger. ~~However,~~ The adaptive data transfer library achieves similar performance ~~with~~as the P2P implementation, because it switches to the P2P implementation. When the MPI message number per process is big (~~Figures 13c~~Figs. 17c and ~~13d~~17d), both the butterfly implementation and adaptive data transfer library significantly outperform the P2P implementation, and the adaptive data transfer library ~~further~~ achieves better performance than the butterfly implementation.

~~Given a fixed~~In the third experiment, we fix MPI message number per process to be 24 and ~~a fixed 2-D~~the coupling field number transferred to be 10, and vary the number of cores. Figure ~~14~~18 shows the execution time of one data transfer ~~in~~with different implementations when varying the number of cores. The results show that both the butterfly implementation and adaptive data transfer library achieve better parallel scalability than the P2P implementation. The execution time of the P2P implementation slightly increases with the increment of the number of cores used. However, the execution times of the butterfly implementation and adaptive data transfer library slightly decrease with the increment of the number of the cores used. The butterfly implementation outperforms the P2P implementation, ~~and~~while the adaptive data transfer library achieves better performance than the butterfly implementation.

The resolution of models becomes higher and higher these days. How about the performance of the data transfer implementations when model resolution becomes higher? Higher model resolution means that a model will use more processor cores for accelerating simulation, while the average number of grid points per processor core can remain constant. Considering that the numbers of grid points are always balanced among the processes of a model, we make each process (which runs on a unique processor core) of the toy models have 96 grid points in this evaluation, while enabling processes to have different message numbers and different message sizes. As shown in Fig. 19, although the execution times of all data transfer implementations increase with the increment of processor core number (from 64 to 1024), both the butterfly implementation and the adaptive data transfer library significantly outperform the P2P implementation, and the adaptive data transfer library achieves the best performance. These results indicate that our proposed implementations can significantly improve the performance of data transfer for higher model resolution.

5.3 Performance of data transfer between realistic models

~~Previous evaluation with toy models reveals that the adaptive data transfer library can achieve the best performance among different implementations.~~ In this subsection, we evaluate the performance ~~with~~using two realistic models: GAMIL2-CLM3 (horizontal resolution of 2.8°×2.8°) and CESM (resolution of 1.9x2.5_gx1v6).

For CESM, we use the data transfer between the coupler CPL7 (Craig et al., 2012) and the land surface model CLM4 (Oleson et al., 2004), where 32 2-D coupling fields on the CLM4 horizontal grid (the grid size is 144×96=13824) are transferred. Figure ~~15~~20 shows the performance of one data transfer of different implementations when increasing the process number of both CPL7 and CLM4 from 6 to 192. When the process number is small (say, smaller than 24 in ~~Figure 15~~Fig. 20), the butterfly implementation is much poorer than the P2P implementation, and the adaptive data transfer library achieves similar performance as the P2P implementation~~.~~ becasue it switches to the P2P implementation. However, when the process number gets bigger (say, larger than 24 in ~~Figure 15~~Fig. 20), the adaptive data transfer library dramatically outperforms the P2P implementation with more speedup and also outperforms the butterfly implementation. When each component uses 192 cores, the adaptive data transfer library is 4.01 times faster than the P2P implementation.

For GAMIL2-CLM3, we use the data transfer from CLM3 to GAMIL2 where 14 2-D coupling fields on the GAMIL2 horizontal grid (~~the~~whose grid size is 128×60=7680) are transferred. Figure ~~16~~21 shows the execution time of one data transfer of each implementation when increasing the process number of both GAMIL2 and CLM3 from 6 to 192. The results in ~~Figure 16~~Fig. 21 confirm that the adaptive data transfer library can constantly ~~keep~~show the best performance~~ among different implementations~~. Compared to the P2P implementation, the adaptive data transfer library achieves an 11.68-fold performance speedup when the process number is 96, but achieves a much lower speedup (only 3.48-fold) when the process number is 192. This is because ~~that~~ the average MPI message number per process reduces from 32 to 18 when the number of process increases from 96 to 192.

5.4 Performance of data rearrangement for interpolation

~~For model coupling, besides the~~Besides data transfer between different component models, there is ~~the other~~another kind of data transfer in model coupling that rearranges ~~the~~ data inside a model ~~in order~~ for parallel interpolation of fields between different grids. Here, we use the data rearrangement for the parallel interpolation from the atmosphere grid (~~the~~whose grid size is 144×96=13824) to the ocean grid (~~the~~whose grid size is 320×384=122880) in the coupled model CESM for further evaluation. ~~The results show that~~As mentioned above, the P2P implentation is sufficient for data rearrangement. However, the butterfly implementation is much poorer than the P2P implementation (~~Figure 17~~Fig. 22). This is because the MPI message number is very small (for example, average MPI message number per process is only 6.49 when each model uses 96 cores) for data rearrangement. ~~As a result~~On the other hand, the adaptive data transfer library achieves almost the same performance as the P2P implementation~~.~~ , because it switches to the P2P implementation. Therefore, the adaptive data transfer library can always show the best performance.

5.5 Performance impovement for a coupled model

With the performance improvement of data transfer, we expect that the adaptive data transfer library will improve the performance of coupled models. For this evaluation, we first import the adaptive data transfer library into C-Coupler1 and then use the coupled model GAMIL2-CLM3 that uses C-Coupler1 for coupling to measure performance results. As shown in Fig. 23, the adaptive data transfer library achieves higher performance improvement (when the P2P implementation is used as the baseline) for GAMIL2-CLM3 when using more processor cores. When each component model uses 128 processor cores, the adaptive data transfer library achieves ~7% performance improvement. This performance improvement would not be low because the model coupling only takes a very small proportion of execution time in the simple coupled model GAMIL2-CLM3 and the parallel scalability of the two coupled models GAMIL2 and CLM3 is not good.

6 Conclusions

Data transfer is the fundamental and most frequently used operation in a coupler. This paper demonstrated that the current ~~implementation (which is named as the~~ P2P implementation ~~in this paper)~~ of data transfer in most state-of-the-art couplers is ~~not efficient~~inefficient for transferring data between two component models. To improve the parallel performance of data transfer, we proposed a butterfly implementation. However, ~~the~~ compared to the P2P implementation, the butterfly implementation has both advantages and disadvantages~~, comparing with the P2P implementation~~. The evaluation results showed that the butterfly implementation did not always outperform the P2P implementation. To ~~completely~~ achieve better parallel performance of data transfer, we built an adaptive data transfer library, which combines the advantages of ~~the~~both butterfly implementation and P2P implementation. The evaluation results demonstrated that, the adaptive data transfer library can ~~always achieve the best performance, comparing with the butterfly implementation and P2P implementation. That is to say the adaptive data transfer library can effectively~~significantly improve the performance of data transfer ~~in~~so as to improve a coupled model ~~coupling.~~.

The initialization overhead for the adaptive data transfer library could become expensive when using a large number of processor cores. In the future version, the adaptive data transfer will allow users to record the results of performance profiling offline to save the time used for performance profiling in next runs of the same coupled model.

Code availability

The source code of the adaptive data transfer library is available at https://github.com/zhang-cheng09/Data_transfer_lib.

Acknowledgements

This work is supported in part by the Natural Science Foundation of China (no. 41275098), the National Grand Fundamental Research 973 Program of China (no. 2013CB956603~~),~~) and the Tsinghua University Initiative Scientific Research Program (no. 20131089356).

References

Armstrong, C. W., Ford, R. W., and Riley, G. D.: Coupling integrated Earth System Model components with BFG2, Concurrency and Computation: Practice and Experience, 2009;21;767–791, doi:10.1002/cpe.1348, 2009.

Balaji, V., Anderson, J., Held, I., Winton, M., Durachta, J., Malyshev, S., and Stouffer, R. J.: The Exchange Grid: a mechanism for data exchange between Earth system components on independent grids, In Parallel Computational Fluid Dynamics 2005 Theory and Applications, 2006, 179-186, doi: 10.1016/B978-044452206-1/50021-5, 2006.

~~Collins, W. D., Bitz, C. M., Blackmon, M. L., Bonan, G. B., Bretherton, C~~Chong, F. T., and Brewer, E. A.: Packaging and multiplexing of hierarchical scalable expanders, Parallel Computer Routing and Communication, Springer Berlin Heidelberg, 1994:200-214.

~~. S., Carton, J. A., Chang, P., Doney, S. C., Hack, J. J., Henderson, T. B., Kiehl, J. T., Large, W. G., McKenna, D. S., Santer, B. D., and Smith, R. D.: The Community Climate System Model Version 3 (CCSM3), Journal of Climate, 19(11), 2122–2143, 2006.~~

Craig, A. P., Vertenstein, M., and Jacob, R.: A new flexible coupler for Earth system modelling developed for CCSM4 and CESM1, Int. J. High Perform. C., 26, 31-42, doi:10.1177/1094342011428141, 2012.

Craig, A. P., Jacob, R., Kauffman, B., Bettge, T., Larson, J., Ong, E., Ding, C., and He, Y.: CPL6: the New Extensible, High Performance Parallel Coupler for the Community Climate System Model, Int. J. High Perform. C., 19, 309–327, 2005.

Dennis, J. M.: Inverse space-filling curve partitioning of a global ocean model, In IEEE International Parallel & Distributed Processing Symposium, Long Beach, CA, 2007.

Dennis, J. M. and Tufo, H. M.: Scaling climate simulation applications on the IBM Blue Gene/L system, IBM J. Res. Dev., 52, 117-126, DOI:10.1147/rd.521.0117, 2008.

Dennis, J. M., Edwards, J., Evans, K. J., Guba, O., Lauritzen, P. H., Mirin, A. A., St-Cyr, A., Taylor, M. A., and Worley, P. H.: CAM-SE: a scalable spectral element dynamical core for the Community Atmosphere Model, Int. J. High Perform. C., 26, 74-89, doi:10.1177/1094342011428142, 2012.

Dickinson, R. E., Oleson, K. W., Bonan, G., Hoffman, F., Thornton, P., Vertenstein, M., Yang, Z.-L., and Zeng X.: The Community Land surface model and its climate statistics as a component of the Community Climate System Model, Journal of Climate, 19(11), 2302–2324, 2006.

Ford, R. W., Riley, G. D., Bane, M. K., Armstrong, C. W., and Freeman, T. L.: GCF: a general coupling framework, Concurrency and Computation: Practice and Experience, 18(2), 163–181, 2006.

~~Gent, P. R., Danabasoglu, G., Donner, L. J., Holland, M. M., Hunke, E. C., Jayne, S. R., Lawrence, D. M., Neale, R. B., Rasch, P. J., Vertenstein, M., Worley, P. H., Yang, Z. L., and Zhang, M.: The Community Climate System Model version 4, J. Climate, 24, 4973–4991, 2011.~~

Foster I.: Designing and building parallel programs: concepts and tools for parallel software engineering, Addison-Wesley, 1995.

Heckbert P.: Fourier Transforms and the Fast Fourier Transform (FFT) Algorithm, Computer Graphics, 2: 15-463, 1995.

Hemmert, K. S., and K. D. Underwood.: An analysis of the double-precision floating-point FFT on FPGAs. Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on IEEE, 2005:171-180.

Hill, C., DeLuca, C., Balaji, V., Suarez, M., and da Silva, A.: The Architecture of the Earth System Modelling Framework, Computing in Science & Engineering, 6(1), 18–28, 2004.

Hurrell, J. W., Holland, M. M., Gent, P. R., Ghan, S., Kay, J. E., Kushner, P. J., Lamarque, J.-F., Large, W. G., Lawrence, D., Lindsay, K., Lipscomb, W. H., Long, M. C., Mahowald, N., Marsh, D. R., Neale, R. B., Rasch, P., Vavrus, S., Vertenstein, M., Bader, D., Collins, W. D., Hack, J. J., Kiehl, J., and Marshall, S.: The Community Earth System Model: a framework for collaborative research, Bulletin of the American Meteorological Society, 94(9), 1339–1360, 2013.

Hunke, E. C. and Lipscomb W. H.: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual 4.0, Technical Report LA-CC-06-012, Los Alamos National Laboratory, T-3 Fluid Dynamics Group, 2008.

Hunke, E. C., Lipscomb, W. H., Turner, A. K., Jeffery, N., and Elliott, S.: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual Version 5.0, LA-CC-06-012, Los Alamos National Laboratory, Los Alamos NM, 87545, 115, 2013.

Jacob, R., Larson, J., and Ong, E.: M × N Communication and Parallel Interpolation in Community Climate System Model version 3 using the Model Coupling Toolkit, International Journal of High Performance Computing Applications, 19(3), 293–307, 2005.

Jan, B., Montrucchio, B., Ragusa, C., Khan, F. G., and Khan, O.: Parallel butterfly sorting algorithm on gpu, Acta Press, 2013.

Kerbyson, D. J., and Jones, P. W.: A performance model of the parallel ocean program, International Journal of High Performance Computing Applications, 19(3), 261-276, doi:10.1177/1094342005056114, 2005.

Kim J., Dally W. J., and Abts D.: Flattened butterfly: A cost-efficient topology for high-radix networks, ISCA, 2007, 35(2):126-137.

Li, L. J., Wang, B., Dong, L., Liu, L., Shen, S., Hu, N., Sun, W., Wang, Y., Huang, W., Shi, X., Pu, Y., G. and Yang.: Evaluation of Grid-point Atmospheric Model of IAP LASG version 2 (GAMIL2), Advances in Atmospheric Sciences, 30, 855–867, doi:10.1007/s00376-013-2157-5, 2013.

Liu, L., Yang, G., Wang, B., Zhang, C., Li, R., Zhang, Z., Ji, Y., and Wang, L.: C-Coupler1: a Chinese community coupler for Earth system modeling, Geoscientific Model Development, 7(5), 2281-2302, doi:10.5194/gmd-7-2281-2014, 2014.

Morrison, H., and A. Gettelman: A new two-moment bulk stratiform cloud microphysics scheme in the Community Atmosphere Model, version 3 (CAM3). Part I: Description and numerical tests, Journal of Climate, 21(15), 3642–3659, doi:10.1175/2008JCLI2105.1, 2008.

Neale, R. B., Richter, J. H., Conley, A. J., Park, S., Lauritzen, P. H., Gettelman, A., Williamson, D. L., Rasch, P. J., Vavrus, S. J., Taylor, M. A., Collins, W. D., Zhang, M., and Lin, S.: Description of the NCAR Community Atmosphere Model (CAM 4.0), National Center for Atmospheric Research Ncar Koha Opencat, TN-485+STR, 222p., 2010.

Neale, R. B., Chen, C. C., Gettelman, A., Lauritzen, P. H., Park, S., Williamson, D. L., Conley, A. J., Garcia, R., Kinnison, D., Lamarque, J. F., Marsh, D., Mills, M., Smith, A. K., Tilmes, S., Vitt, F., Morrison, H., Cameron-Smith, P., Collins, W. D., Iacono, M. J., Easter, R. C., Ghan, S. J., Liu, X., Rasch, P. J., and Taylor, M. A.: Description of the NCAR Community Atmosphere Model (CAM 5.0), National Center for Atmospheric Research Ncar Koha Opencat,TN-486+STR, 289p., 2012.

Oleson, K. W., Dai, Y., Bonan, G., Bosilovich, M., Dickinson, R., Dirmeyer, P., Hoffman, F., Houser, P., Levis, S., Niu, G. Y., Thornton, P., Vertenstein, M., Yang, Z. L., and Zeng, X.: Technical Description of the Community Land Surface Model (CLM), National Center for Atmospheric Research Ncar Koha Opencat, TN-461+STR, 186p., 2004.

Petagon, R., and Werapun, J.: Embedding the optimal all-to-all personalized exchange on multistage interconnection networks + + mathContainer Loading Mathjax, Journal of Parallel & Distributed Computing 88(2016):16-30.

Redler, R., Valcke, S., and Ritzdorf, H.: OASIS4–a coupling software for next generation Earth System Modelling, Geoscientific Model Development, 3(1), 87–104, doi:10.5194/gmd-3-87-2010, 2010.

Smith, R., Jones, P., Briegleb, B., Bryan, F., Danabasoglu, G., Dennis, J., Dukowicz. J., Eden, C., Fox-Kemper, B., Gent, P., Hecht, M., Jayne, S., Jochum, M., Large, W., Lindsay, K., Maltrud, M., Norton, N., Peacock, S., Vertenstein, M., and Yeager, S.: The Parallel Ocean Program (POP) reference manual ocean component of the Community Climate System Model (CCSM) and Community Earth System Model (CESM), Los Alamos National Laboratory, LAUR-10-01853, available at http://www.cesm.ucar.edu/models/cesm1.1/pop2/doc/sci/POPRefManual.pdf (last access: 15 October 2015), 141 p., 2010.

Valcke, S., Balaji, V., Craig, A., DeLuca, C., Dunlap, R., Ford, R. W., Jacob, R., Larson, J., O'Kuinghttons, R., Riley, G. D., and Vertenstein, M.: Coupling technologies for Earth System Modelling, Geoscientific Model Development, 5(6), 1589–1596, doi:10.5194/gmd-5-1589-2012, 2012.

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

Valcke, S., Craig, T., and Coquart, L.: The OASIS3-MCT parallel coupler, in: The Second Workshop on Coupling Technologies for Earth System Models (CW2013), available at: https://wiki.cc.gatech.edu/CW2013/images/a/a0/OASIS_MCT_abstract.pdf (last access: 15 October 2015), 2013.

Table 1. The application program interfaces (APIs) of the adaptive data transfer library. Couplers or component models can improve the performance of data transfer through calling these APIs.

API	Brief description	Parameter description
instance_id = ***data_transfer_register_instance***(local_comm, global_rank_remote_root, action)	This API registers one data transfer instance and returns the index of this data transfer instance. A component model can register multiple different data transfer instances.	This API takes local communicator *local_comm*, global rank of the root process in the remote model *global_rank_remote_root* and the transfer direction *action* (send, recv or sendrecv) as input, and returns the instance index *instance_id*.
call ***data_transfer_register_decomp***(instance_id, num_grid_cells, num_local_cells, local_cells_global_index)	This API registers one parallel decomposition to one data transfer instance.	This API takes the instance index *instance_id*, the number of grid cells *num_grid_cells*, the number of local cells *num_local_cells* and the global index of local cells *local_cells_global_index* as input.
call ***data_transfer_register_field*** (instance_id, data_buf, input)	This API registers a coupling field to enable one data transfer instance to access the memory space of this field. One data transfer instance can register multiple coupling fields.	This API takes the instance index *instance_id*, the memory space of this field *data_buf* and the action of this field *input* (true stands for input field and false stands for output field) as input.
call ***data_transfer_register_mask*** (instance_id, mask_array)	This API registers a mask array to enable one data transfer instance to transfer different coupling fields at different coupling steps.	This API takes the instance index *instance_id* and the mask array *mask_array* as input.
call ***data_transfer_init_instance*** (instance_id)	This API initializes one data transfer instance.	This API takes the instance index *instance_id* as input.
call **data_transfer_exec_instance**(instance_id)	This API executes one data transfer instance.	This API takes the instance index *instance_id* as input.
call ***data_transfer_final_instance*** (instance_id)	This API finalizes one data transfer instance.	This API takes the instance index *instance_id* as input.

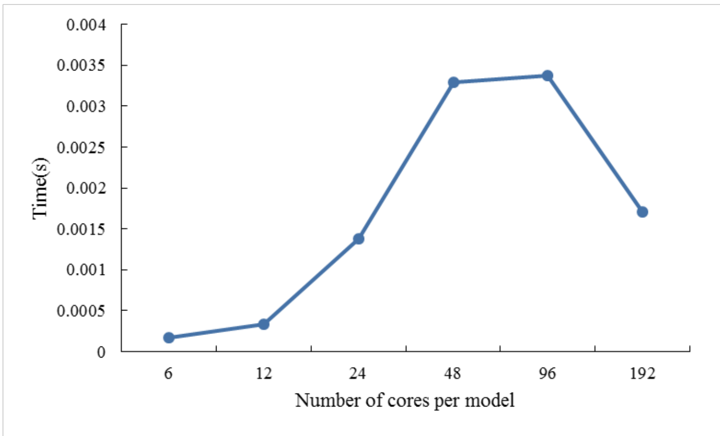


Figure 1. Average execution time of the P2P implementation when transferring 14 2-D fields from CLM3 to GAMIL2. In each test, the atmosphere model GAMIL2 and the land surface model CLM3 use the same number of cores ~~and~~; they do not share the same computing node. The horizontal grid of the 14 2-D fields contains 7680 (128×60) grid points.

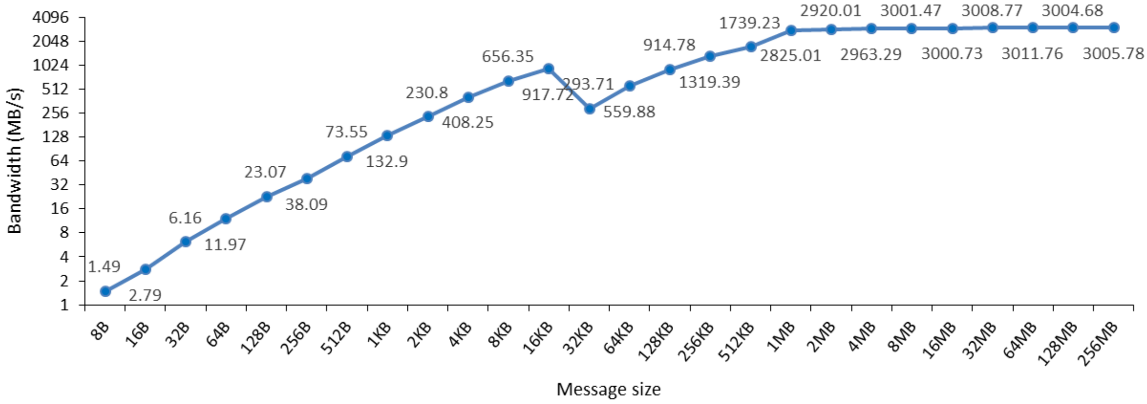


Figure 2. Variation of bandwidth (y-axis) of an MPI P2P communication with respect to the increment of message size. (x-axis). The results are generated from our benchmark. In the benchmark, one process sends messages with different sizes to the other process. The two processes of the P2P communication run on two different computing nodes. of Tansuo100.

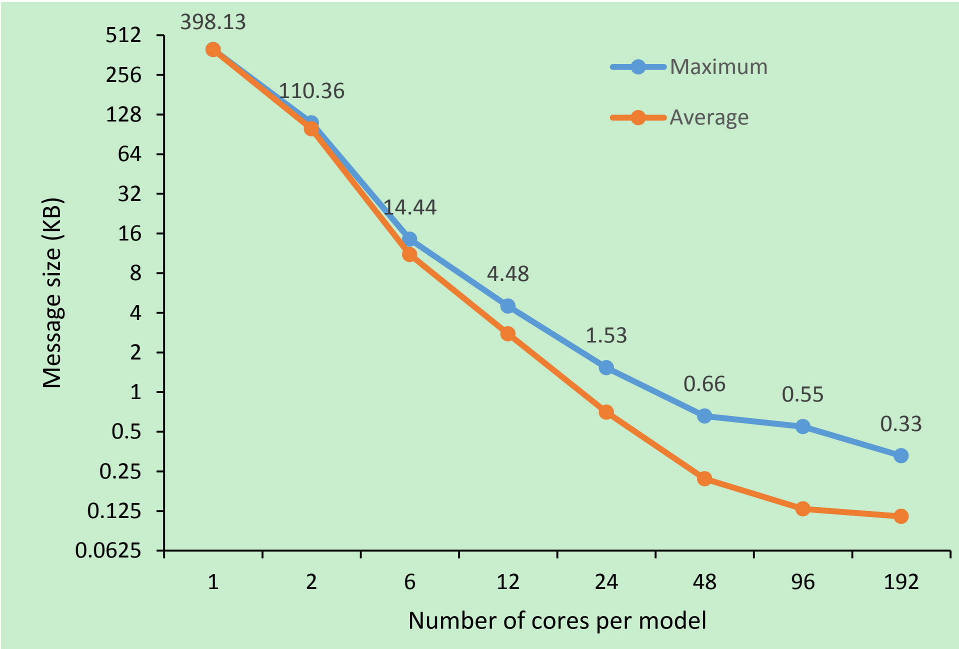


Figure 3. Variation of ~~maximum~~ message size of the P2P implementation (y-axis) in GAMIL2-CLM3 with respect to the increment of ~~process~~core number~~.~~ (x-axis). The experimental setup ~~here~~ is similar ~~with~~to that shown in Fig. 1.

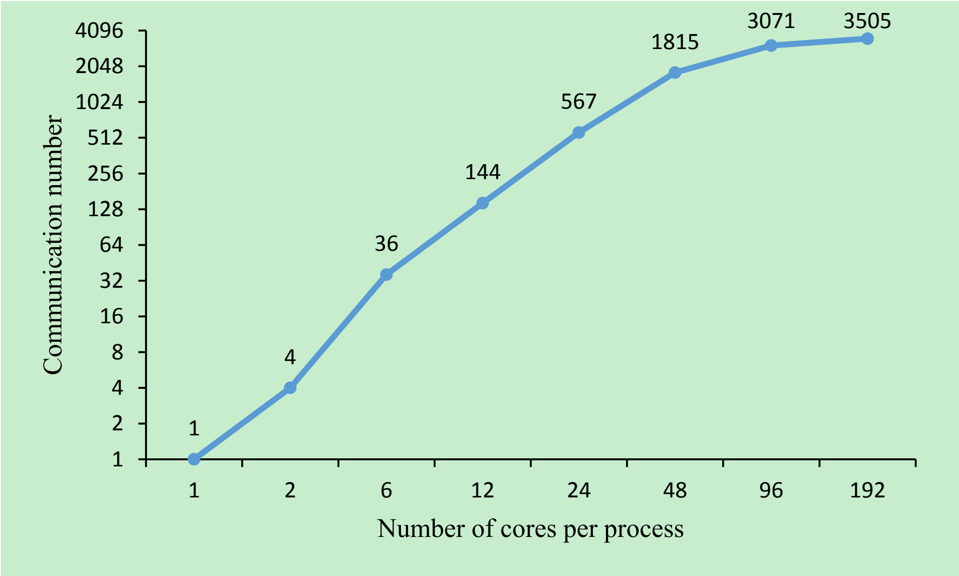


Figure 4. Variation of total MPI message number ~~of MPI messages~~ (y-axis) of the P2P implementation in GAMIL2-CLM3 with respect to the increment of ~~process~~core number (x-axis). The experimental setup ~~here~~ is similar ~~with~~to that shown in Fig. 1.

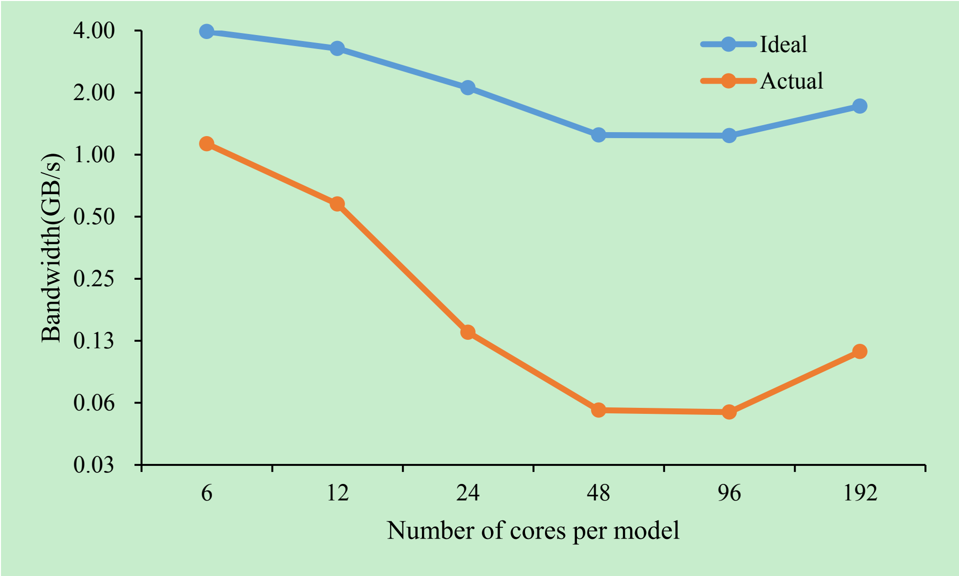


Figure 5. ~~The ~~Ideal and actual bandwidths of the P2P implementation (y-axis) in GAMIL2-CLM3 when gradually increasing the number of ~~processes for ~~cores used by each component model~~.~~ (x-axis). The experimental setup ~~here ~~is similar ~~with~~to that shown in ~~Figure~~Fig. 1. The ideal bandwidth is calculated from the message size and the MPI bandwidth measured in ~~Figure~~Fig. 2~~,~~; and the actual bandwidth is calculated from Fig. 1.

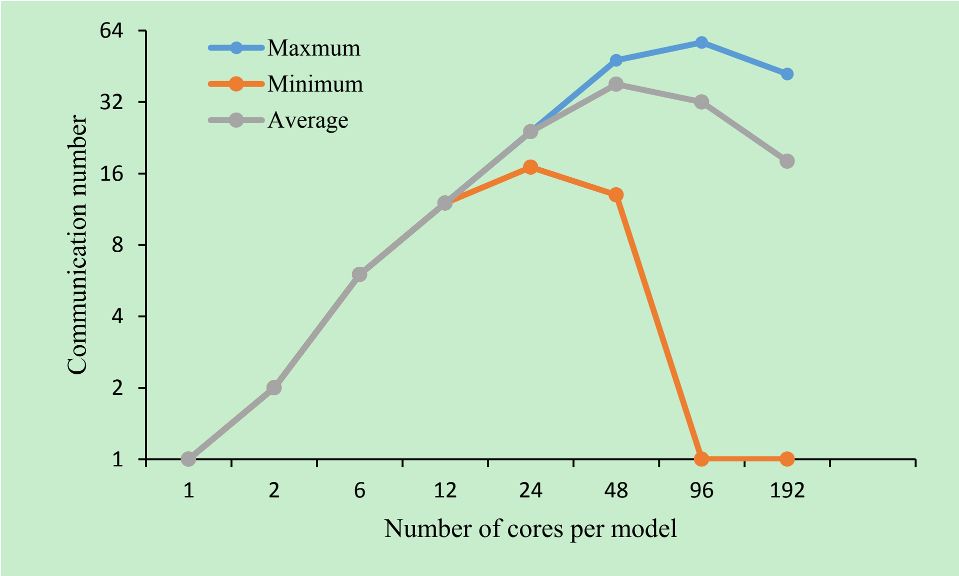


Figure 6. Variation of message number of one process (y-axis) using the P2P implementation in GAMIL2-CLM3 with respect to the increment of core number (x-axis). The experimental setup is similar to that shown in Fig. 1.

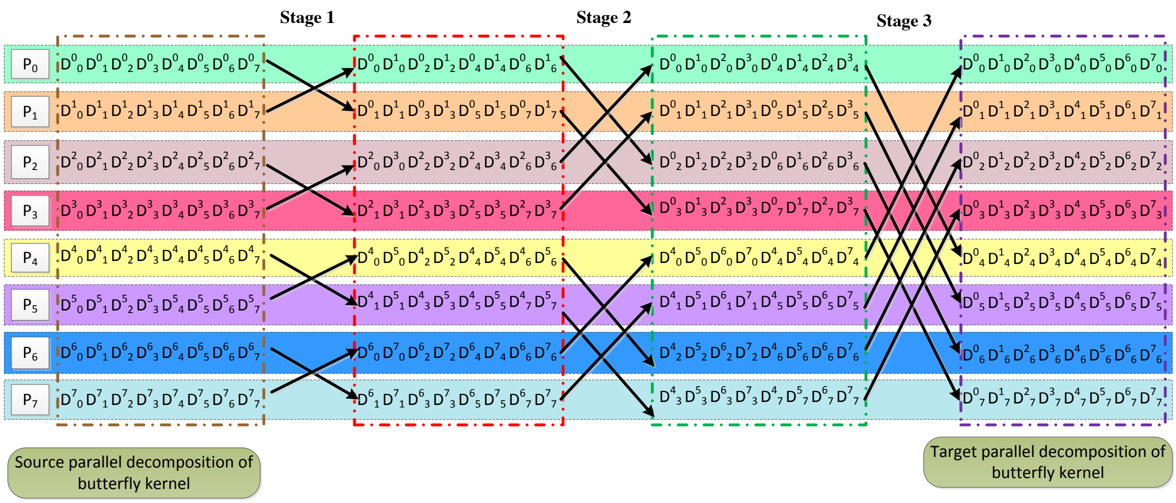


Figure 7. An example of the butterfly ~~implementation with 8 processes. The butterfly implementation targets to rearrange the data from the source parallel decomposition to the target parallel decomposition.~~kernel with eight processes. Each colored row stands for ~~a~~one process (P_0-P_7). ~~D_i represents the subset of data corresponding to process P_i determined by the target parallel decomposition.~~ There are multiple stages (each ~~colored~~ column of arrows represents a stage (Stage ~~0~~1 to Stage 3)) in the butterfly ~~implementation, and each stage has a unique parallel decomposition.~~kernel. Each arrow stands for an MPI P2P communication from one process to another. D^i_j means the data is originally in process P_i according to the source parallel decomposition and is finally in process P_j according to the target parallel decomposition.

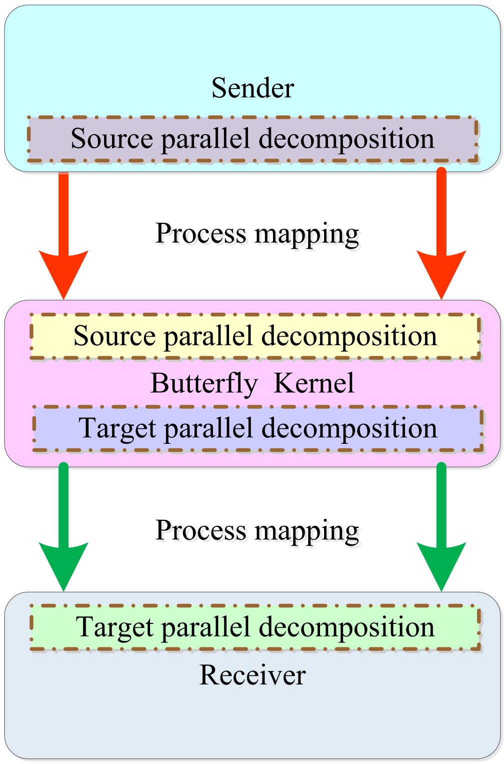


Figure 8. The butterfly implementation, which is composed of three parts: the butterfly kernel; process mapping from the sender to the butterfly kernel; and process mapping from the butterfly kernel to the receiver.

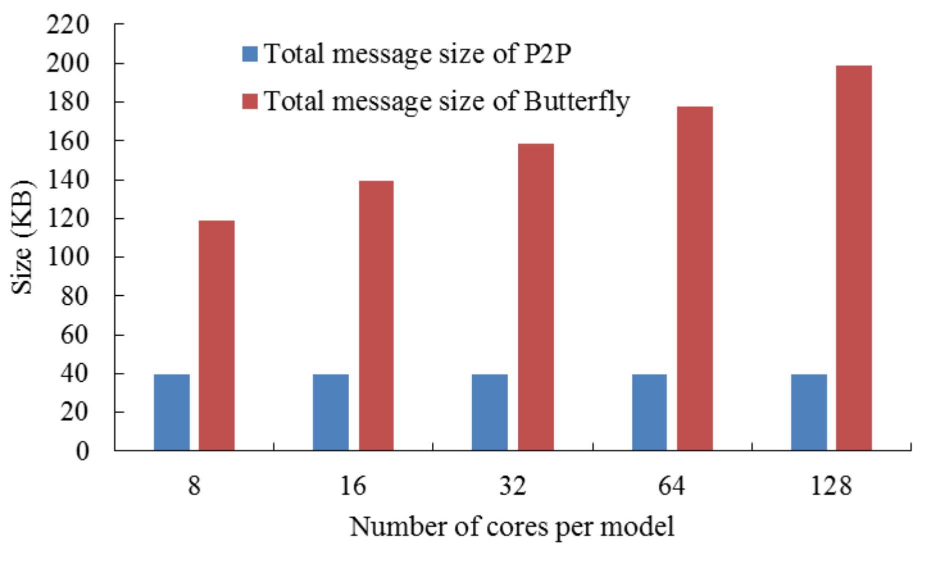


Figure 9. Total message size transferred by P2P implementation and butterfly implementation (y-axis) in GAMIL2-CLM3, when varying the number of cores used by each model (x-axis). The experimental setup is similar to that shown in Fig. 1.

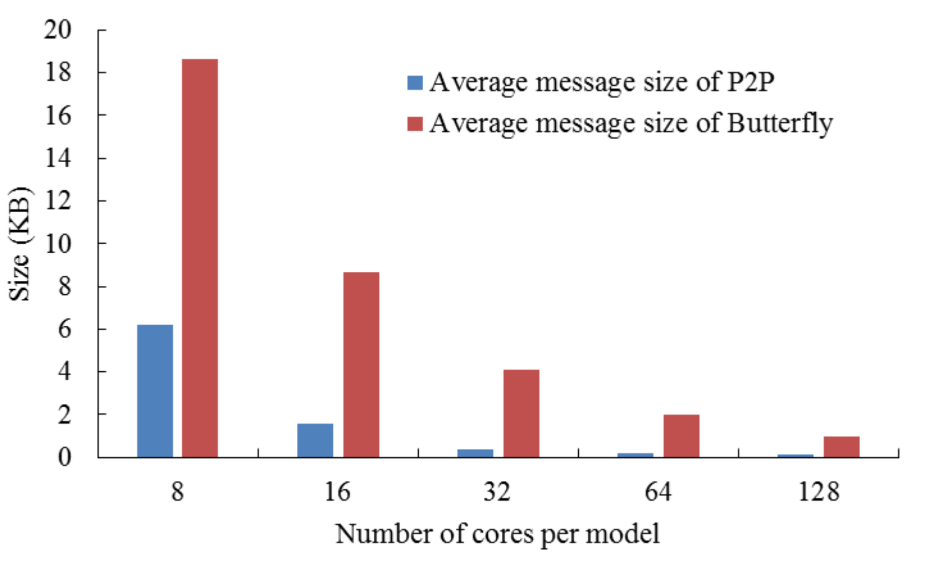


Figure 10. Average message size transferred by P2P implementation and butterfly implementation (y-axis) in GAMIL2-CLM3, when varying the number of cores used by each model (x-axis). The experimental setup is similar to that shown in Fig. 1.

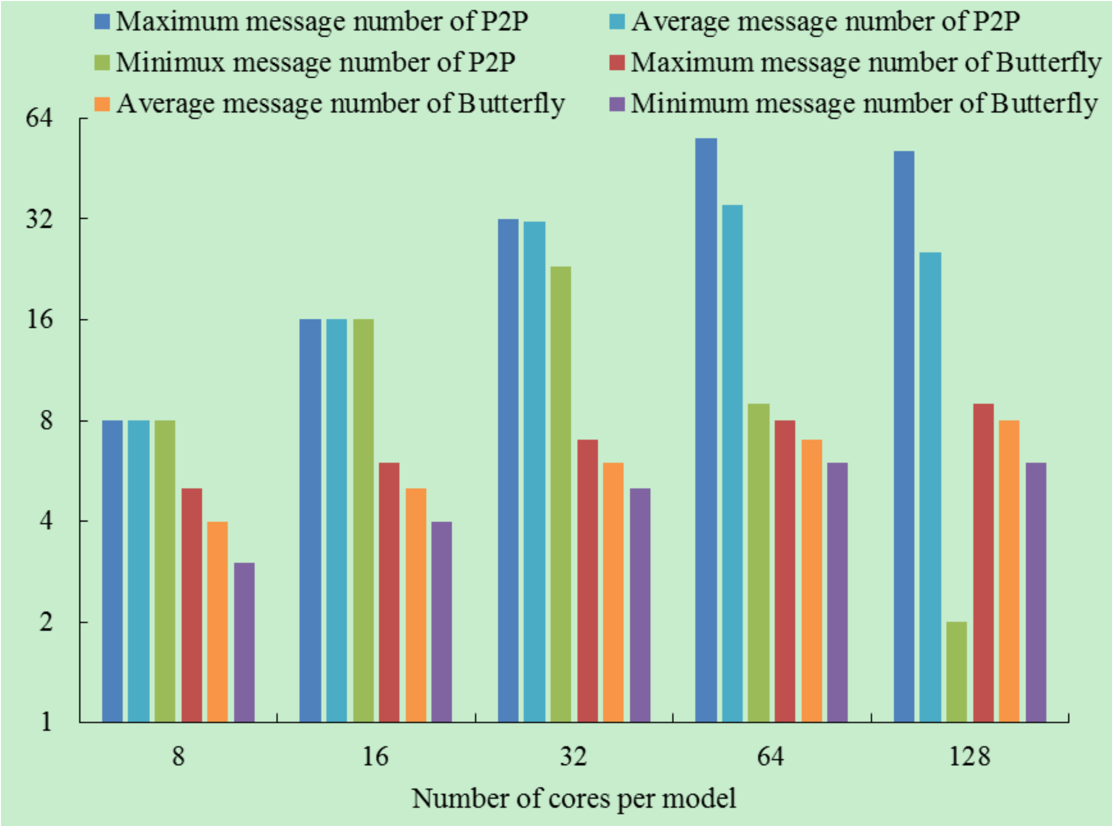
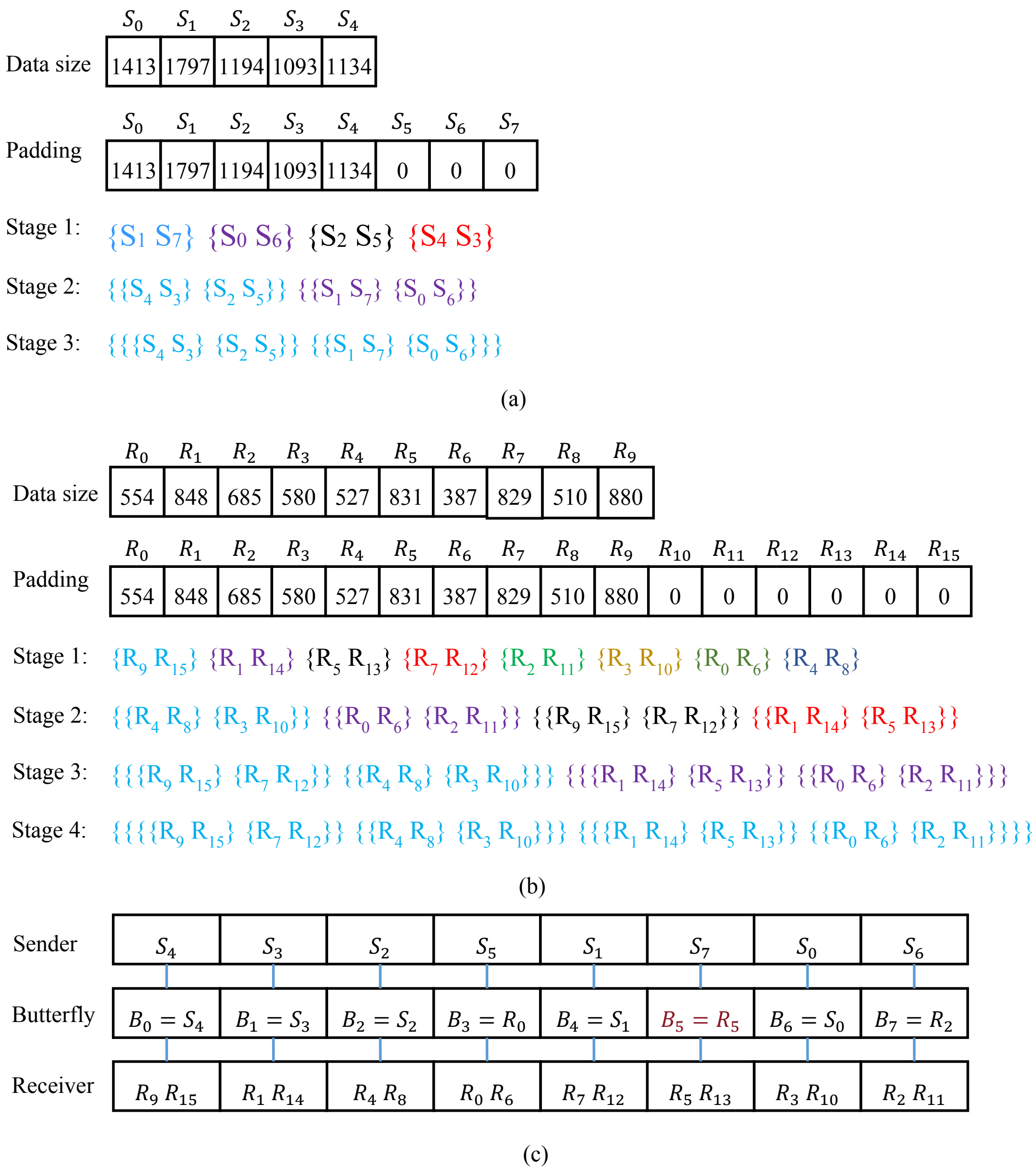


Figure 11. Maximum message number, average message number and minimum message number of processes in P2P implementation and butterfly implementation (y-axis), when varying the number of cores used by each model (x-axis) in GAMIL2-CLM3. The experimental setup is similar to that shown in Fig. 1.



~~Figure 7.~~

Figure 12. An example of process ~~mapping~~mappings, given that the sender has ~~5~~five processes (S_0-S_4), the receiver has 10 processes (R_0-R_9) (there is no common process between the sender and receiver), and the butterfly kernel contains ~~8~~eight processes (B_0-B_7). Panels (a) and (b) show how to iteratively pair processes of the sender and receiver, respectively. There are multiple stages in the iterative pairing of processes of the sender and receiver. In each stage, the processes in the same color are grouped into one process pair. Panel (c) shows how to map the reordered processes of the sender and receiver ~~to~~onto the processes of the butterfly kernel. All ~~the 5~~five processes of the sender ~~are~~ and three processes of the receiver are used ~~for~~as the processes of the butterfly kernel. Each process of the sender is mapped ~~to~~onto a

process of the butterfly kernel, while ~~each~~every two processes of the receiver are mapped ~~to~~onto one process of the butterfly kernel.

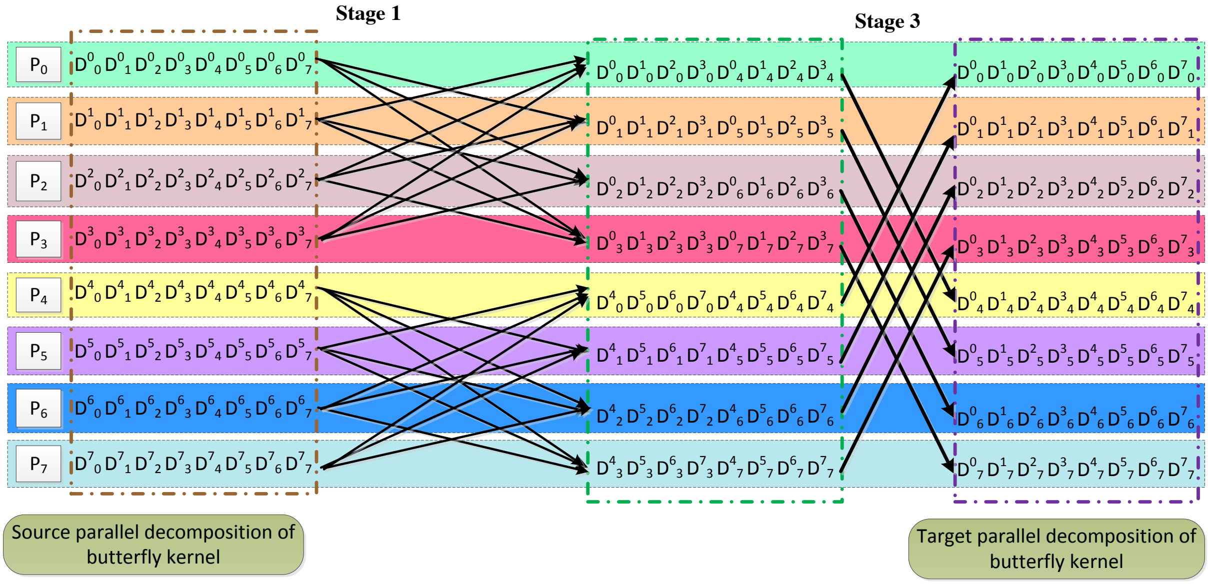


Figure ~~8~~13. An example of the adaptive data transfer library ~~given 8~~with eight processes, where Stage ~~1~~2 of the butterfly implementation is skipped with the P2P implementation of ~~3~~three MPI messages per process.

Input: Process number of the butterfly implementation ***Proc_num***

Output: Stage mask array of the butterfly implementation ***Stage_mask***

Program ***Profiling***

Begin

Set the stage number ***Stage_num*** to be log_2***Porc_num+1***

For ***i=0*** to ***Stage_num-1***; then set ***Stage_mask[i]*** to be ***true***

Execute the butterfly instance with the stage mask array ***Stage_mask***, and record the execution time as ***best_timer***

For ***i=0*** to ***Stage_num-1***

Do

Set ***Stage_mask[i]*** to be ***false***

Execute the butterfly instance with the stage mask array ***Stage_mask***, and record the execution time as ***cur_timer***

If ***best_timer*** is larger than ***cur_timer***

Set ***best_timer*** to be ***cur_timer***

Else set ***Stage_mask[i]*** to be ***true***

End do

End

Figure 10. An example of how to implement data transfer with the APIs of the adaptive data transfer library. The APIs of the adaptive data transfer library are marked in red.

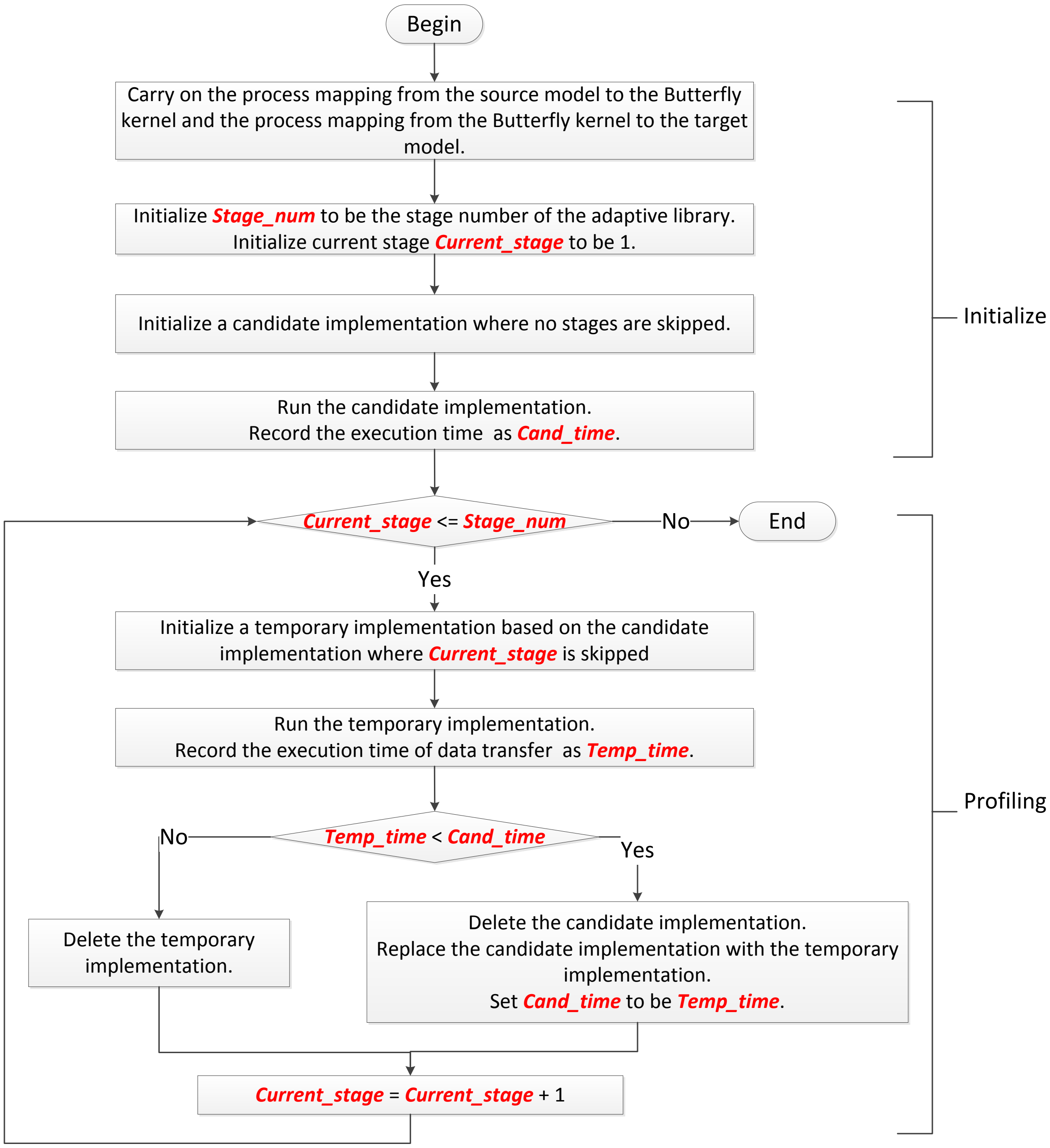
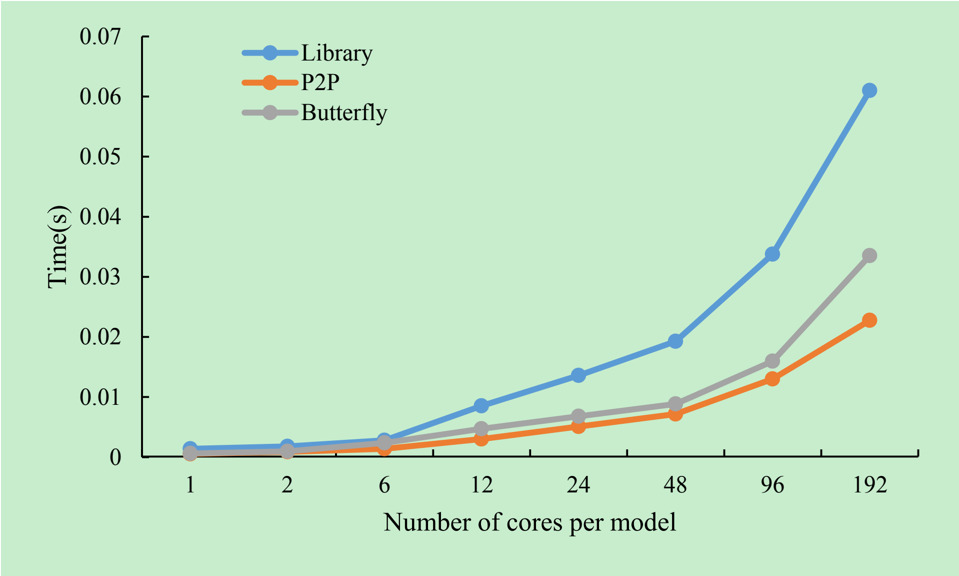


Figure ~~9~~14. A flowchart for determining an appropriate implementation of the adaptive data transfer library.



~~Figure 11.~~Figure 15. Initialization time (y-axis) of one data transfer between two toy models using a rectangular grid (of 192×96 grid points) when varying the number of cores used by each toy model (x-axis). There are 10 2-D coupling fields transferred from the source toy model to the target toy model. If the number of cores per toy model is less than 24, the MPI message number per process is set to be the number of cores. Otherwise, the MPI message number per process is set to 24.

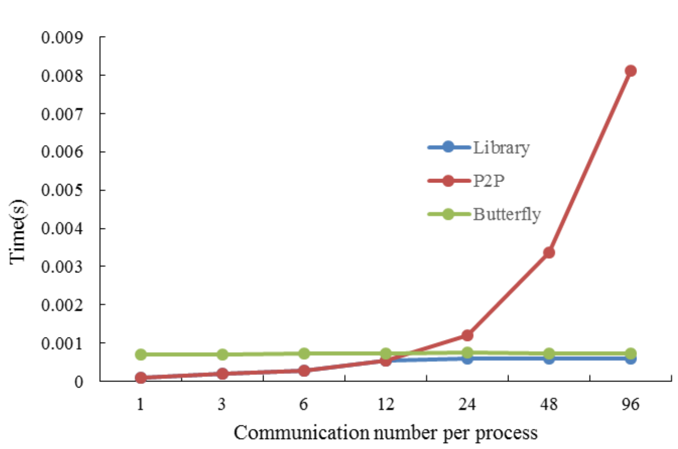


Figure ~~12~~16. Average execution time (y-axis) ~~for~~of one data transfer between two toy models with the same rectangular grid (of 192×96 grid points) when varying the MPI message number per process (x-axis). Each toy model is run with 192 cores~~ (or processes).~~. There are 10 2-D coupling fields transferred from the source toy model to the target toy model.

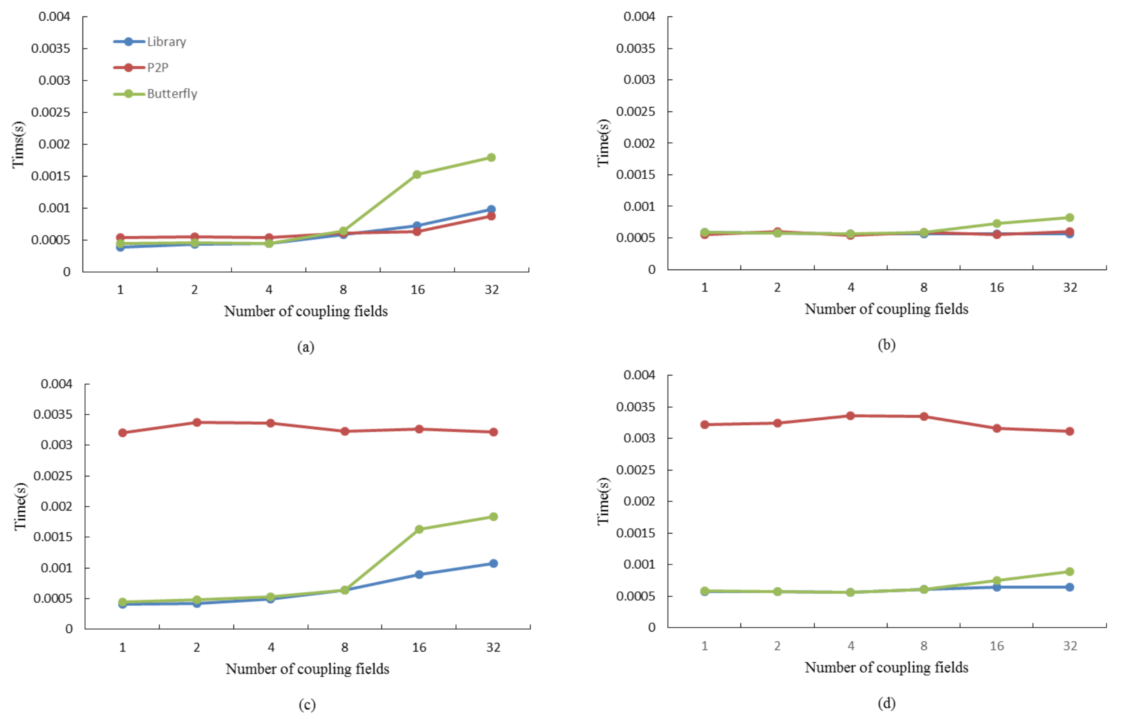


Figure ~~13~~17. Average execution time (y-axis) of one data transfer between two toy models with the same rectangular grid (of 192×96 grid points) when varying the number of coupling fields transferred (x-axis). There are four simulation tests for the evaluation. In simulation (a), each toy model is run with 48 cores, and the MPI message number per process is 12. In simulation (b), each toy model is run with 192 cores, and the MPI message number per process is 12. In simulation (c), each toy model is run with 48 cores, and the MPI message number per process is 48. In simulation (d), each toy model is run with 192 cores (or processes~~)~~), and the MPI message number per process is 48.

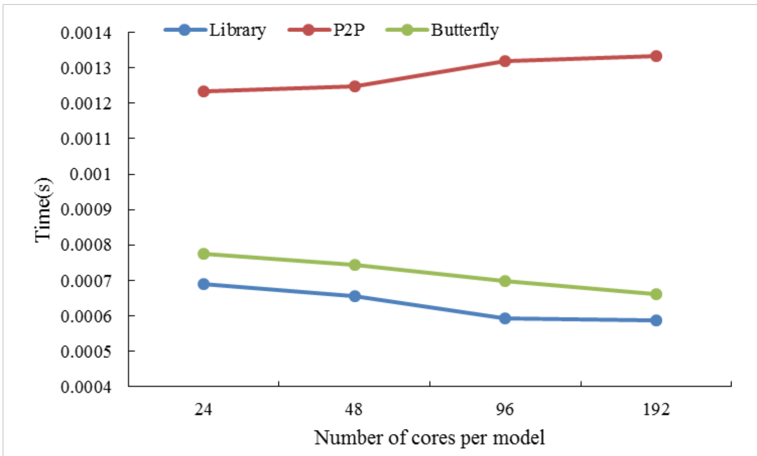


Figure ~~14~~18. Average execution time (y-axis) of one data transfer between two toy models with the same rectangular grid (of 192×96 grid points) when varying the number of cores used by each toy model (x-axis). There are 10 2-D coupling fields transferred from the source toy model to the target toy model. In each test, the MPI message number per process is set to 24.

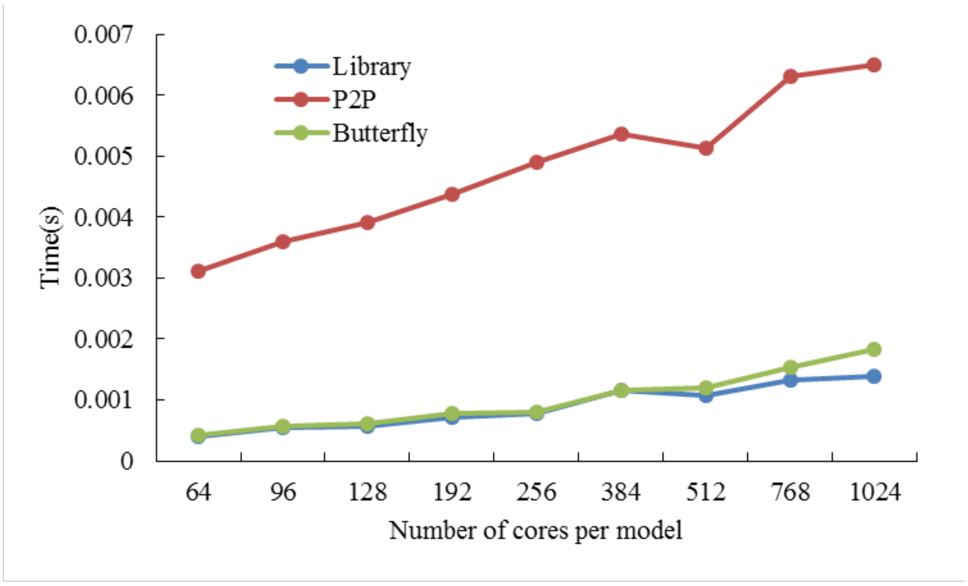


Figure 15.Figure 19. Average execution time (y-axis) of one data transfer between two toy models. In this evaluation, each process (running on a unique processor core) of the toy models have 96 grid points, while different processes have different message numbers and different message sizes. The number of coupling fields transferred is set to 20.

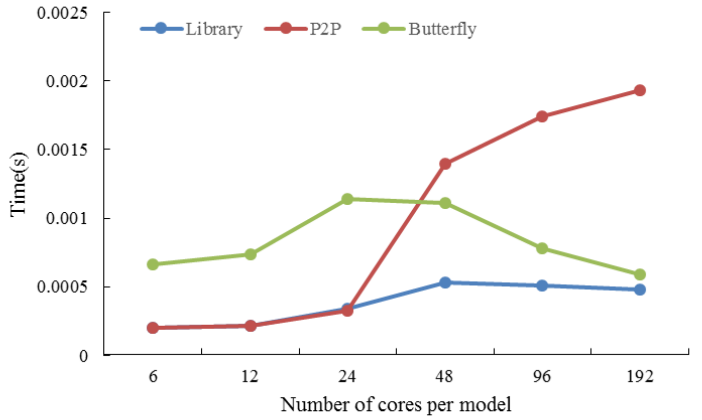


Figure 20. Average execution time (y-axis) of one data transfer between the land surface model CLM4 and the coupler CPL7 in CESM when varying the number of cores used by each model (x-axis): 32 coupling fields on the CLM horizontal grid (the grid size is 144×96=13824) are transferred from the land surface model CLM4 to the coupler CPL7. The P2P results are from the adaptive data transfer library which switches to the P2P implementation.

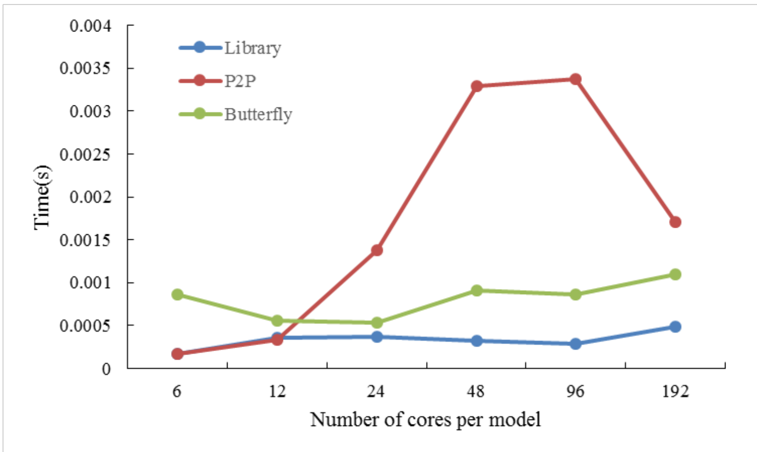


Figure ~~16~~21. Average execution time (y-axis) of one data transfer between the atmosphere model GAMIL2 and the land surface model CLM3 in GAMIL2-CLM3 when varying the number of cores used by each model (x-axis): 14 coupling fields on the GAMIL2 horizontal grid (the grid size is 128×60=7680) are transferred from the land surface model CLM3 to the atmosphere model GAMIL2.

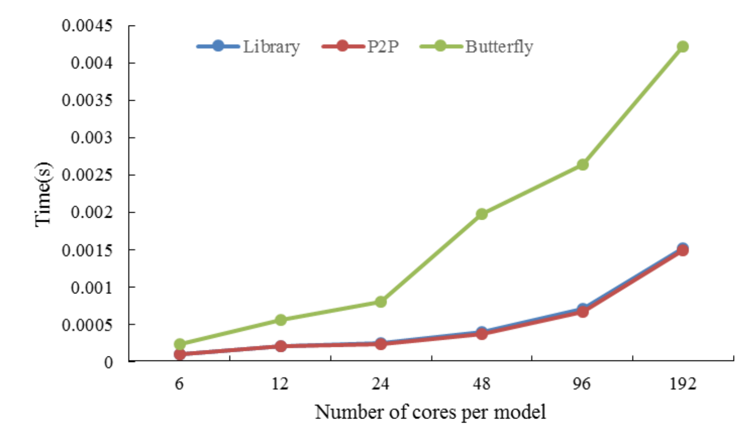


Figure ~~17~~22. Average execution time (y-axis) of one data rearrangement for the parallel interpolation from the atmosphere grid (the grid size is 144×96=13824) to the ocean grid (the grid size is 320×384=122880) in CESM when varying the number of cores used by each model (x-axis).

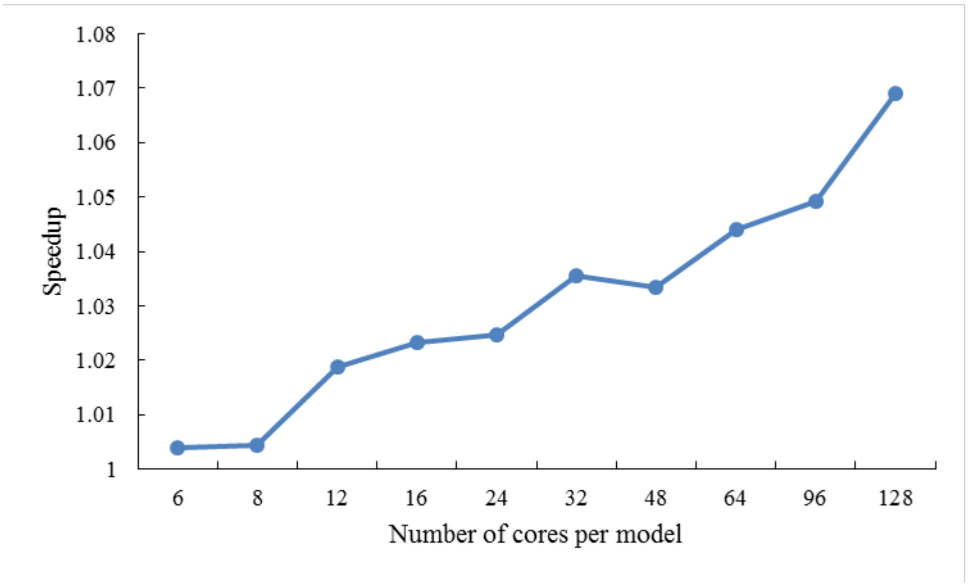


Figure 23. Performance imporvement of the coupled model GAMIL2-CLM3 achieved by the adaptive data transfer library, with the performance of GAMIL2-CLM3 using the P2P implementation as the baseline.