We thank Dr. Sophie Valcke very much for the comments and for the help of improving the English grammar and syntax. We’d like to reply the comments one by one as follows.

1. The writing still needs to be revised in many places. Please consider the modifications I proposed in the .doc file I will send you separately.

Response: The modifications you proposed have been merged into the current revised version.

2. I have never heard “communication depth” used for “number of MPI messages” and I think it is not easy to understand. Please come back to “high number of MPI messages” everywhere in the text!

Response: We replaced “communication depth” with “number of MPI messages”.

3. Please replace “increment of number of processes” by “increasing number of processes” everywhere in the text.

Response: We replaced “increment of number of processes” with “increasing number of processes”.

4. In some places, you use “number of cores per model”, in some other places “number of processes per model”; I know this is equivalent here but the text and figure captions should use the same expression.

Response: We replaced “number of cores per model” with “number of processes per model” in the text, figures and figure captions.

5. Fig 19: I still do not understand the sentence “The performance results of the P2P implementation are obtained through running the adaptive data transfer library when it completely switches to the original P2P implementation.” I propose “The performance results of the P2P implementation are obtained through running the adaptive data transfer library forcing it to completely switch to the original P2P implementation.” but I am not sure this is right.

Response: The sentence has been modified. Please refer to Fig. 19.

A new adaptive data transfer library 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 means transferring data fields from a sender to a receiv~~i~~er. It is a fundamental and ~~most ~~frequently used operation of a coupler. Most versions of state-of-the-art couplers currently use an implementation based on the point-to-point (P2P) communication of the Message Passing Interface (MPI) (referred to ~~such an implementation ~~as “P2P implementation” hereafter~~for short~~). In this paper, we reveal~~ed~~ the drawbacks of the P2P implementation when the parallel decompositions of the sender and the receiver are different, including low communication bandwidth due to small message size, variable and ~~big communication depth~~high number of MPI messages, as well as network contention. To overcome these drawbacks, we propose~~d~~ a butterfly implementation for data transfer. Although the butterfly implementation ~~can ~~outperforms the P2P implementation in many cases, it degrades the performance when the sender and the receiver have similar parallel decompositions or when the number of processes~~or cores~~ used for running models is small. To ensure~~ make~~ data transfer with ~~always keep the ~~optimal performance, we design~~ed~~ and implement~~ed~~ an adaptive data transfer library that combines the advantages of both butterfly implementation and P2P implementation. As the adaptive data transfer library ~~can adaptively~~ automatically uses the best~~tter~~ implementation for data transfer, it outperforms the P2P implementation in many cases while it does not decrease the performance in any cases. Now,

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

1 Introduction

Climate System Models (CSMs) and Earth System Models (ESMs) are fundamental tools for simulating, predicting and projecting 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 behaviours of the climate system, including the interactions between components of the climate system. More and more coupled models have sprung up in the world. For example, the number of coupled model 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 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 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).

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 et al, 2012). A number of couplers now are available, e.g., the Model Coupling Toolkit (MCT; Jacob et al., 2005), the Ocean Atmosphere Sea Ice Soil coupling software (OASIS) coupler (Redler et al., 2010; Valcke, 2013; Valcke et al, 2015), 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., 2014).

A coupler generally has much smaller overhead than the component models in current coupled systems. However, it is potentially a time-consuming component in future coupled models. This is because more and more component models (such as land-ice model, chemistry model and biogeochemical model) will be coupled into a coupled model, and the coupling frequency between component models will be higher and higher. Data transfer is a fundamental and frequently used operation in a coupler. It is responsible for transferring data fields between the processes of two component models and for rearranging data fields among 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 may be not efficient enough. For example, due to the low efficiency of data transfer, the coupling 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 parallel data transfer of couplers.

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

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

2 Data transfer implementations in existing couplers

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 the sender and the receiver, 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 sender to a process of the receiver, 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

~~To motivate~~In this work, we first investigate the performance characteristics of the P2P implementation, and therefore derive a benchmark from a real coupled model GAMIL2-CLM3, 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 the P2P implementation between the sender and the receiver with the same horizontal grid of GAMIL2-CLM3. The parallel decomposition of the sender is derived from CLM3, and the parallel decomposition of the receiver is derived from GAMIL2. A high-performance computer named Tansuo100 at Tsinghua University, China is used for the performance 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 sender and the receiver. In each test, the sender and the receiver use the same number of processes. Since there are 12 processor cores on each computing node, the number of processes is set to be an integral multiple of 12. ~~When the number of processes is less than 12, t~~The sender and the receiver are located on~~ two~~ different computing nodes~~.~~ ~~The sender and the receiver do not share the same computing node, so~~ and the communication of the P2P implementation must go through the InfiniBand network.

Figure 1 demonstrates that poor parallel scalability of the P2P implementation can be obtained when the parallel decompositions of the sender and receiver are different. It is well known that the communication performance heavily depends on message size. As shown in 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~~when the ~~increment of~~ number of model processes ~~of models~~increases (Fig. 3), indicating that the communication bandwidth becomes lower ~~with~~when increasing the~~ increment of~~ number of processes. The performance of data transfer also heavily depends on ~~another term of~~the number of MPI messages~~communication depth, which is defined as the number of the communications that a process is associated with~~. ~~The communication depth is determined by the parallel decompositions of the sender and the receiver. In the P2P implementation, if one process of the sender/receiver has common grid points with *N* processes of the receiver/sender, the communication depth of this process is *N*.~~ As shown in Fig. 4, the variation of average ~~communication depth~~number of MPI messages in the P2P implementation is consistent with the variation of the execution time ~~of the P2P implementation in Fig. 1: both the average communication depth and the execution time of the P2P implementation~~in Fig. 1: both increase with the number of processes~~cores~~ from 6 to 48, and go down with the number of processes~~cores~~ from 96 to 192. Lower execution time of the P2P implementation will be obtained if more processes~~cores~~ are used (the maximum number of processes~~cores~~ in both Fig. 1 and Fig. 4 is limited to 192 because GAMIL2-CLM3 will not be further accelerated when using more processes~~cores~~) since the average number of MPI messages ~~communication depth~~ will further go down.

To further reveal possible reasons for the poor parallel scalability, we evaluate the ideal performance and actual performance in Fig. 5. The ideal performance is much better than the actual performance, and the ratio between the ideal performance and the actual performance significantly increases ~~with the increment of~~when increasing the number of processes. The significant gap between the ideal performance and the actual performance is due to network contention. For example, when multiple P2P communications share the same sender process or receiver process, they must wait in order.

3 Butterfly implementation for better performance of data transfer

The drawbacks of the P2P implementation when the sender and the receiver use different parallel decompositions can be identified as low communication bandwidth due to small message size, variable and high number of MPI messages~~big communication depth~~, as well as network contention. To overcome these drawbacks, a prospective solution is to organize the transfer of data using a better algorithm, e.g., the butterfly algorithm (Fig. 6), which has already been studied in computing sciences (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). In hardware aspect, the traditional butterfly algorithm and its transformation have been used to design networks (Chong et al., 1994; Kim et al., 2007); in software aspect, the butterfly algorithm has been used to improve the parallel algorithms with all-to-all communications (Foster, 1995), e.g., Fast Fourier Transform (FFT; Heckbert et al., 1995; Hemmert et al., 2005), matrix transposition (Petagon et al, 2016) and sorting (Jan et al., 2013).

Unfortunately, the classical butterfly algorithm cannot be used as is to improve data transfer, because it requires that one process communicates with every other process, that the communication load among processes is balanced and that the number of processes must be a power of 2. In practice, data transfer for model coupling has different characteristics~~charateristics~~, i.e., one process needs to communicate with a part of other processes, the communication load among processes is always unbalanced and the number of processes cannot be restricted to a power of 2. Therefore, we propose here a new implementation of data transfer involving an additional butterfly kernel to transfer data from the sender with the source parallel decomposition to the receiver with the target parallel decomposition. As the number of processes of the butterfly kernel must be a power of 2, while the number of processes of the sender or the receiver are not necessarily, the butterfly kernel has its own source~~ parallel decomposition~~ and target parallel decompositions, and process mappings are needed from the sender onto the butterfly kernel and from the butterfly kernel onto the receiver (see Fig. 7). Next, we present the butterfly kernel and the process mappings, respectively.

3.1 Butterfly kernel

The first question for the butterfly kernel is how to decide its number of processes. Any process of the sender or receiver can be used as a process for the butterfly kernel. Given that the total number of unique processes of the sender and receiver is N_T, the number of processes 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~~to maximize 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. Given the number of processes $N=2^n$, there are n stages in the butterfly kernel. In a stage, all processes are divided into a number of pairs and the two processes of a pair uses MPI P2P communication to exchange data. After each stage, the number of butterfly kernel processes that may have the data that will finally belong to any one process on the target parallel decomposition will become a half. Figure 6 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 that data; and after the second stage, only two processes (P_0 and P_4) may have it.

To reveal the advantages and disadvantages of the two implementations, we measure the characteristics of the two implementations based on the benchmark introduced in Section 2.2. The results show that the total ~~message size~~amount of data transferred by the butterfly ~~implantation~~implementation is larger than that by the P2P implementation (Fig. 8), which is the major disadvantage of the butterfly implementation. Meanwhile, comparing with the P2P implementation, the butterfly implementation can have the following advantages:

1) bigger message size for better communication bandwidth (Fig. 9);

2) balanced and smaller number of MPI processes~~communication depth~~ among processes (Fig. 10);

3) ordered communications among processes and fewer communications operated concurrently (Fig. 10), which can dramatically reduce network contention.

3.2 Process mapping

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 map one or multiple processes of the butterfly kernel onto a process of the receiver if the butterfly kernel has more processes than the receiver; otherwise, we map a process of the butterfly kernel 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 the receiver may be unbalanced in terms of~~ the size of~~ the data size transferred, which may result in unbalanced communications among processes of the butterfly kernel. As mentioned in Section 3.1, at each stage of the butterfly kernel, all processes are 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, so we propose to reorder the processes of the sender or the receiver according to data size. At the first stage, each time we pick out the process with the largest data size and the process with the smallest data size 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 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 number of processes of the sender (or receiver) is N_C and the number of processes of the butterfly kernel is N_B, we first pad empty processes (whose data size is zero) before the iterative pairing to make the number of processes of 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 onto a unique process of the butterfly kernel.

Figure 11 shows an example of the process mapping, where the sender has five processes (S_0-S_4 in Fig. 11a), the receiver has 10 processes (R_0-R_9 in Fig. 11b), and the butterfly kernel uses eight processes (B_0-B_7 in Fig. 11c). At first, empty processes are padded to the sender (S_5-S_7 in Fig. 11a) and the receiver (R_{10}-R_{15} in Fig. 11b). 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. 11a), 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. 11a). 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. 11a), 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 being mapped onto one process of the butterfly kernel (Fig. 11c). 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, with pairs of these being mapped onto one process of the butterfly kernel (Fig. 11c).

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 in many cases (examples are given in Section 5), it has some drawbacks: 1) it generally has a larger total ~~message size~~amount of data transferred than the P2P implementation; 2) its number of stages is log_2N (where N is the number of processes for the butterfly kernel) (Foster, 1995), which may be bigger than the average number of MPI messages~~communication depth~~ in the P2P implementation in some cases (for example, when the sender and the receiver use the similar parallel decompositions). Therefore, it is possible that the P2P implementation outperforms the butterfly implementation in some cases. To achieve optimal performance for data transfer, we propose an adaptive data transfer library that can take the advantages of the two implementations in all cases.

As introduced in Section 3.1, the butterfly implementation is divided into multiple stages. Actually, the data transfer 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 and replaced by P2P communication of more MPI messages per process. When all stages of the butterfly implementation are skipped, the adaptive data transfer library completely switches to the original P2P implementation. That is to say, the adaptive data transfer can adaptively choose the optimal implementation from the P2P implementation and the butterfly implementation. Figure 12 shows an example of the adaptive data transfer library with eight processes, where Stage 2 of the butterfly implementation is skipped and replaced by P2P communication of three MPI messages per process.

The most significant challenge of such an adaptive approach is to determine which stage(s) of the butterfly implementation should be skipped. The first attempt was to design a cost model that can accurately predict the performance of data transfer in various implementations. We eventually gave up this approach as 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 of earth system modelling always takes a long time to run. Figure 13 shows our flowchart of how the adaptive data transfer library determines an appropriate implementation. It consists of an initialization segment and a profiling segment. The initialization segment generates the process mappings and a candidate implementation that is a butterfly implementation with no skipped stages. The profiling segment iterates through each stage of the butterfly implementation to determine whether the current stage should be skipped or kept. 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 is set to be the candidate implementation. To reduce the overhead introduced by the adaptive data transfer library, the profiling segment truly transfers the data for model coupling. In other words, before obtaining an optimal implementation, the data is transferred by the profiling segment.

5 Performance evaluation

In this section, we empirically evaluate the adaptive data transfer library, through comparing it to the P2P implementation and the butterfly implementation. Both toy models and realistic models (GAMIL2-CLM3 and CESM) are used for the performance evaluation. GAMIL2-CLM3 has been introduced in Section 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.

Next, we will evaluate the overhead of initialization, the performance of transferring data fields between two toy models and between different realistic component models, and the performance of rearranging data fields ~~intra~~within a component model for parallel interpolation.

5.1 Overhead of initialization

We first evaluate the initialization overhead of data transfer implementations. As shown in Fig. 14, the initialization overhead of each implementation increases when increasing the number of processes~~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 2-3 folds higher than that of the P2P implementation, because the adaptive data transfer library uses extra time on the performance profiling (see Section 4). Considering that one data transfer instance should only be initialized at the beginning and executed many times in 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

The factors that can impact the performance of a data transfer implementation generally include the number of MPI messages~~communication depth~~, the size of the data to be transferred (also referred to as the number of fields in this evaluation) and the number of processes~~cores~~ used. In this subsection, we evaluate the impact of each factor on the performance of data transfer for different implementations. We first build two toy models that both use the same logically rectangular grid of 192×480 grid points. Coupling fields are transferred between the two toy models. For any test, the two toy models use the same number of processes~~cores~~. Next, we evaluate the performance of data transfer through varying one factor while fixing the other two factors.

In the first experiment, we fix the number of processes~~cores~~ to be 1024 and the number of coupling fields to be 10, while only vary the number of MPI messages~~communication depth~~ in the P2P implementation. In each test, all processes of the sender have the same number of MPI messages~~communication depth~~. As the number of MPI messages~~communication depth~~ is determined by the parallel decompositions of the sender and the receiver, we design an algorithm (Algorithm 1) that can generate the parallel decompositions of the two toy models according to the average number of MPI messages~~communication depth~~ of the sender in the P2P implementation. Figure 15 shows the execution time of one data transfer with different implementations when increasing the number of MPI messages~~communication depth~~ per sender process in the P2P implementation from 1 to 90. The P2P implementation can outperform the butterfly implementation when the number of MPI messages~~ communication depth~~ is small (say, smaller than 12 in Fig. 15), while the butterfly implementation can outperform the P2P implementation when the number of MPI messages~~communication depth~~ is big (say, bigger than 12 in Fig. 15). The adaptive data transfer library can adaptively choose the optimal implementation from the P2P implementation and the butterfly implementation, and moreover, it improves the performance based on the butterfly implementation when the number of MPI messages~~communication depth~~ is big, because some butterfly stages of the butterfly implementation are skipped. When the number of MPI messages~~communication depth~~ is 90, the adaptive data transfer library can achieve a 19.2-fold performance speedup compared to the P2P implementation.

In the second experiment, we fix the number of processes~~cores~~ and the number of MPI processes~~communication depth~~ per sender process in the P2P implementation, and vary the number of coupling fields transferred. Figure 16 shows the execution time of one data transfer with different implementations in this experiment. The results show that the execution time of each implementation increases with the increment of data size. When the number of MPI processes~~communication depth~~ per sender process in the P2P implementation is small (Figs. 16a and 16b), 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. When the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is big (Figs. 16c and 16d), the butterfly implementation significantly outperforms the P2P implementation, however, the advantage of the butterfly implementation decreases ~~with the increment of~~ when increasing the number of coupling fields. The results also demonstrate that the adaptive data transfer library can adaptively choose the optimal implementation from the P2P implementation and the butterfly implementation, and can further improve the performance based on the butterfly implementation.

In the third experiment, we fix the number of MPI messages~~communication depth~~ per sender process in the P2P implementation to be 24 and the number of coupling fields transferred to be 10, and vary the number of processes~~cores~~ used. Figure 17 shows the execution time of one data transfer with different implementations when varying the number of processes~~cores~~. The P2P implementation outperforms the butterfly implementation, when small number of processes~~cores~~ are used (say, smaller than 256 in Fig. 17); while the butterfly implementation outperforms the P2P implementation when large number of processes~~cores~~ are used (say, larger than 256 in Fig.17). Similar to above two experiments, the adaptive data transfer library can adaptively choose the optimal implementation from the P2P implementation and the butterfly implementation.

The ~~resolutions~~resolution of models become higher and higher these days. How about the performance of the data transfer implementations when model ~~resolutions become~~resolution becomes higher? Higher model ~~resolutions mean~~resolution means that a model will use more processes~~processor cores~~ for accelerating a simulation, while the average number of grid points per process~~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 evenly have around 96 grid points in this evaluation, while enabling processes to have different number of MPI messages~~communication depth~~ and different message sizes (the average number of MPI messages~~communication depth~~ of the sender in P2P implementation is 34). As shown in Fig. 18, although the execution times of all data transfer implementations increase when increasing the number of processes~~ processor cores~~ (from 64 to 1024), the butterfly implementation significantly outperforms the P2P implementation. So the adaptive data transfer library adaptively chooses the butterfly implementation, and further slightly outperforms the butterfly implementation when each model uses more than 512 ~~core~~processes~~s~~ because some butterfly stages are skipped.

5.3 Performance of data transfer between realistic models

In this subsection, we evaluate the performance 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 19 shows the performance of one data transfer of different implementations when increasing the number of processes of both CPL7 and CLM4 from 6 to 192. When the number of processes is small (say, smaller than 24 in Fig. 19), the butterfly implementation is much poorer than the P2P implementation. In this case, the adaptive data transfer library chooses the P2P implementation as the optimal implementation. However, when the number of processes gets bigger (say, larger than 24 in Fig. 19), the butterfly implementation outperforms the P2P implementation. In this case, the adaptive data transfer library based on the butterfly implementation skip~~pe~~s some stages, so it outperforms the butterfly implementation.~~implmentation.~~ Figure 19 also shows that the butterfly implementation~~implementaion~~ and the adaptive transfer library seem to converge when increasing the number of processes~~cores~~ per model. When each model uses 192 processes~~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 (whose grid size is 128×60=7680) are transferred. Figure 20 shows the execution time of one data transfer of each implementation when increasing the number of processes of both GAMIL2 and CLM3 from 6 to 192. The results in Fig. 20 confirm that the adaptive data transfer library can adaptively choose the optimal implementation from the P2P implementation and the butterfly implementation. Compared to the P2P implementation, the adaptive data transfer library achieves an 11.68-fold performance speedup when the number of processes is 96, but achieves a much lower speedup (only 3.48-fold) when the number of processes is 192. This is because the average number of MPI messages~~communication depth~~ per process in the P2P implementation reduces from 32 to 18 when the number of process increases from 96 to 192.

5.4 Performance of data rearrangement for interpolation

Besides data transfer between different component models, there is another kind of data transfer in model coupling that rearranges data inside a model for parallel interpolation of fields between different grids. Here, we use the data rearrangement for the parallel interpolation from the atmosphere grid (whose grid size is 144×96=13824) to the ocean grid (whose grid size is 320×384=122880) in the coupled model CESM for further evaluation. As shown on Fig. 21, the P2P implementation significantly outperforms the butterfly implementation. This is because the ~~corresponding ~~parallel decompositions ~~for~~before and after data rearrangement are always similar which leads to~~while similar parallel decompositions generally introduce~~ small number of MPI messages~~communication depth~~. For example, average number of MPI messages~~communication depth~~ in the P2P implementation corresponding to Fig. 21 is only 6.49 when the model uses 96 processes~~cores~~. In this case, the P2P implementation is chosen as the optimal implementation of the data transfer library, so the data transfer library~~ library~~ does not provide real benefit compared to the P2P implementation.

5.5 Performance improvement 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 imported the adaptive data transfer library into C-Coupler1~~ and then~~, used it in the coupled model GAMIL2-CLM3~~ that uses C-Coupler1 for coupling to measure~~, and measured performance results. As shown in Fig. 22, the adaptive data transfer library achieves higher speedup with respect to the whole model time~~performance improvement~~ (when the P2P implementation is used as the baseline) for GAMIL2-CLM3 when using more than 16 processes~~processor cores~~. When each component model uses 128 processes~~processor cores~~, the butterfly implementation achieves ~4.6% performance improvement, and the adaptive data transfer library achieves ~6.9% performance improvement. So the data transfer library can improve the performance of data transfer, and then improve the performance of the whole coupled model~~s~~.

6 Conclusions

Data transfer is a fundamental and~~ most~~ frequently used operation in a coupler. This paper showed that the P2P implementation currently used in most state-of-the-art couplers for data transfer is inefficient when the parallel decompositions of the sender and the receiver are different, and further re~~a~~vealed the corresponding performance bottlenecks. ~~To overcome these bottlenecks, we proposed~~We showed that the butterfly implementation~~ that~~ can outperform the P2P implementation in many cases~~, however,~~ but degrades the performance in some cases, for example~~,~~ when a small number of processes~~cores~~ are used to run models or when the parallel decompositions of the sender and receiver are similar. We therefore ~~further~~ designed and implemented an adaptive data transfer library that automatically chooses~~can not only adaptively choose~~ an optimal implementation ~~from~~between the P2P ~~implementation~~one and the butterfly ~~implemtation, but~~one and also further improves the performance based on the butterfly implementation through skipping some butterfly stages. Compared to the P2P implementation, the adaptive data transfer library can improve the performance of data transfer when the parallel decompositions of the sender and the receiver are different.

The initialization overhead for the adaptive data transfer library could become expensive when using a large number of processes~~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 version 1.0 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.

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.

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.

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.

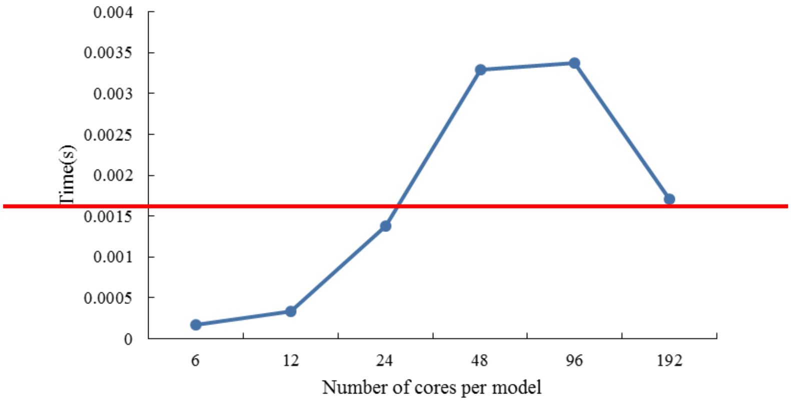
Valcke, S.: The OASIS3 coupler: a European climate modelling community software, Geoscientific Model Development, 6(2), 373–388, doi:10.5194/gmd-6-373-2013, 2013.

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.

Valcke, S., Craig, T. and Coquart, L.: OASIS3-MCT User Guide, OASIS3-MCT_3.0, Technical Report TR/CMGC/15/38, Cerfacs, France, 2015. http://www.cerfacs.fr/oa4web/oasis3-mct_3.0/oasis3mct_UserGuide.pdf

Algorithm 1. Generating the parallel decompositions of the sender and the receiver according to an average number of MPI messages~~communication depth~~ of the sender in the P2P implementation.

Input	Number of processes of the sender: *M* Number of processes of the receiver: *N* Number of points in the grid: *Grid_pnts* Average number of MPI messages~~communication depth~~ per process of the sender in the P2P implementation: *Avg_send_msgs~~depth~~*, *Avg_send_msgs~~depth~~* ≤ *N* The flag that specifies whether the number of MPI messages~~communication depths~~ among processes are the same: *Is_balanced*
Output	Parallel decomposition of the sender Parallel decomposition of the receiver
1	Determine the parallel decomposition of the sender Considering that the numbers of grid points are always balanced among the processes of a model, assign around *Grid_pnts/M* grid points to each process of the sender.
2	Determine the number of MPI messages~~communication depth~~ of each process of the sender
2.1	If the flag *Is_balanced* is set to true, set the number of MPI messages~~communication depth~~ of each process of the sender to be *Avg_send_msgs~~depth~~*;
2.2	Otherwise, randomly determine the number of MPI messages~~communication depth~~ of each process of the sender
2.2.1	Initialize the number of MPI messages~~communication depth~~ of each process of the sender to be 1
2.2.2	Randomly select a process of the sender whose number of MPI messages~~communication depth~~ does not exceed *N* and *Grid_pnts/M*, and then increase its number of MPI messages~~communication depth~~ by 1, until the average number of MPI messages~~communication depth~~ of all processes of the sender reaches *Avg_send_msgs~~depth~~*.
3	Determine the grid points of each MPI message~~communication~~ For each process of the sender, assign the corresponding grid points to all MPI message~~communication~~s of this process (a grid point belongs to only one MPI message~~communication~~)
3.1	If the flag *Is_balanced* is set to true, assign the grid points to all MPI message~~communication~~s evenly.
3.2	Otherwise, assign the grid points to each MPI message~~communication~~ randomly
3.2.1	Assign one grid point to each MPI message~~communication~~
3.2.2	For each of remaining grid points, randomly select an MPI message~~a communication~~ for it
4	Determine the parallel decomposition of the receiver through assigning the grid points in each MPI message~~communication~~ to a process of the receiver For each process of the sender, assign the grid points in each MPI message~~communication~~ of it to a distinct receiver process: to make the numbers of grid points balance among the processes of the receiver in the final parallel decomposition, an MPI message~~a communication~~ with bigger number of grid points will be assigned to a receiver process with smaller total number of grid points that have been assigned to it.



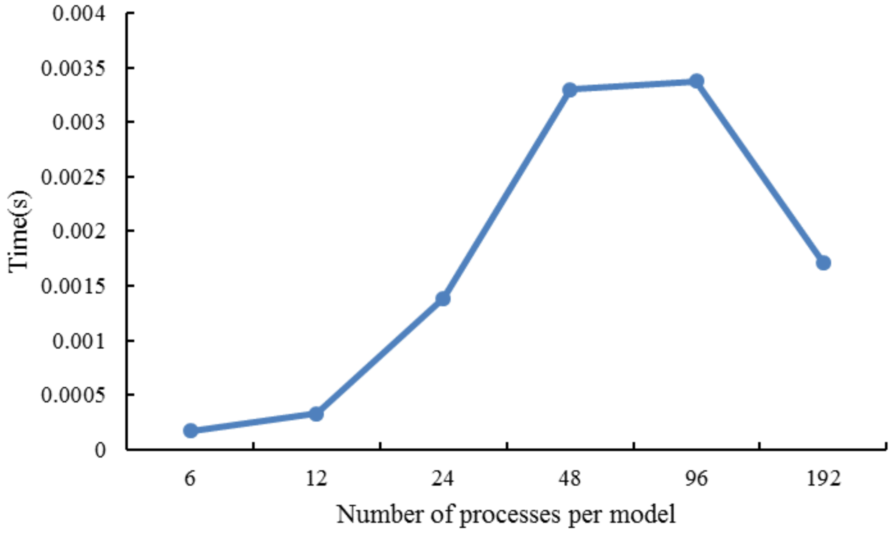


Figure 1. Average execution time of the P2P implementation when transferring 14 2-D fields from CLM3 to GAMIL2. In each test, the atmosphere model GAMIL2 and the land surface model CLM3 have~~use~~ the same number of processes~~cores~~; they do not share the same computing nodes. The horizontal grid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

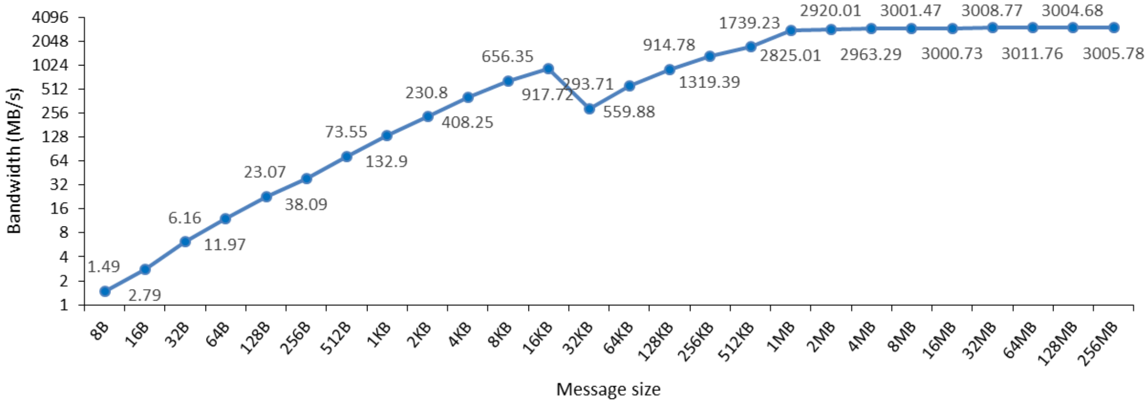
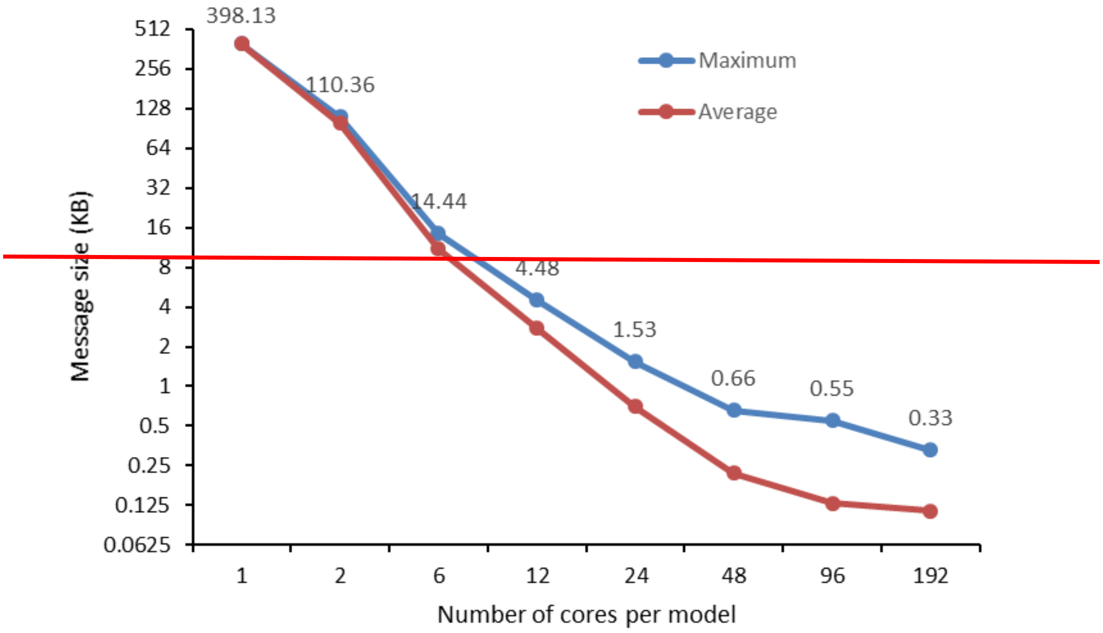


Figure 2. Variation of bandwidth (y-axis) of an MPI P2P communication with respect to the message size (x-axis). The results are generated from our benchmark. In the benchmark, one process sends messages with different sizes to the other process. The two processes of the P2P communication run on two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



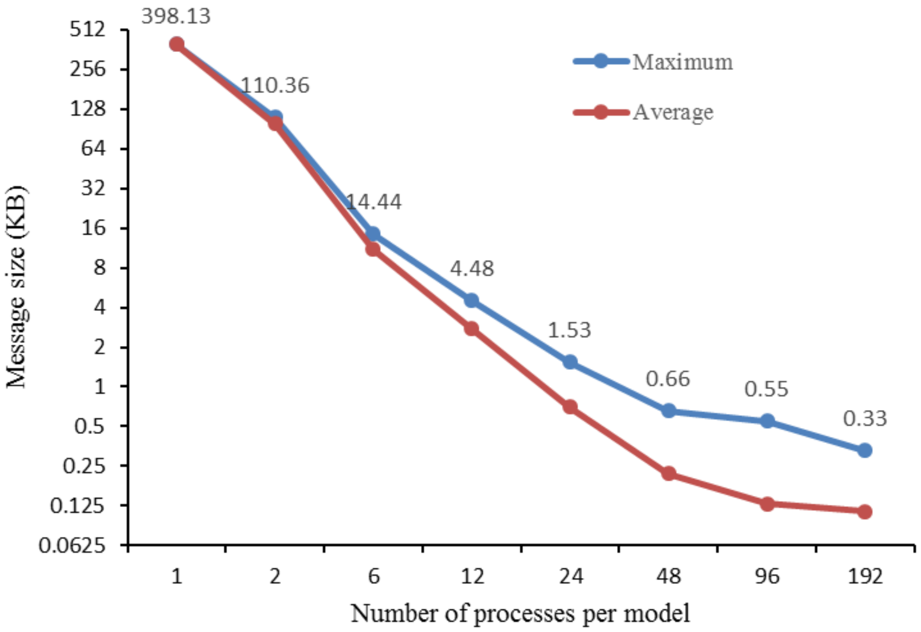
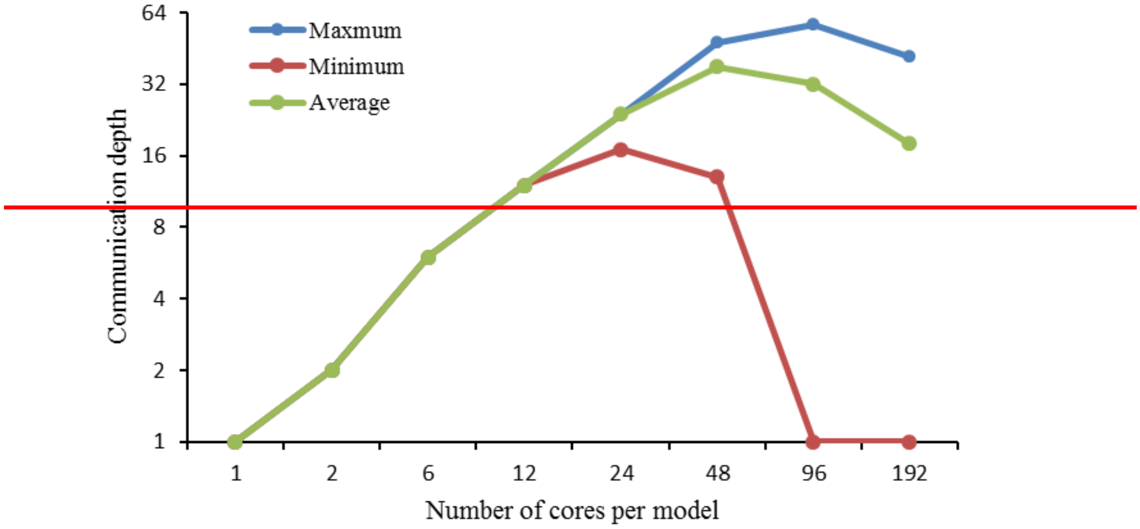
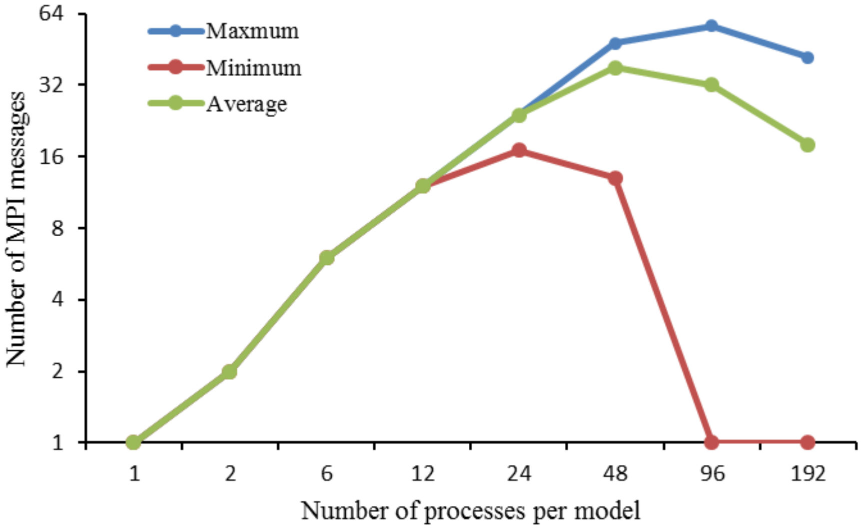


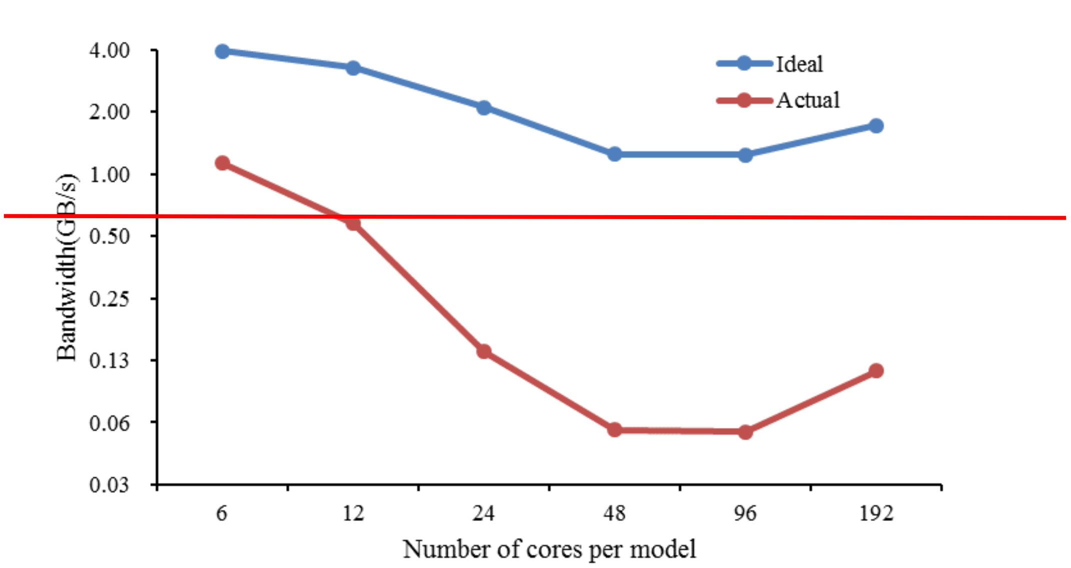
Figure 3. Variation of message size of the P2P implementation (y-axis) in GAMIL2-CLM3 with respect to the number of processes~~cores~~ per model (x-axis). The experimental setup is similar to that shown in Fig. 1.





-

Figure 4. Variation of the number of MPI messages~~communication depth~~ of one process (y-axis) using the P2P implementation in GAMIL2-CLM3 with respect to the number of processes~~cores~~ per model (x-axis). The experimental setup is similar to that shown in Fig. 1.



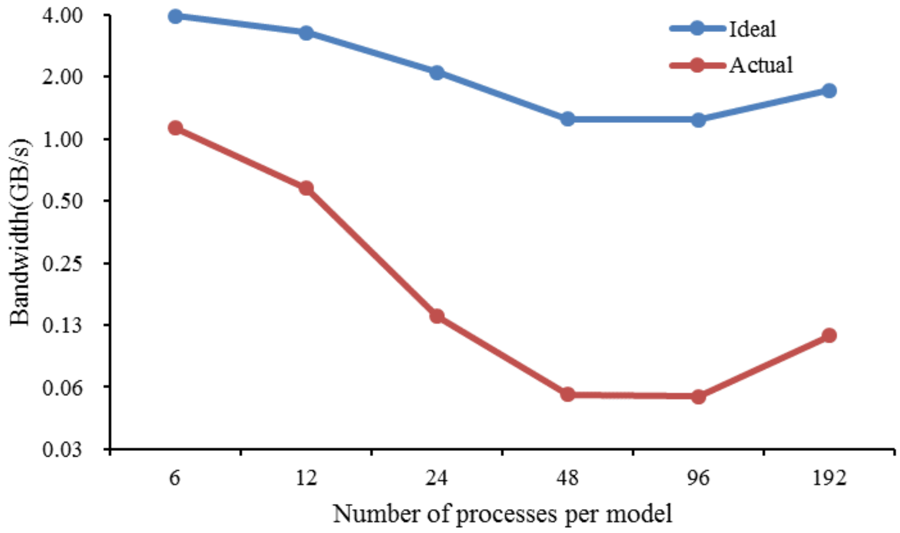


Figure 5. Ideal and actual bandwidths of the P2P implementation (y-axis) in GAMIL2-CLM3 when gradually increasing the number of processes per model~~cores used by each component model~~ (x-axis). The experimental setup is similar to that shown in Fig. 1. The ideal bandwidth is calculated from the message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

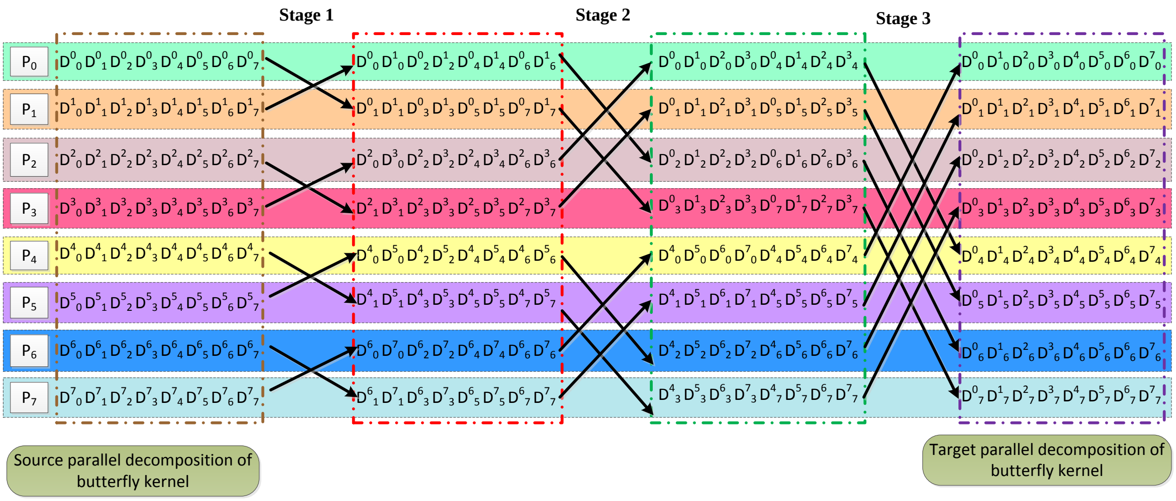

Stage 1
Stage 2
Stage 3
P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7
$D^0_0 D^0_1 D^0_2 D^0_3 D^0_4 D^0_5 D^0_6 D^0_7$
$D^1_0 D^1_1 D^1_2 D^1_3 D^1_4 D^1_5 D^1_6 D^1_7$
$D^2_0 D^2_1 D^2_2 D^2_3 D^2_4 D^2_5 D^2_6 D^2_7$
$D^3_0 D^3_1 D^3_2 D^3_3 D^3_4 D^3_5 D^3_6 D^3_7$
$D^4_0 D^4_1 D^4_2 D^4_3 D^4_4 D^4_5 D^4_6 D^4_7$
$D^5_0 D^5_1 D^5_2 D^5_3 D^5_4 D^5_5 D^5_6 D^5_7$
$D^6_0 D^6_1 D^6_2 D^6_3 D^6_4 D^6_5 D^6_6 D^6_7$
$D^7_0 D^7_1 D^7_2 D^7_3 D^7_4 D^7_5 D^7_6 D^7_7$
$D^0_0 D^1_0 D^0_2 D^1_2 D^0_4 D^1_4 D^0_6 D^1_6$
$D^0_1 D^1_1 D^0_3 D^1_3 D^0_5 D^1_5 D^0_7 D^1_7$
$D^2_0 D^3_0 D^2_2 D^3_2 D^2_4 D^3_4 D^2_6 D^3_6$
$D^2_1 D^3_1 D^2_3 D^3_3 D^2_5 D^3_5 D^2_7 D^3_7$
$D^4_0 D^5_0 D^4_2 D^5_2 D^4_4 D^5_4 D^4_6 D^5_6$
$D^4_1 D^5_1 D^4_3 D^5_3 D^4_5 D^5_5 D^4_7 D^5_7$
$D^6_0 D^7_0 D^6_2 D^7_2 D^6_4 D^7_4 D^6_6 D^7_6$
$D^6_1 D^7_1 D^6_3 D^7_3 D^6_5 D^7_5 D^6_7 D^7_7$
$D^0_0 D^1_0 D^2_0 D^3_0 D^0_4 D^1_4 D^2_4 D^3_4$
$D^0_1 D^1_1 D^2_1 D^3_1 D^0_5 D^1_5 D^2_5 D^3_5$
$D^0_2 D^1_2 D^2_2 D^3_2 D^0_6 D^1_6 D^2_6 D^3_6$
$D^0_3 D^1_3 D^2_3 D^3_3 D^0_7 D^1_7 D^2_7 D^3_7$
$D^4_0 D^5_0 D^6_0 D^7_0 D^4_4 D^5_4 D^6_4 D^7_4$
$D^4_1 D^5_1 D^6_1 D^7_1 D^4_5 D^5_5 D^6_5 D^7_5$
$D^4_2 D^5_2 D^6_2 D^7_2 D^4_6 D^5_6 D^6_6 D^7_6$
$D^4_3 D^5_3 D^6_3 D^7_3 D^4_7 D^5_7 D^6_7 D^7_7$
$D^0_0 D^1_0 D^2_0 D^3_0 D^4_0 D^5_0 D^6_0 D^7_0$
$D^0_1 D^1_1 D^2_1 D^3_1 D^4_1 D^5_1 D^6_1 D^7_1$
$D^0_2 D^1_2 D^2_2 D^3_2 D^4_2 D^5_2 D^6_2 D^7_2$
$D^0_3 D^1_3 D^2_3 D^3_3 D^4_3 D^5_3 D^6_3 D^7_3$
$D^0_4 D^1_4 D^2_4 D^3_4 D^4_4 D^5_4 D^6_4 D^7_4$
$D^0_5 D^1_5 D^2_5 D^3_5 D^4_5 D^5_5 D^6_5 D^7_5$
$D^0_6 D^1_6 D^2_6 D^3_6 D^4_6 D^5_6 D^6_6 D^7_6$
$D^0_7 D^1_7 D^2_7 D^3_7 D^4_7 D^5_7 D^6_7 D^7_7$
Source parallel decomposition of butterfly kernel
Target parallel decomposition of butterfly kernel

Figure 6. An example of the butterfly kernel with eight processes. Each colored row stands for one process (*P_0-P_7*). There are multiple stages (each column of arrows represents a stage (Stage 1 to Stage 3)) in the butterfly 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.

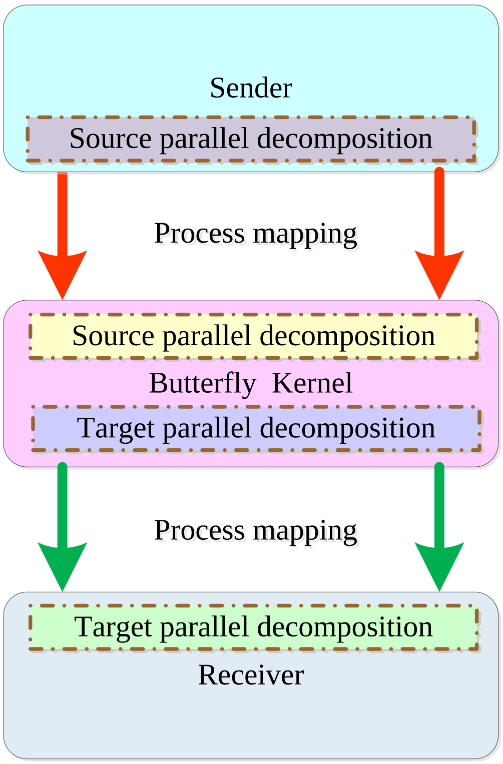
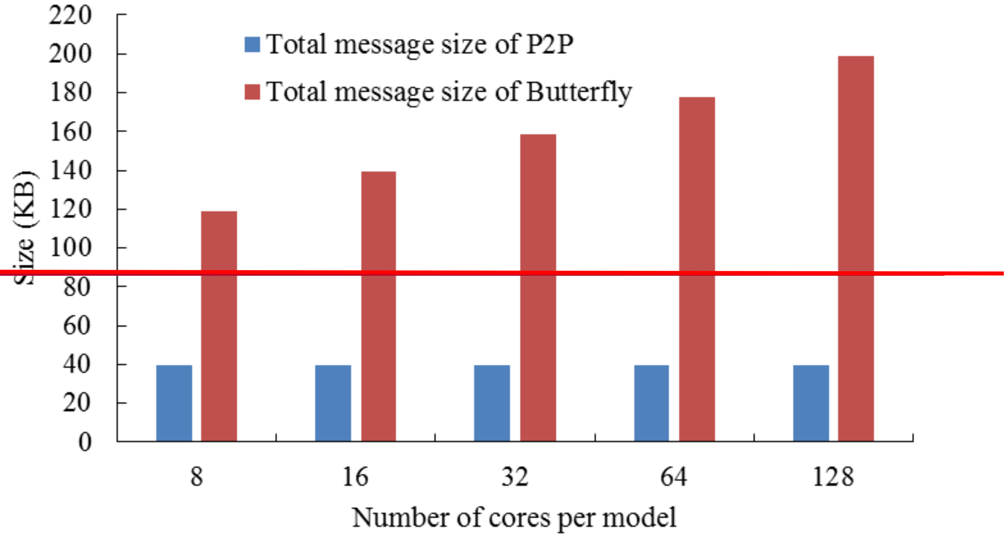

Sender
Source parallel decomposition
Process mapping
Source parallel decomposition
Butterfly Kernel
Target parallel decomposition
Process mapping
Target parallel decomposition
Receiver

Figure 7. 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.



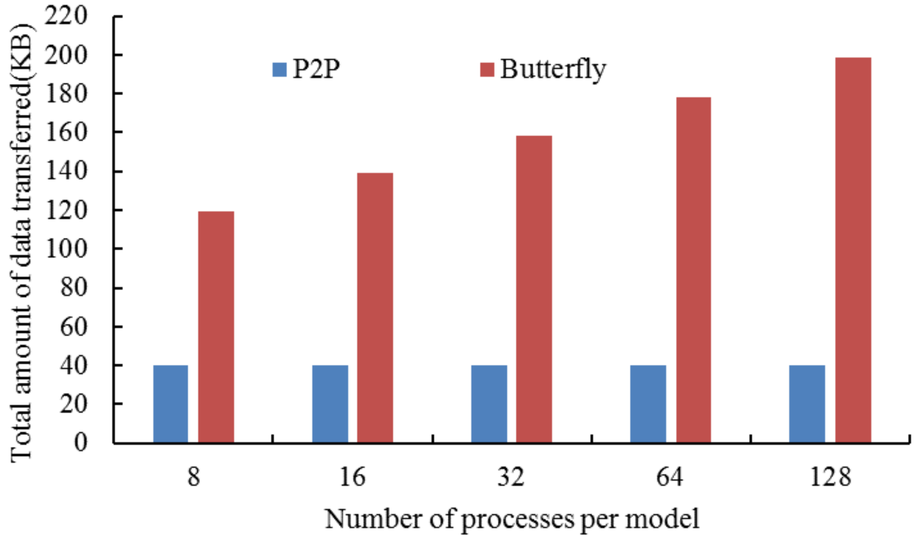
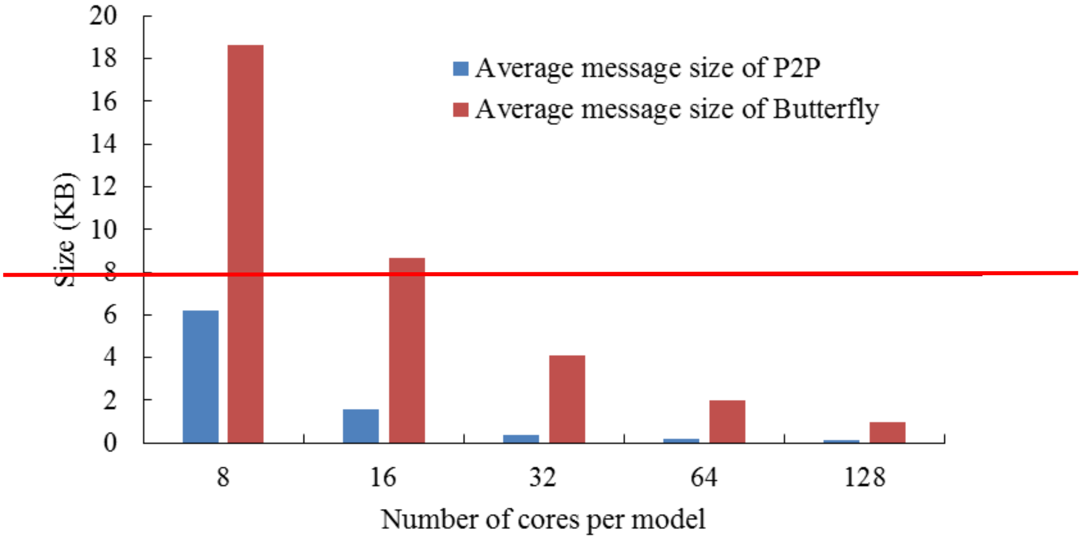


Figure 8. Total amount of data ~~message size~~ transferred by P2P implementation and butterfly implementation (y-axis) in GAMIL2-CLM3, when varying the number of processes per model~~cores used by each model~~ (x-axis). The experimental setup is similar to that shown in Fig. 1.



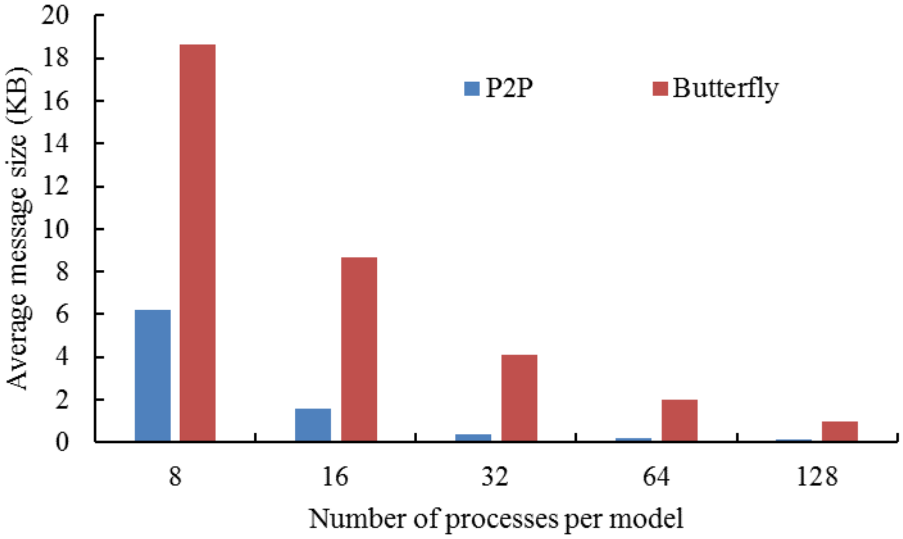
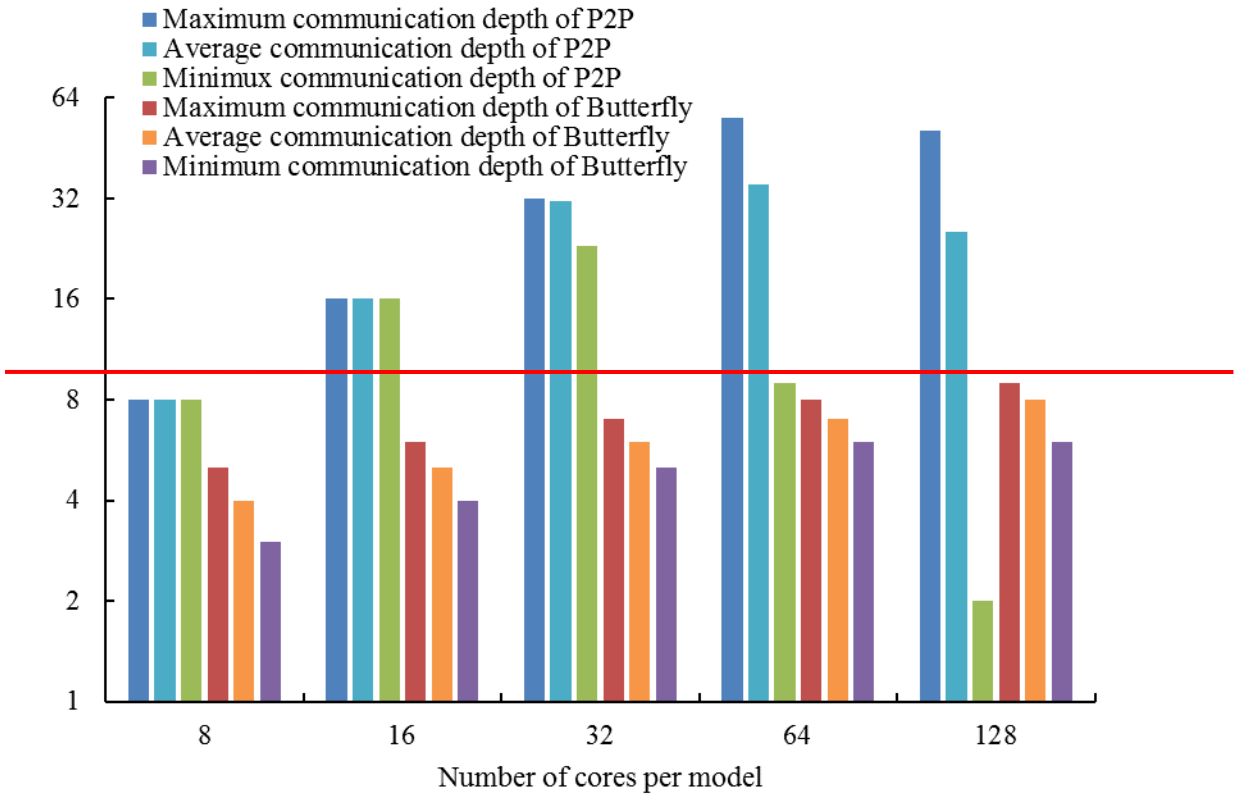


Figure 9. Average message size transferred by P2P implementation and butterfly implementation ~~-~~(y-axis) in GAMIL2-CLM3, when varying the number of processes per model~~cores used by each model~~ (x-axis). The experimental setup is similar to that shown in Fig. 1.



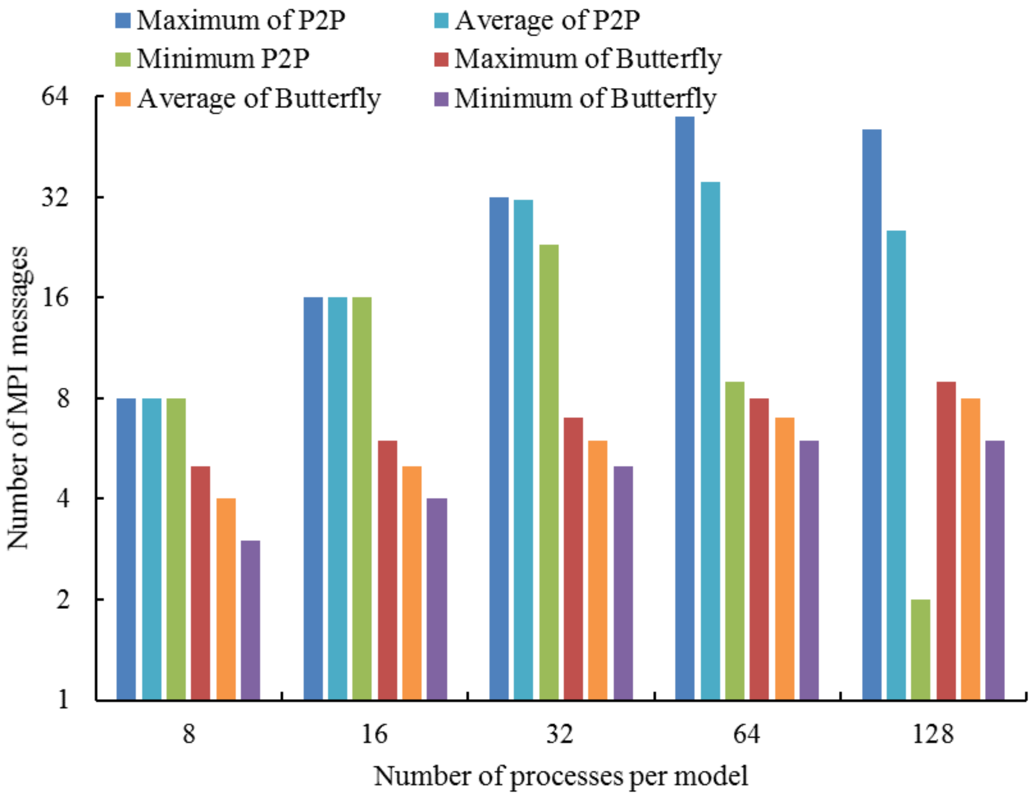
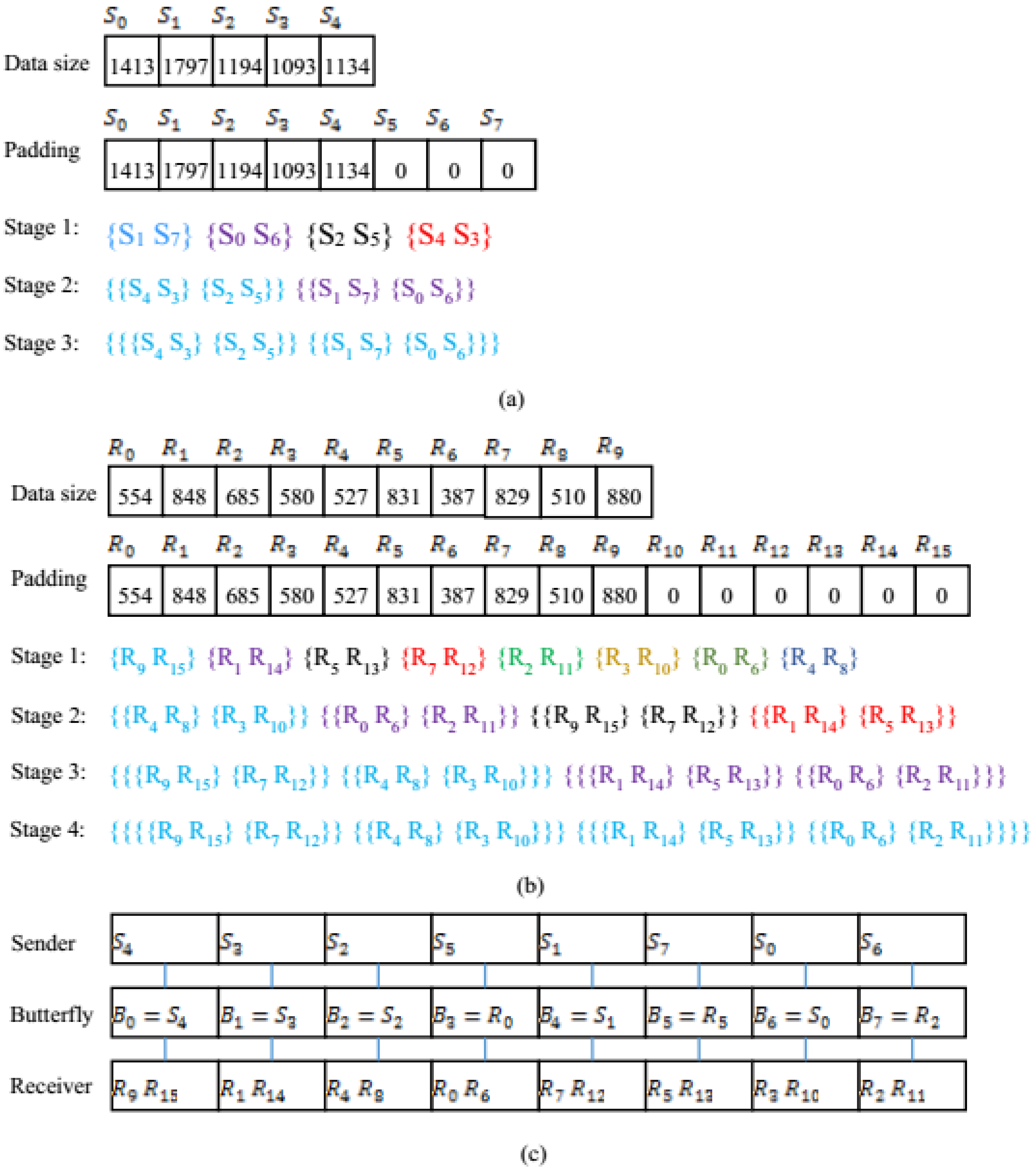


Figure 10. Maximum number of MPI messages~~communication depth~~, average number of MPI messages ~~communication depth~~ and minimum MPI messages ~~communication depth~~ in P2P

implementation and butterfly implementation (y-axis), when varying the number of processes per model~~cores used by each model~~ (x-axis) in GAMIL2-CLM3. The experimental setup is similar to that shown in Fig. 1.



Data size
S0 S1 S2 S3 S4
1413 1797 1194 1093 1134
Padding
S0 S1 S2 S3 S4 S5 S6 S7
1413 1797 1194 1093 1134 0 0 0
Stage 1: {S1 S7} {S0 S6} {S2 S5} {S4 S3}
Stage 2: {{S4 S3} {S2 S5}} {{S1 S7} {S0 S6}}
Stage 3: {{{S4 S3} {S2 S5}} {{S1 S7} {S0 S6}}}
(a)
Data size
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9
554 848 685 580 527 831 387 829 510 880
Padding
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
554 848 685 580 527 831 387 829 510 880 0 0 0 0 0 0
Stage 1: {R9 R15} {R1 R14} {R5 R13} {R7 R12} {R2 R11} {R3 R10} {R0 R6} {R4 R8}
Stage 2: {{R4 R8} {R3 R10}} {{R0 R6} {R2 R11}} {{R9 R15} {R7 R12}} {{R1 R14} {R5 R13}}
Stage 3: {{{R9 R15} {R7 R12}} {{R4 R8} {R3 R10}}} {{{R1 R14} {R5 R13}} {{R0 R6} {R2 R11}}}
Stage 4: {{{{R9 R15} {R7 R12}} {{R4 R8} {R3 R10}}} {{{R1 R14} {R5 R13}} {{R0 R6} {R2 R11}}}}
(b)
Sender
S4 S3 S2 S5 S1 S7 S0 S6
Butterfly
B0 = S4 B1 = S3 B2 = S2 B3 = R0 B4 = S1 B5 = R5 B6 = S0 B7 = R2
Receiver
R9 R15 R1 R14 R4 R8 R0 R6 R7 R12 R5 R13 R3 R10 R2 R11
(c)

Figure 11. An example of process mappings, given that the sender has 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 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 onto the processes of the butterfly kernel.

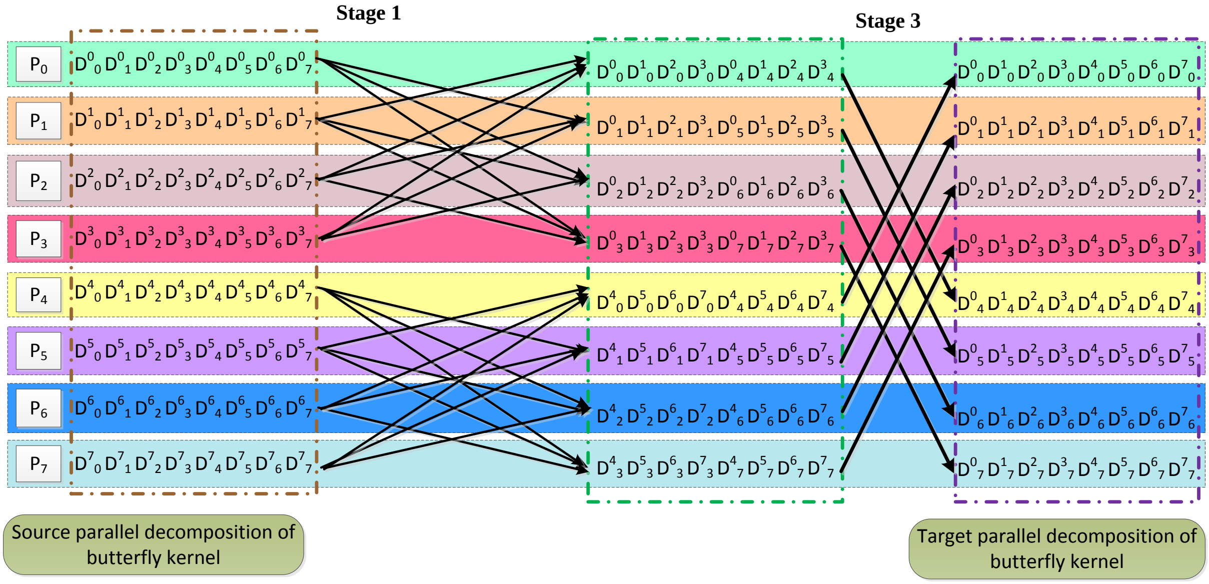

Stage 1
Stage 3
P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7
$D^0_0 D^0_1 D^0_2 D^0_3 D^0_4 D^0_5 D^0_6 D^0_7$
$D^1_0 D^1_1 D^1_2 D^1_3 D^1_4 D^1_5 D^1_6 D^1_7$
$D^2_0 D^2_1 D^2_2 D^2_3 D^2_4 D^2_5 D^2_6 D^2_7$
$D^3_0 D^3_1 D^3_2 D^3_3 D^3_4 D^3_5 D^3_6 D^3_7$
$D^4_0 D^4_1 D^4_2 D^4_3 D^4_4 D^4_5 D^4_6 D^4_7$
$D^5_0 D^5_1 D^5_2 D^5_3 D^5_4 D^5_5 D^5_6 D^5_7$
$D^6_0 D^6_1 D^6_2 D^6_3 D^6_4 D^6_5 D^6_6 D^6_7$
$D^7_0 D^7_1 D^7_2 D^7_3 D^7_4 D^7_5 D^7_6 D^7_7$
$D^0_0 D^1_0 D^2_0 D^3_0 D^0_4 D^1_4 D^2_4 D^3_4$
$D^0_1 D^1_1 D^2_1 D^3_1 D^0_5 D^1_5 D^2_5 D^3_5$
$D^0_2 D^1_2 D^2_2 D^3_2 D^0_6 D^1_6 D^2_6 D^3_6$
$D^0_3 D^1_3 D^2_3 D^3_3 D^0_7 D^1_7 D^2_7 D^3_7$
$D^4_0 D^5_0 D^6_0 D^7_0 D^4_4 D^5_4 D^6_4 D^7_4$
$D^4_1 D^5_1 D^6_1 D^7_1 D^4_5 D^5_5 D^6_5 D^7_5$
$D^4_2 D^5_2 D^6_2 D^7_2 D^4_6 D^5_6 D^6_6 D^7_6$
$D^4_3 D^5_3 D^6_3 D^7_3 D^4_7 D^5_7 D^6_7 D^7_7$
$D^0_0 D^1_0 D^2_0 D^3_0 D^4_0 D^5_0 D^6_0 D^7_0$
$D^0_1 D^1_1 D^2_1 D^3_1 D^4_1 D^5_1 D^6_1 D^7_1$
$D^0_2 D^1_2 D^2_2 D^3_2 D^4_2 D^5_2 D^6_2 D^7_2$
$D^0_3 D^1_3 D^2_3 D^3_3 D^4_3 D^5_3 D^6_3 D^7_3$
$D^0_4 D^1_4 D^2_4 D^3_4 D^4_4 D^5_4 D^6_4 D^7_4$
$D^0_5 D^1_5 D^2_5 D^3_5 D^4_5 D^5_5 D^6_5 D^7_5$
$D^0_6 D^1_6 D^2_6 D^3_6 D^4_6 D^5_6 D^6_6 D^7_6$
$D^0_7 D^1_7 D^2_7 D^3_7 D^4_7 D^5_7 D^6_7 D^7_7$
Source parallel decomposition of butterfly kernel
Target parallel decomposition of butterfly kernel

Figure 12. An example of the adaptive data transfer library with eight processes, where Stage 2 of the butterfly implementation is skipped and replaced by P2P communication of three MPI messages per process.

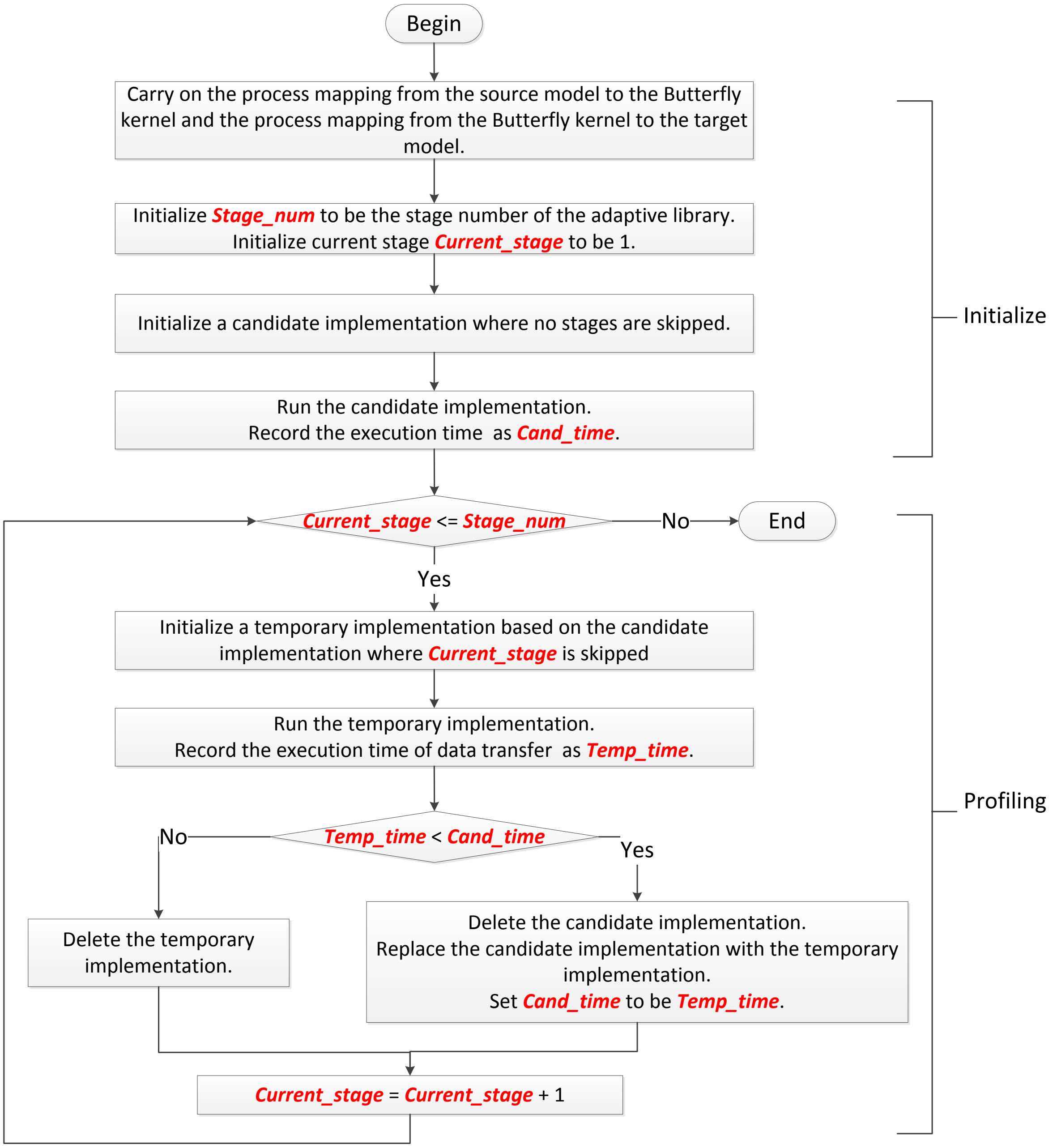
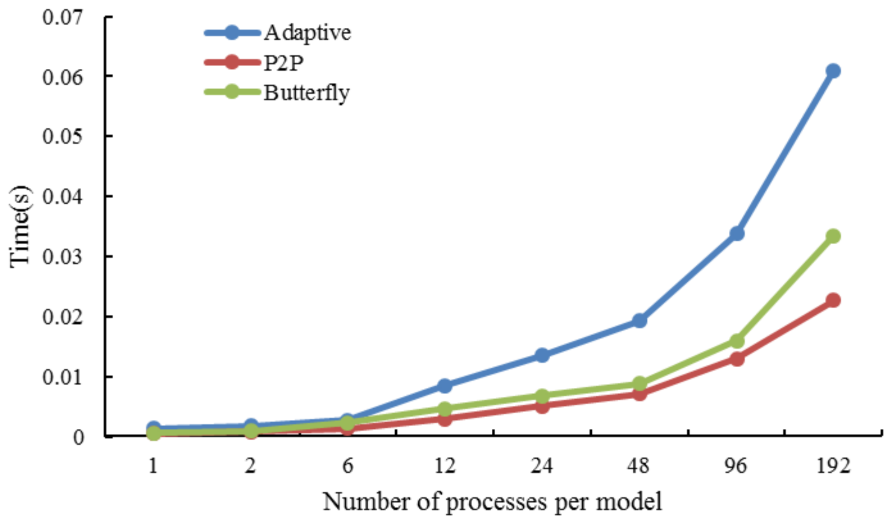

Begin
Carry on the process mapping from the source model to the Butterfly kernel and the process mapping from the Butterfly kernel to the target model.
Initialize Stage_num to be the stage number of the adaptive library.
Initialize current stage Current_stage to be 1.
Initialize a candidate implementation where no stages are skipped.
Run the candidate implementation.
Record the execution time as Cand_time.
Initialize
Current_stage <= Stage_num
No
End
Yes
Initialize a temporary implementation based on the candidate implementation where Current_stage is skipped
Run the temporary implementation.
Record the execution time of data transfer as Temp_time.
Profiling
Temp_time < Cand_time
No
Yes
Delete the temporary implementation.
Delete the candidate implementation.
Replace the candidate implementation with the temporary implementation.
Set Cand_time to be Temp_time.
Current_stage = Current_stage + 1

Figure 13. A flowchart for determining an appropriate implementation of the adaptive data transfer library.



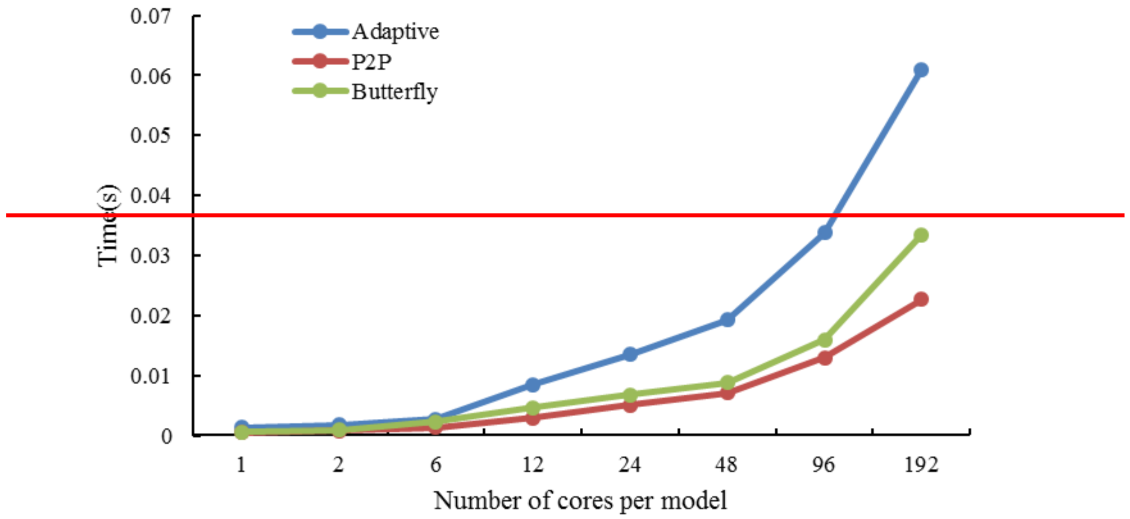


Figure 14. Initialization time (y-axis) of one data transfer between two toy models using a rectangular grid (of 192×96 grid points) when varying the number of processes per model~~cores used by each toy model~~ (x-axis). There are 10 2-D coupling fields transferred from the source toy model to the target toy model. In each test, all processes of the sender in the P2P implementation have the same number of MPI messages~~communication depth~~. If the number of processes~~cores~~ per model~~ used~~ is less than 24, the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is equal to the number of processes~~cores~~ per model; otherwise, the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is 24. The parallel decompositions of the sender and the receiver for a given average~~setting of~~ number of MPI messages~~communication depth~~ are generated by Algorithm 1.

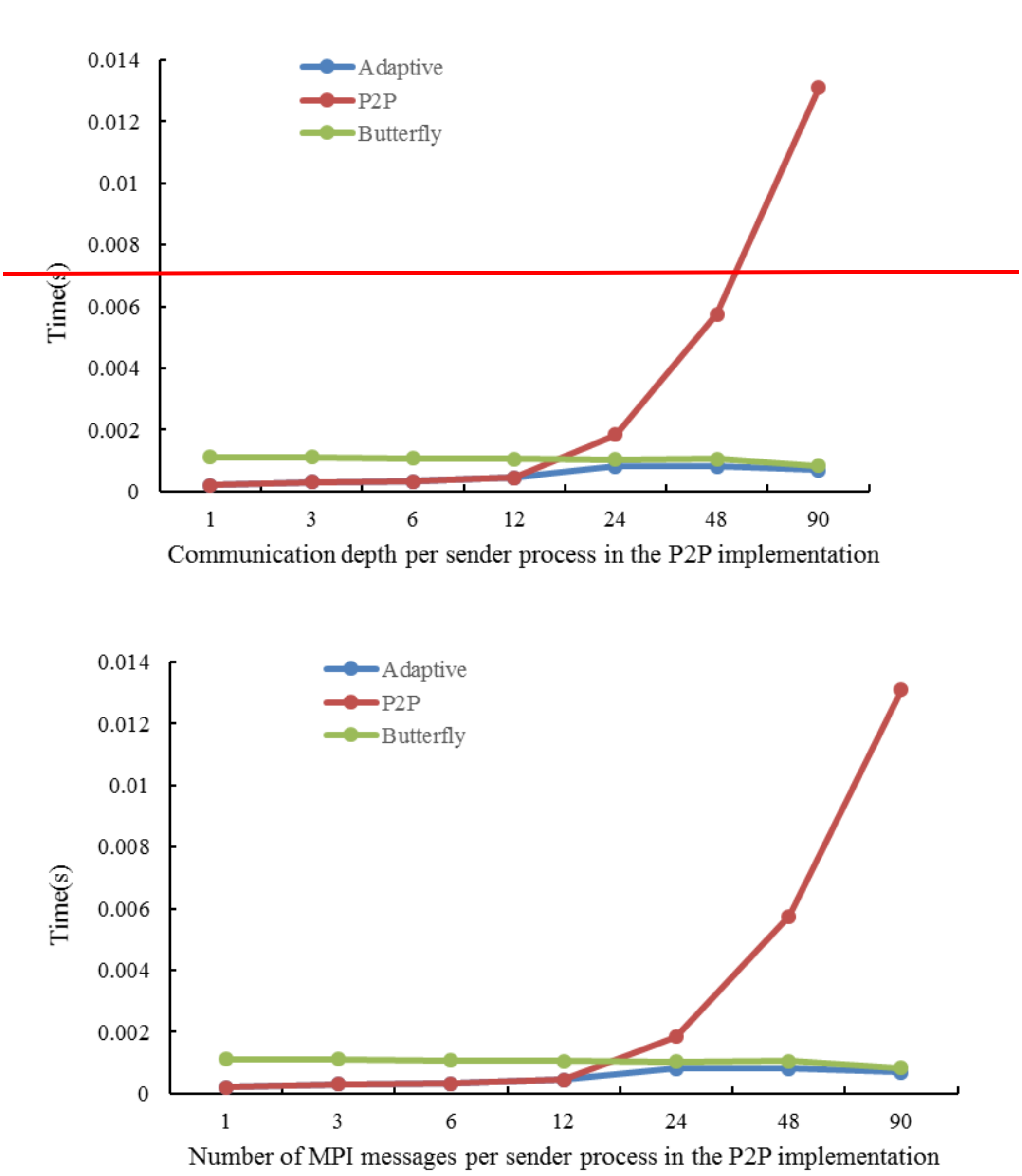
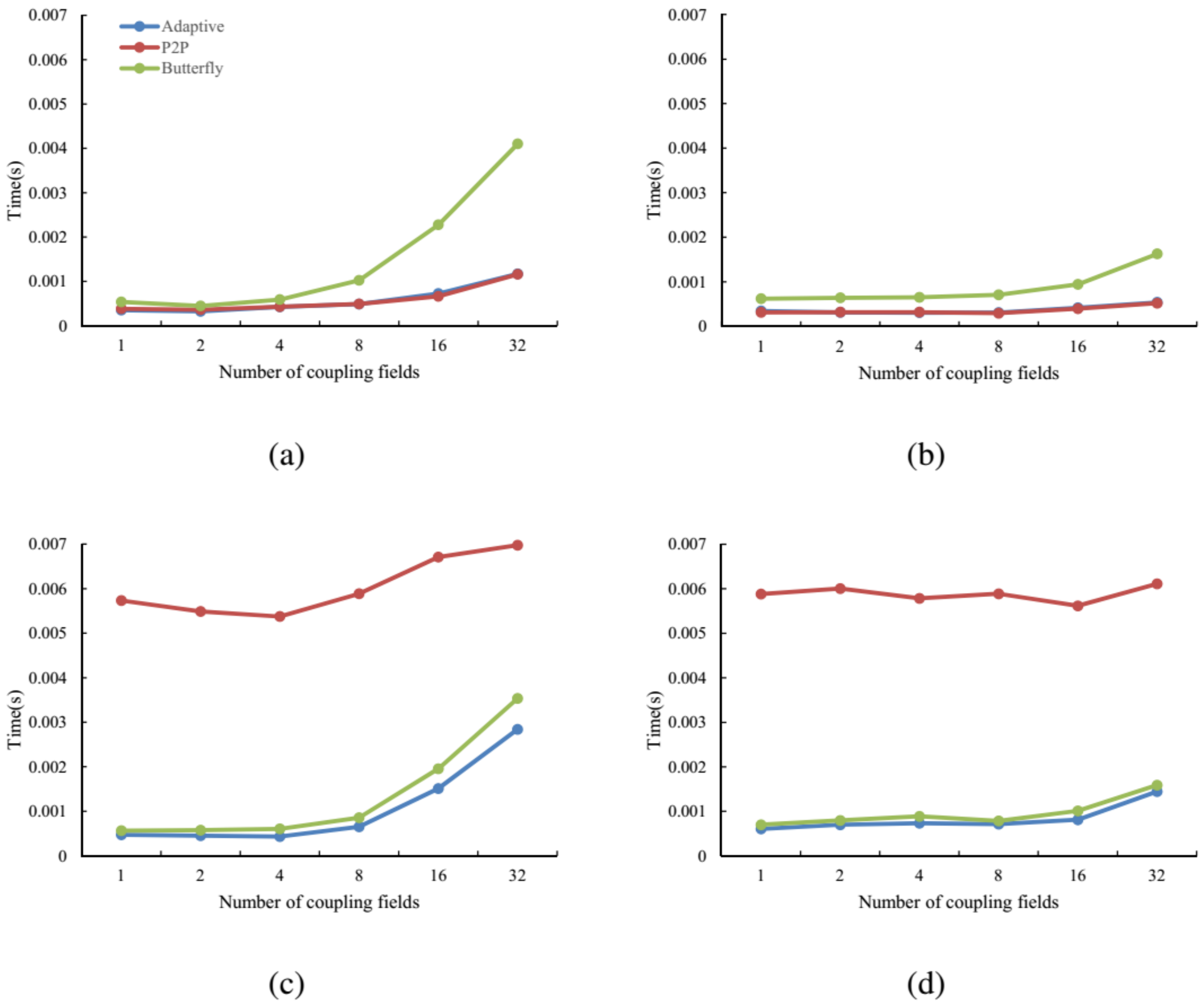


Figure 15. Average execution time (y-axis) of one data transfer between two toy models with the same rectangular grid (of 192×480 grid points) when varying the number of MPI messages~~communication depth~~ per sender process in the P2P implementation (x-axis). Each toy model is run with 1024 processes~~cores~~. There are 10 2-D coupling fields transferred from the source toy model to the target toy model.



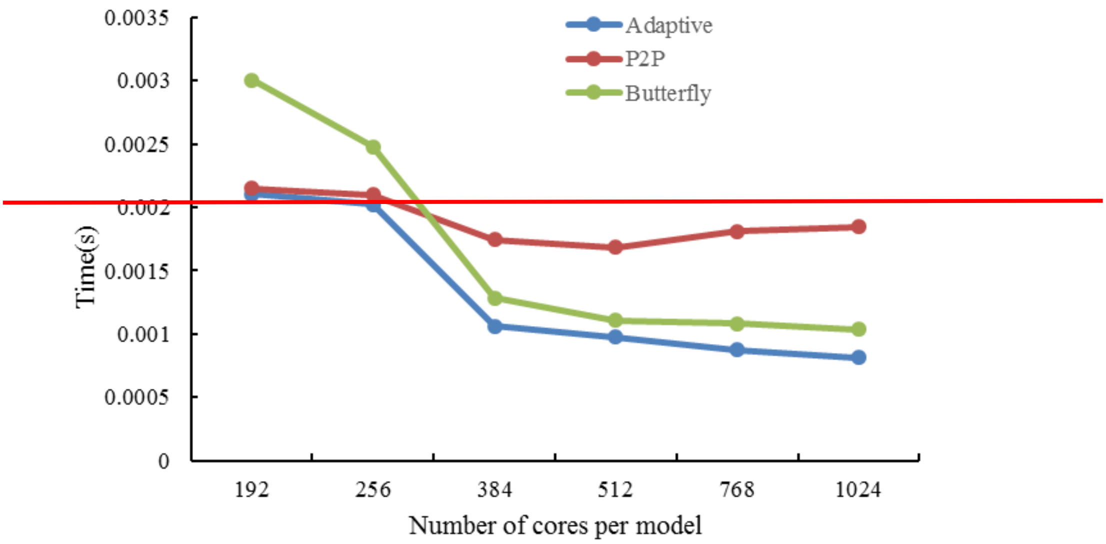
(a)

(b)

(c)

(d)

Figure 16. Average execution time (y-axis) of one data transfer between two toy models with the same rectangular grid (of 192×480 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 256 processes~~cores~~, and the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is 12. In simulation (b), each toy model is run with 1024 processes~~cores~~, and the number of MPI messages~~communication depth~~ per sender process is in the P2P implementation 12. In simulation (c), each toy model is run with 256 processes~~cores~~, and the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is 48. In simulation (d), each toy model is run with 1024 ~~cores (or~~ processes~~)~~, and the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is 48.



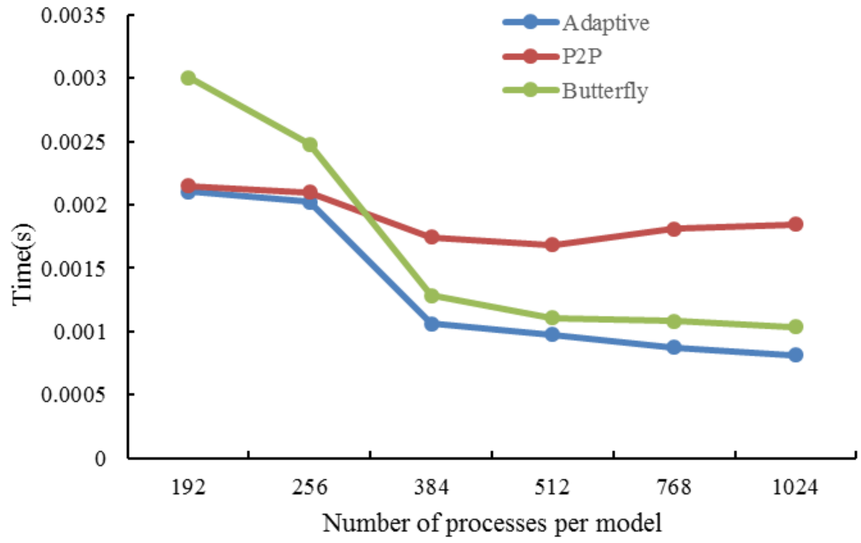


Figure 17. Average execution time (y-axis) of one data transfer between two toy models with the same rectangular grid (of 192×480 grid points) when varying the number of processes per model ~~cores used by each toy model~~ (x-axis). There are 10 2-D coupling fields transferred from the source toy model to the target toy model. In each test, the number of MPI messages~~communication depth~~ per sender process in the P2P implementation is 24.

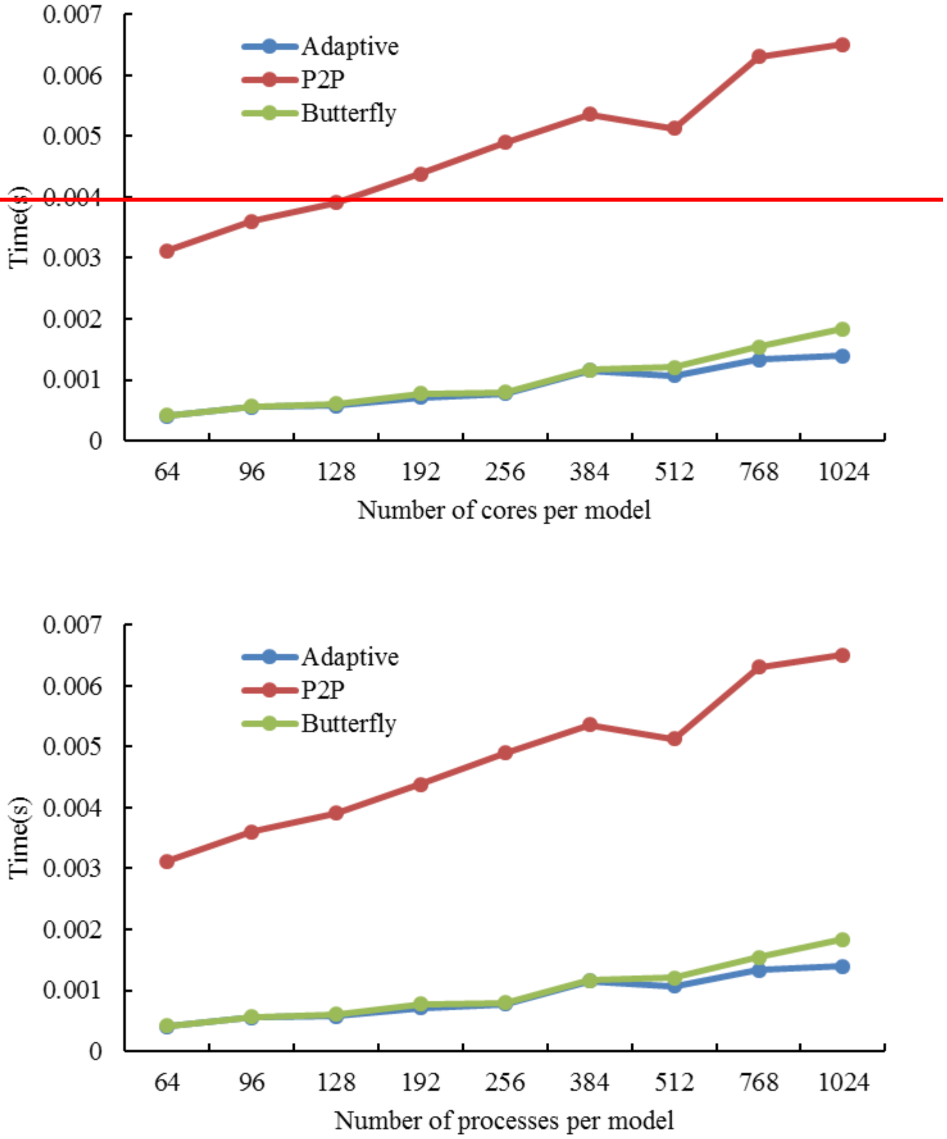
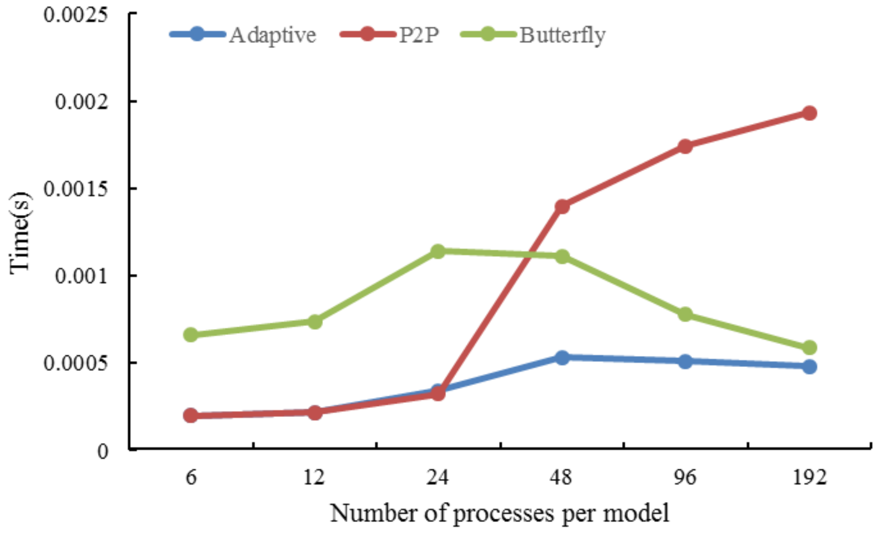


Figure 18. Average execution time (y-axis) of one data transfer between two toy models. In this evaluation, each process (running on a unique processor core) of the toy models have 96 grid points, while different processes have different number of MPI messages~~communication depth~~ and different message sizes in the P2P implementation. The number of coupling fields transferred is set to 20.



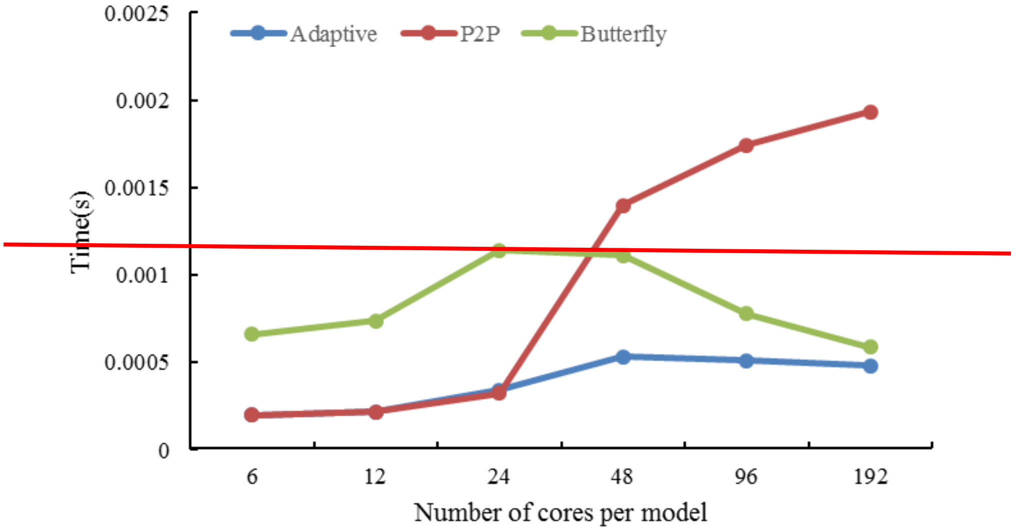
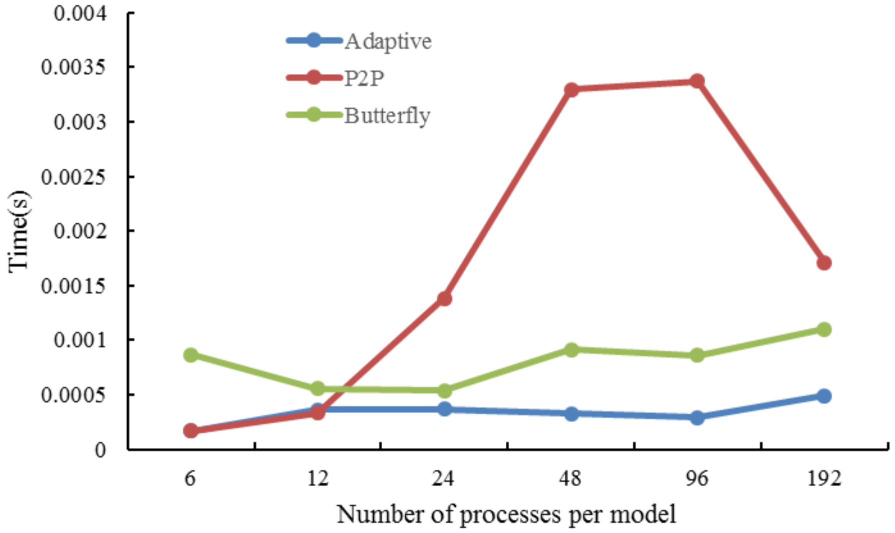


Figure 19. Average execution time (y-axis) of one data transfer between the land surface model CLM4 and the coupler CPL7 in CESM when varying the number of processes per model ~~cores used by each model~~ (x-axis): 32 coupling fields on the CLM horizontal grid (the grid size is 144×96=13824) are transferred from the land surface model CLM4 to the coupler CPL7. The performance results of the P2P implementation are obtained through running the adaptive data transfer library ~~when~~forcing it to completely switch~~es~~ to the original P2P implementation.



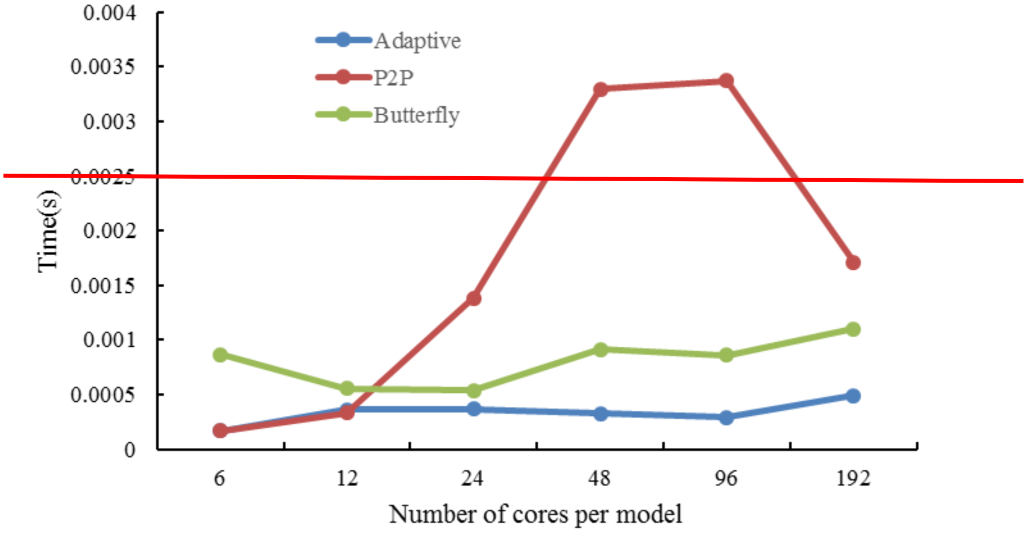
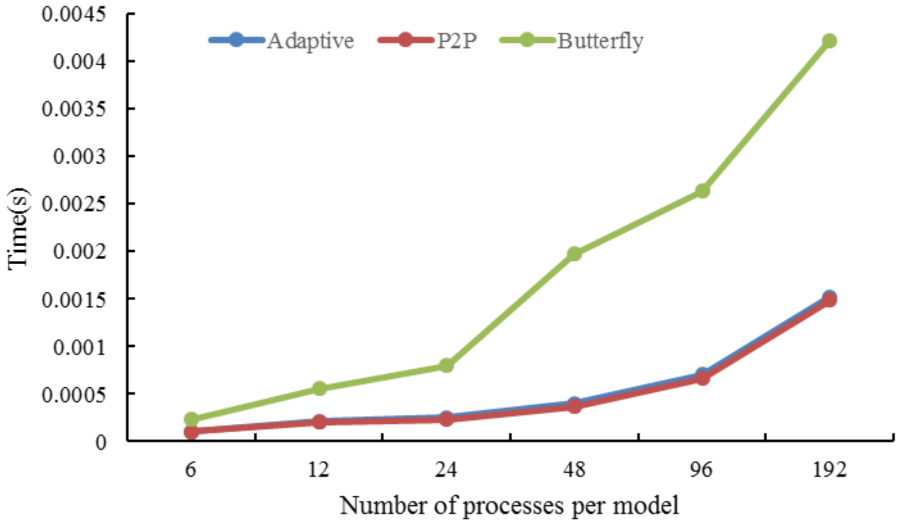


Figure 20. Average execution time (y-axis) of one data transfer between the atmosphere model GAMIL2 and the land surface model CLM3 in GAMIL2-CLM3 when varying the number of processes per model ~~cores used by each model~~ (x-axis): 14 coupling fields on the GAMIL2 horizontal grid (the grid size is 128×60=7680) are transferred from the land surface model CLM3 to the atmosphere model GAMIL2.



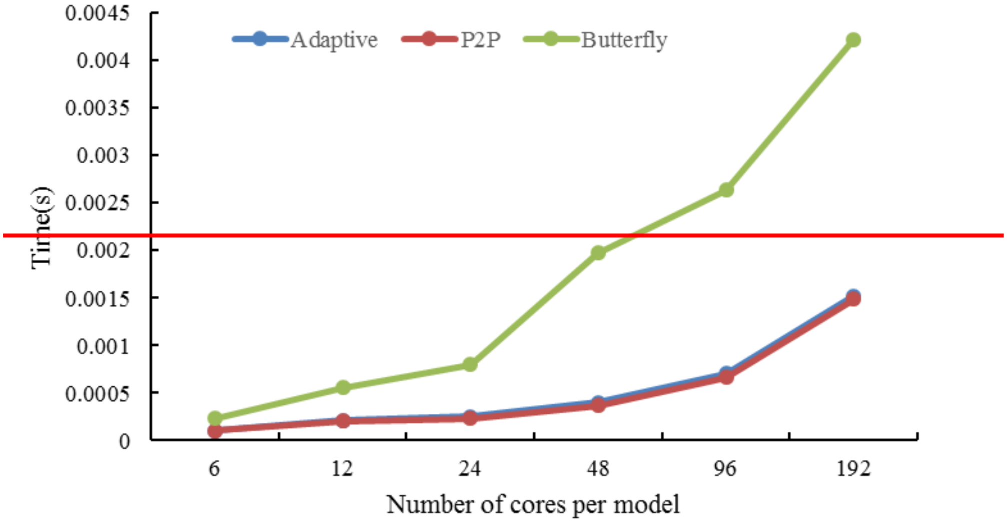
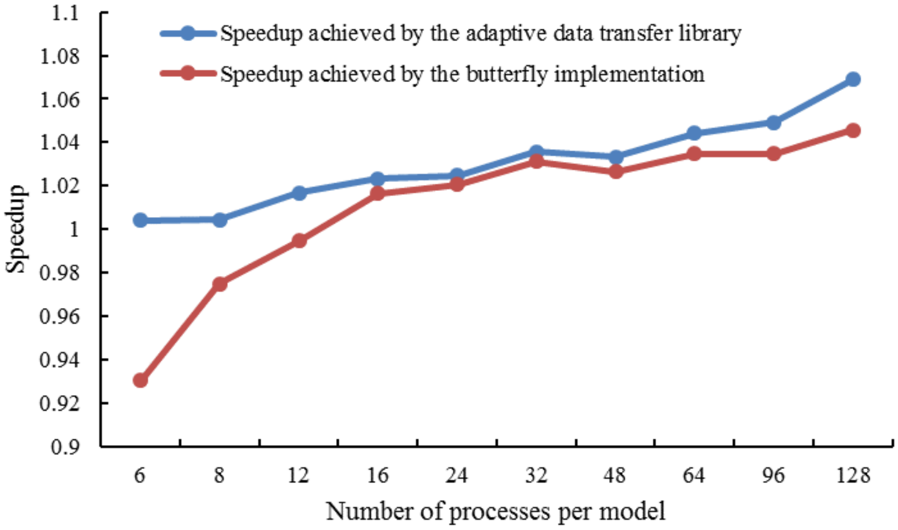


Figure 21. Average execution time (y-axis) of one data rearrangement for the parallel interpolation from the atmosphere grid (the grid size is 144×96=13824) to the ocean grid (the grid size is 320×384=122880) in CESM when varying the number of processes per model ~~cores used by each model~~ (x-axis).



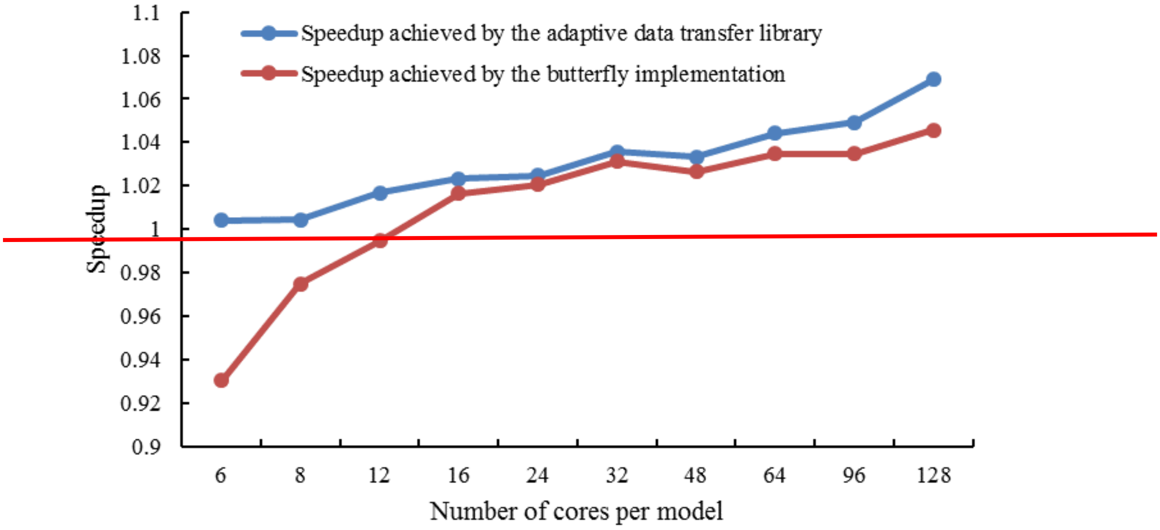


Figure 22. Performance improvement with respect to the whole model time for~~imporvement of~~ the coupled model GAMIL2-CLM3 achieved by the butterfly implementation and the adaptive data transfer library, ~~with the whole model time of GAMIL2-CLM3~~ using the P2P implementation as the baseline.