Development of the data buffer holding time-series data across multiple applications

Jingde Zhou

Graduate School of Informatics, Kyoto University

1. Introduction

Cross-reference simulation is a calculation model combining multiple parallel simulation codes when some simulation codes need to read the data calculated by other simulation codes to do their own calculation. The data communication between different simulation codes is the most important part of a cross-reference simulation. In many research fields, cross-reference simulation is used to study complicated phenomena involving multiple simulations which may have substantial differences in the spatial and temporal scales.

To execute cross-reference simulation efficiently, a cross-reference simulation framework called Code-To-Code-Adapter (CoToCoA)[1] is being developed based on Message Passing Interface (MPI)[2]. CoToCoA is a framework to connect a requester application to multiple worker applications through a coupler application. CoToCoA can be used to execute cross-reference simulation in an efficient and smooth way called strong cross-reference simulation. Different from other cross-reference simulation frameworks like preCICE[3], CoToCoA mainly focuses on the data communication between different simulation codes. The user only needs to add minimal modifications to the simulation codes to implement the data communication when other couplers may require several sophisticated settings. Therefore, CoToCoA users can easily couple the simulation codes which are developed by other developers.

There is one main simulation code in many cross-reference simulations. To keep the efficiency and stability of the whole cross-reference simulation, the main simulation code's overhead brought by data communication should be as little as possible. One-sided communication is a technology that allows one process to read or write the memory of another process without its response. In CoToCoA, MPI Remote Memory Access (RMA) is used to implement one-sided communication. With MPI RMA, other simulation codes (usually executed as the workers) can directly read the data calculated by the main simulation code (usually executed as the requester). Then the main simulation code's overhead comes to a minimal level.

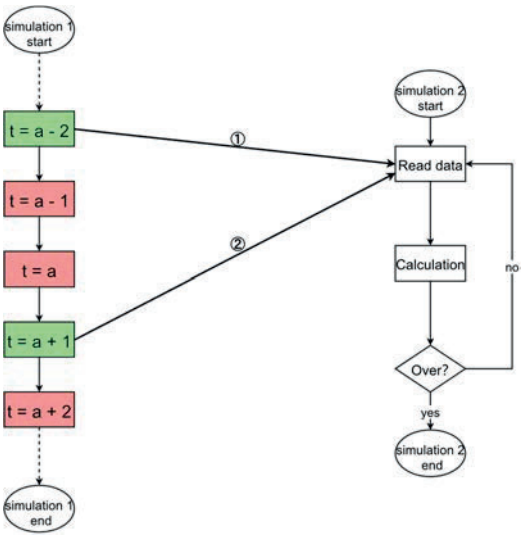

Fig. 1 The communication loss in continual MPI RMA calls

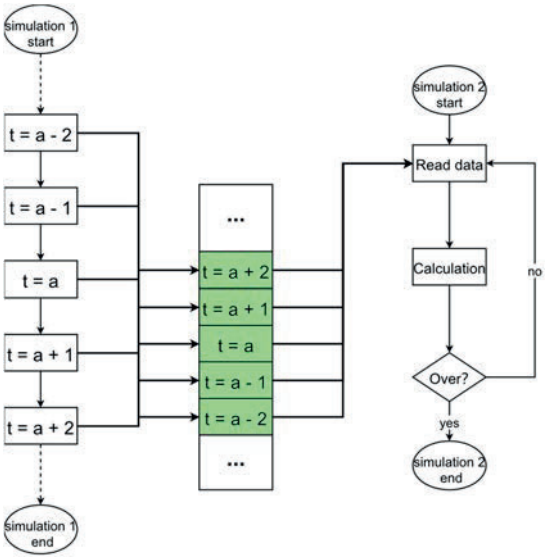

Fig. 2 The utilization of data buffer can avoid the communication loss in continual MPI RMA calls

When one-sided communication is utilized as the communication method in CoToCoA, a large amount of data may be lost due to the different execution speeds. An example is shown in **Fig. 1**. At some point, simulation 2 reads the data from simulation 1 when the timestamp of simulation 1 is *a – 2*. to do its calculation. Next time, simulation 2 reads the data when the timestamp of simulation 1 is *a + 1*. The data at time stamp *a - 1* and time stamp *a* are lost unavoidably.

2. Data buffer holding time-series data across multiple applications

2.1 Create the specific data buffer

This paper develops a new CoToCoA function utilizing a specific data buffer to avoid frequent communication loss. Meanwhile, each process of each worker can read a particular part of a *n*-dimension data when CoToCoA user specifies the start position and end position of the data. In addition, each process of each worker reads multiple time steps data instead of only one time step data in one communication call. The working principle of this function is shown in **Fig. 2**. A specific data buffer temporarily saves the calculated data, then the data will not be overwritten in the next timestep. The workers read data from the data buffer. Then no data will be lost as long as the data buffer is not full.

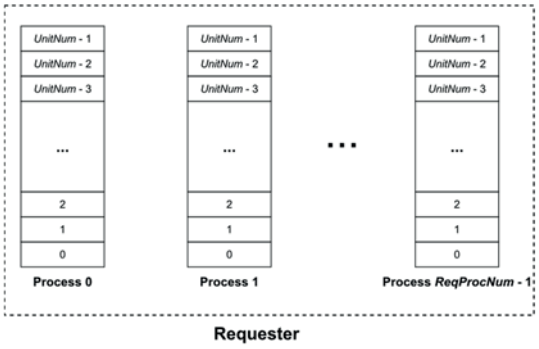

Fig. 3 Construction of the specific data buffer

In the new function, a specific data buffer is created in the memory of each process of the requester and read by the workers. Each unit of this data buffer saves the data calculated in each timestep.

Fig. 3 shows the construction of the data buffer, *UnitNum* is the number of buffer units, *ReqProcNum* indicates the number of processes of the requester. The value of *UnitNum* is determined by the user-specified buffer size and the size of data at each time step. The requester sequentially saves calculated data in the data buffer after it does its calculation in each time step. If the data buffer is full, the new data will overwrite the oldest data.

2.2 Create derived datatype

In this new function, the non-contiguous data communication is implemented by MPI's derived datatype. Two derived datatype array is created in each process of the workers. $Datatype\_req_{i,j}$ denotes the layout of data in the data buffer for the data communication between the requester's

process *i* and this worker's process *j*. *Datatype_wrk*$_{i,j}$ denotes the layout of data in the receive buffer for requester's process *i* and worker's process *j*.

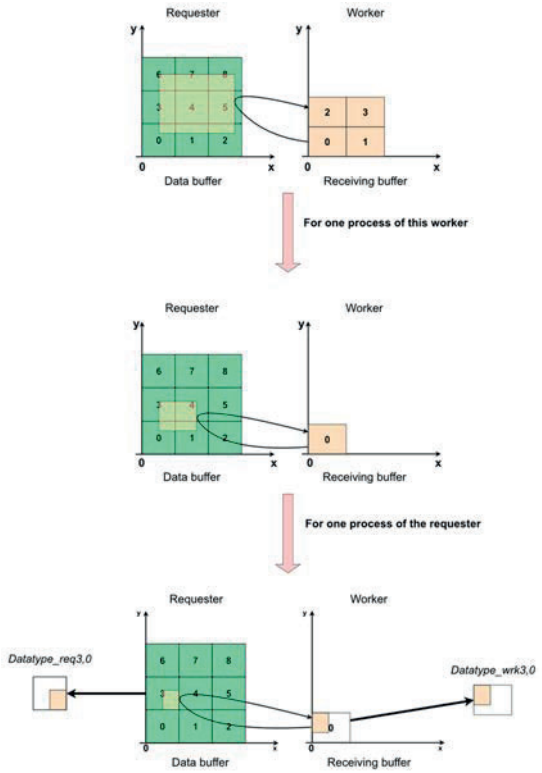

Fig. 4 The use of MPI's derived datatype

Fig.4 is an example illustrating the layout of the derived datatype. In this example, there is only one worker. There are nine processes of the requester and four processes of the worker. Each process of the requester is in charge of one part of the calculation. For each process of the worker, it needs to read different layouts of data from different process of the requester. For example, the process *0* of the worker reads *Datatype_req*$_{3,0}$ layout of data from the data buffer in process *3* of the requester, and saves it with *Datatype_wrk*$_{3,0}$ layout of data in its receiving buffer.

2.3 Read data from requester

To minimize the data communication overhead of the workers, the number of communication calls should be reduced as much as possible. In the new function, the workers can read data across multiple time steps by only one (sometimes two) communication call.

There are two Execution Mode in the new function. In Execution Mode 1, the requester will not suspend when the data buffer is full. The requester only needs to copy the calculated data to the data buffer at the price of communication loss (Communication loss still happens when the data buffer is full). In Execution Mode 2, the requester will wait for the workers when the data buffer is full. No communication loss will happen but the requester may sometimes suspend.

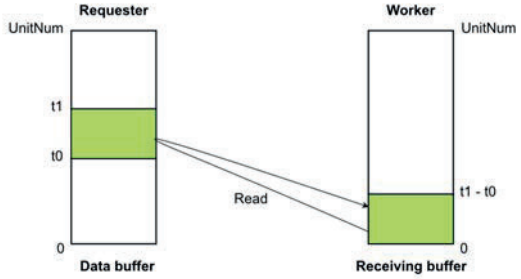

Fig. 5 Worker reads data from the data buffer (situation 1)

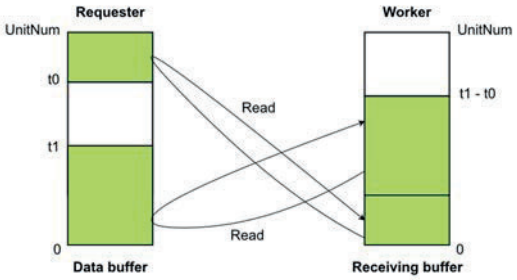

Fig.6 Worker reads data from the data buffer (situation 2)

Both in Execution Mode 1 and 2:

In general, workers read data from the data buffer by only one time as shown in **Fig. 5**.

When a worker's process reads data across the boundary of the data buffer, it needs to read two times. An example is shown in **Fig. 6**.

Extra issue only in Execution Mode 2:

In Execution Mode 2, some data may not be read by the workers because of the imbalanced execution speed. A CoToCoA routine *CTCAW_get_timestep* is provided to get the exact time step of current data.

The workers may read the data mixed with different time steps due to the overwriting. An example is as shown in **Fig. 7**. During the data communication, The timestamp of the requester is increased from t_1*_old* to t_1*_new*. The received data from *0* to t_1*_new* – t_0, which is the yellow part in the figure, may be mixed with different time steps data unavoidably. Though overwriting is inevitable, CoToCoA users can use *CTCAW_get_overwrite_flag* to distinguish if the current timestamp data is mixed or not.

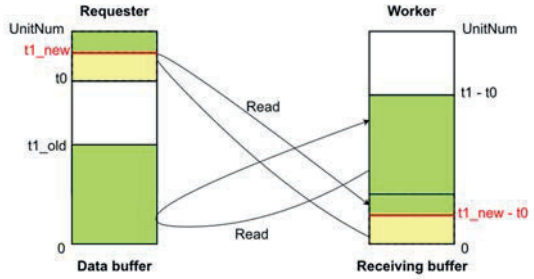

Fig. 7 Worker reads data from the data buffer (situation 3)

3. Evaluation result

An evaluation is implemented to measure the performance of three different data communication methods on the Vector Engine (VE) of SQUID. They are traditional two-sided communication, ordinary one-sided communication without data buffer and the new function. In this evaluation, data communication is between two processes distributed in different VEs. The Size of the data buffer is 4GB. The total timestep is 100*N*. Process 1 calculates 900 * 900 2-dimension integer data array in each timestep. Meanwhile, Process 2 reads 600 * 600 2-dimension integer data array from Coordinates (150, 150) to Coordinates (750, 750) in the data buffer until there is no useful data in the data buffer. This program is executed in Execution Mode 1.

The evaluation result on VE of SQUID is as shown in **Fig. 8**. According to the evaluation result, when $N < 7$, the new function is 12.6% faster than ordinary one-sided communication without data buffer. However, when $N >= 7$, one-sided communication with data buffer is 4.5% slower than one-sided communication without data buffer. The performance of the new function decreased rapidly. The reason may be that there is a limitation that will be broken through when $N >= 7$ in one one-sided communication call. Then the data communication is carried out in an implicit and inefficient way.

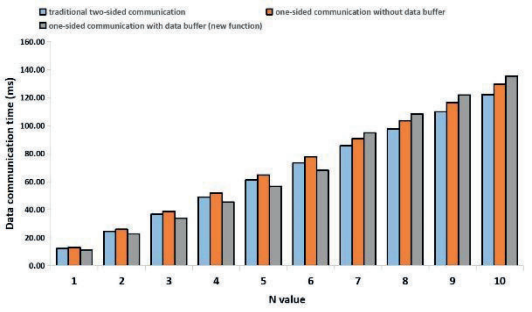

Fig. 8 Data communication time in the evaluation

References

[1] Keiichiro Fukazawa, Yuto Katoh, Takeshi Nanri, and Yohei Miyake. Application of cross-reference framework cotocoa to macro- and micro-scale simulations of planetary

magnetospheres. In *2019 Seventh International Symposium on Computing and Networking Workshops(CANDARW)*, pages 121–124, 2019

[2] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*, June 2022.

[3] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. precice–a fully parallel library for multi-physics surface coupling. *Computers & Fluids*, 141:250–258, 2016.