Toward a General I/O Arbitration Framework for netCDF based Big Data Processing

Jianwei Liao, Balazs Gerofi, Guo-Yuan Lien, Seiya Nishizawa,
Takemasa Miyoshi, Hirofumi Tomita and Yutaka Ishikawa

RIKEN Advanced Institute for Computational Science, JAPAN

Abstract. On the verge of the convergence between high performance computing (HPC) and Big Data processing, it has become increasingly prevalent to deploy large-scale data analytics workloads on high-end supercomputers. Such applications often come in the form of complex workflows with various different components, assimilating data from scientific simulations as well as from measurements streamed from sensor networks, such as radars and satellites. For example, as part of the next generation flagship (post-K) supercomputer project of Japan, RIKEN is investigating the feasibility of a highly accurate weather forecasting system that would provide a real-time outlook for severe guerrilla rainstorms. One of the main performance bottlenecks of this application is the lack of efficient communication among workflow components, which currently takes place over the parallel file system.
In this paper, we present an initial study of a direct communication framework designed for complex workflows that eliminates unnecessary file I/O among components. Specifically, we propose an I/O arbitrator layer that provides direct parallel data transfer among job components that rely on the netCDF interface for performing I/O operations, with only minimal modifications to application code. We present the design and an early evaluation of the framework on the K Computer using up to 4800 nodes running RIKEN's experimental weather forecasting workflow as a case study.

1 Introduction

With the accelerating convergence between high performance computing (HPC) and a new generation of Big Data technologies, high-end supercomputers are increasingly being leveraged for processing the unprecedented amount of data scientific simulations and sensor networks produce [4]. Consequently, the high-performance computing community has been heavily focusing on how to provide the appropriate execution environment for Big Data workloads on large scale HPC systems.

A motivating example, as well as our case study in this paper, is *SCALE-LETKF* [25], a complex weather forecasting application that is being developed at RIKEN. With the next generation Japanese flagship supercomputer (post-K) as its primary target platform, *SCALE-LETKF* is intended to provide high-resolution, real-time weather forecasting of severe guerrilla rainstorms

in Japan. Similar to other operational weather forecasting applications, the SCALE-LETKF mainly consist of two components developed separately: a numerical weather prediction (NWP) model and a data assimilation system. The NWP model used here is the SCALE-LES (Scalable Computing for Advanced Library and Environment-LES [7]), which simulate the time evolution of the weather-related atmosphere and land/sea surfaces based on physical equations (hereafter "*Simulation*"). Meanwhile, the data assimilation method used here is the Local Ensemble Transform Kalman Filter (*LETKF* [6]), which assimilate observation data taken from the real world into the simulated state to produce a better initial condition for the model (hereafter "*Assimilation*"). The two components run in a cyclic way: after the simulation finishes, the data assimilation starts taking the results from the simulation as its input data, and after the data assimilation finishes, the simulation of the next cycle follows, depending on the results from the data assimilation. Additionally, since the observation data are required in each cycle, they need to be streamed directly into RIKEN's supercomputing facility when executing the workflow in real-time.

Both the simulation (*SCALE*) and data assimilation (*LETKF*) components in the current workflow rely on *netCDF* for I/O operations using the parallel file system. *netCDF* is a self-describing, portable, scalable, appendable and shareable file format, which is widely used to exchange array-oriented scientific data, such as grids and time-series [1]. Historically, the decision for file based data exchange was mainly driven by the fact that these models are being developed and maintained by independent research entities and it's been strongly desired not to modify either of the component models purely for the purpose of building a coupled forecasting system.

The prediction of guerrilla heavy rains, however, is a strictly time constrained procedure and we identified that file I/O based data transfer between the two components is one of the hindering factors for acquiring the needed realtimeness. A large number of coupling tools, targeting effective integration of separately developed models or applications, have been proposed [9], [15], [16], [23], nevertheless, all of them require numerous modifications to the applications.

Our main focus in this paper is to provide an I/O arbitration framework that can enable high-performance, direct data exchange among workflow components which process large amounts of data and use *netCDF* for their underlying data format. Furthermore, we seek to provide a solution that retains the original *netCDF* API and requires only minimal changes to existing application code. Specifically, this paper makes the following contributions:

- *General I/O Arbitration Middleware.* We propose a general I/O arbitration middleware, i.e., a software library that enables direct parallel data transfer among workflow components that utilize *netCDF* for their data representation. Our library is customizable through configuration files and requires only slight modifications to the source code of existing applications.
- *Support for Integration of Existing Models.* Our middleware benefits the integration of existing, separately developed models for solving complicated problems. Individual models or applications are usually developed to tackle

specific scientific issues and easy integration of existing models into complex workflows enables solving more intricate problems.
- *Accelerated Data Exchange in Coupled Systems.* Compared to file based data exchanging the proposed middleware adopts communication pattern-based optimization to efficiently support direct data transfer. It shortens data exchange time among the components so that rigid time constraints of real-time applications can be satisfied.

The remainder of the paper is structured as follows. Related work is described in Section 2. The design and implementation of the proposed middleware are explained in Section 3. Section 4 shows the evaluation methodology and discusses experimental results. At last, concluding remarks are given in Section 5.

2 Related work

In weather forecasting and geoscientific systems, individual models usually deal with analyzing a sinlge, specific phenomenon. On the other hand, a practical forecasting system takes various aspects into account, and thus it normally employs several models to achieve its final goal. This section introduces related work focusing on coupling existing models or applications, as well as on related work about conducting data transfer among the component models or applications in such systems.

①*Integration mechanisms for individual models.* The intricate global climate problems motivate researchers from different scientific disciplines to integrate existing multi-physics computation models or applications for exhaustive modeling by using a software framework or a coupled system [19]. The Model Coupling Toolkit (*MCT*) [15] is a library providing routines and datatypes for creating a coupled system, and it is mainly used in Community Climate System Model (*CCSM*) [16]. Hereafter, S. Valcke et al. [8] have summarized major coupling technologies used in Earth System Modeling, and their paper shows common features of the existing coupling approaches including the functionality to communicate and re-grid data.

The *OASIS coupler* is another related study (the latest version is *OASIS3*), which is able to process synchronized exchanges of coupling information generated by different components in a climate system, and the coupler mediates communication among the components [18]. But, the *OASIS coupler* has its own interface, and is not a solution for general cases. C. Armstrong et al. [17] have designed an approach to separate the code of models from the coupling infrastructure, but it does not provide coupling functions such as data transfer. However, it enables users to choose the underlying coupling functions from other couplers, such as the *OASIS* coupler. Besides, there are numerous existing middlewares for coupling specific models, such as *ESMF* [10], and *C-Coupler1* [11], which adopt similar integration schemes to the above mentioned solutions, but unfortunately they also require application modifications.

Moreover, G. Waston et al. [21] proposed the scheme to us parallel coupling tool for effectively integrating the existing programming and performance

tools, to benefit the development of parallel applications. Dorier et al. [24] have summarized several tools developed by themselves, which can flexibly couple simulations with visualization packages or analysis workflows.

②*Data transfer approaches in coupling or other large-scale systems.* Many integrated approaches employ file-based I/O to exchange data, since the data stored on the global file system can be easily accessed by all participating components [8]. It is worth mentioning the *MCT* framework again, which also enables data transfer among different components via MPI communication [20] rather than file based I/O. For instance, the *CCSM4* system is a single executable implementation, which includes a top-level driver and components integrated via standard *init/run/finalise* interfaces by leveraging *MCT* [12]. From a functionality view point, the *MCT* tool might be the most similar approach to our work, but it requires to compile all individual models or applications together to generate a single executable binary file. The combined binary ensures that all processes can share the same MPI intra-communicator to communicate with each other through MPI function calls. However, this prerequisite is not easy to meet, because it is difficult to combine a large number of separately developed components due to possible collisions on global variables and function names.

However, since all MPI processes share the same `MPI_COMM_WORLD` communicator in *MCT*, local broadcast operations within a specific (individual) model becomes visible to all other processes belonging to other components. To overcome this limitation, P.A. Browne and S. Wilson [19] have proposed a very similar mechanism for coupling two specified models for the purpose of data assimilation, through a different use of the Message Passing Interface. In their solution, although two models are still compiled together to generate a single MPI job, they split the MPI communicator to enable local MPI communication within individual components. However, this solution implies that the source codes of all involved models have to be modified for enabling usage of split MPI communicators for local communication.

In addition, for supporting flexible communication patterns and better communication efficiency of I/O data transfer, the Adaptable I/O System (*ADIOS*) framework [5] has been proposed to support flexible direct data transfer having different I/O patterns. Similarly to the *OASIS coupler*, the users have to modify the models or applications to use *ADIOS*'s specific interfaces. C. Zhang et al. have proposed and implemented a butterfly implementation of data transfer and then develop an adaptive data transfer library for the coupled systems [13]. F. Zhang et al. presented a distributed data sharing and task execution framework to minimize inter-application data exchange [22]. In summary, existing works fail to provide a general framework to integrate separately developed models or applications into a coupled system (so that direct parallel data transfer among all component models could be supported) without modifying the source codes of individual components. To the contrary, our I/O middleware intends to offer a universal communication framework to accelerate data transfer among components in coupled systems in order to meet strict time constraints. Additionally, our framework requires only minimal modifications to existing application code.

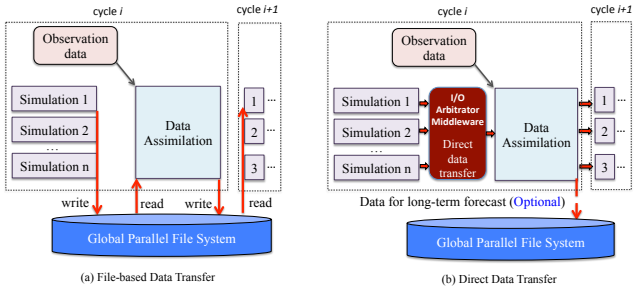


Fig. 1: The communication pattern of one cycle in the *SCALE-LETKF* system utilizing file I/O (a.) or direct data transfer (b.) methods.

3 I/O Arbitrator Middleware

As we mentioned before, our target coupling system of *SCALE-LETKF* repeats a two-step cycle of simulation and data assimilation, performed by two separately developed models, i.e., *SCALE* and *LETKF*. The I/O communication of one cycle in the current *SCALE-LETKF* system is depicted in Figure 1 (a). As seen, the *netCDF* output data of the *Simulation* processes are first written to the global parallel file system, which in turn is read by the *Assimilation* processes. Note that we run multiple *SCALE* model simulations at the same time (denoted by *Simulation* 1 to n in the figure), which are called "ensemble" simulations and are required for applying the *LETKF* data assimilation scheme. Each ensemble member takes slightly different initial condition and outputs different results so the total I/O amount is roughly equal to the I/O amount of one member multiplied by the number of ensemble members. After computation, the ensemble model of *SCALE* generates a large amount of output data, written in *netCDF* format, which are all requested by the subsequent assimilation step of the same cycle. In brief, the output data generated by the simulation process will be used by the corresponding assimilation process, which indicates that I/O communication is performed between process pairs.

To reduce the time needed for data transfer, we have been developing a novel I/O middleware to allow direct parallel data transfer between the two component models. Figure 1 (b) illustrates the workflow of the system when the I/O middleware is utilized. As a result, in each cycle, the output data of simulation processes are directly forwarded to the assimilation processes, as well as the analyzed results generated by assimilation processes which can be directly transferred to the simulation processes in the next cycle. Specifically, the I/O middleware connects the two kinds of processes by using MPI communication [20], and consequently, it enables direct communication between the simulation processes and the assimilation processes.

Although only the *SCALE-LETKF* application is detailed in this paper, it is worth emphasizing that our proposed I/O middleware is a general solution for coupled Big Data processing applications. In order to handle a wide range of possible I/O patterns the middleware is customizable using configurations

files. Different configurations enable deployment for applications with different properties, such as different number of component models, or different I/O communication patterns.

3.1 High Level Architecture

Figure 2 shows the I/O stack of the I/O arbitrator middleware, which is used to support direct parallel data transfer between simulation processes (*SIM* in the figure) and data assimilation processes (*DA* in the figure) in our case study. Except for the application layer itself, the *netCDF*, *POSIX*, and MPI layers are involved in the middleware. Briefly speaking, the mechanism of direct parallel data transfer is transparent to the applications.

As it is shown in the figure, communication is currently performed by using MPI, for which the following subsection will discuss the construction of the communication context between two kinds of processes.

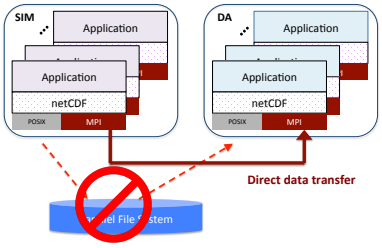


Fig. 2: Architectural overview of the middleware

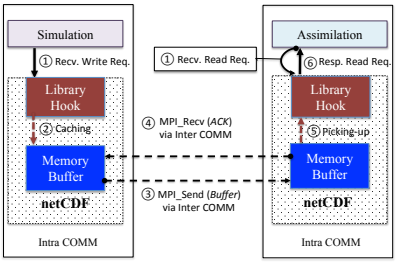


Fig. 3: The internals of direct data transfer among workflow components

3.2 Establishing Communication

Because the simulation and data assimilation models are separately developed applications, and are executed as separate MPI jobs, they do not share the same MPI communicator. To overcome this problem, our prototype implementation currently utilizes the standard MPI intercommunicator family of routines to establish a communication context between the two types of jobs.

We provide an overview of the current MPI based implementation. At initialization time, the server process, i.e., an *Assimilation* process opens a port using `MPI_Open_port()`, and then publishes it by calling the `MPI_Publish_name()` feature. Subsequently, the connection thread of each individual *Assimilation* process waits in `MPI_Comm_accept()`. Client processes, i.e., processes of the *Simulation* component, connect to the server processes with `MPI_Comm_connect()` once they successfully obtained the service name by using `MPI_lookup_name()`. As a result, processes of both components can communicate with each other by using standard MPI functions. Once the data transfer had taken place, the client processes proactively disconnect and the server processes can unpublish their connection services with `MPI_Unpublish_name()`.

3.3 Direct Data Transfer

The parallel data transfer between the simulation processes and the assimilation processes is carried out when the communication context has been constructed. Figure 3 depicts the details of the synchronized direct data transfer in the I/O middleware, where the interaction between two kinds of processes can be described as follows:

1. The *Simulation* process writes the output data to the file system through calling the `write()` system call. We assume that the *Assimilation* process will eventually read the contents of the same file, but the *Assimilation* process is blocked until the requested data is satisfied in Step ⑥.
2. The `write()` call are intercepted by the middleware, and the write contents are cached in the designated *Memory Buffer* instead of flushing them to the global file system.
3. The buffered data is forwarded from the *Simulation* node to the destination node, i.e. the *Assimilation* process, by calling the `MPI_Send()` routine.
4. The *Assimilation* process responds an *ACK* message, when it has received the data sent by the *Simulation* process, through calling `MPI_Recv()`. Consequently, the data is cached in the designated memory buffer for satisfying potential future read requests.
5. According to the parameters of the `read()` request, which was blocked because the required data was not yet available, the specified piece of data will be picked up by *Library Hook* from *Memory Buffer*.
6. The *Assimilation* process resumes its execution after it received the data from *Library Hook*.

Both *Simulation* and *Assimilation* processes are able to exchange their data through direct data transfer. Specifically, all `write()` requests will be fulfilled when the contents have been buffered in the memory, and all cached data is eventually sent to the destination process. On the other side, all `read()` requests will be satisfied with the data buffered in the memory, which was initially received from the source process.

We provide two kinds of direct data transfer schemes, i.e. *Synchronous Data Transfer* and *Asynchronous Data Transfer*. In *Synchronous Data Transfer*, all data are transferred when the output data have been completely flushed (i..e the output *netCDF* file is closed). On the other hand, the mechanism of *Asynchronous Data Transfer* employs a `Communication Thread` to asynchronously send parts of the flushed data during the computation, which enables overlapping computation and data transfer. Once the output *netCDF* file is closed, only dirty data, which have been updated since the previous data transmission, are transferred.

3.4 Implementation for SCALE-LETKF

To demonstrate the effectiveness of direct parallel data transfer between the simulation and assimilation processes in *SCALE-LETKF*, we have developed a

proof-of-concept implementation of the proposed I/O middleware. Besides, since data is exchanged between each *SCALE* process and the corresponding *LETKF* process in *netCDF* format, we have made slight modifications to the *netCDF* library itself (using *ver. 4.2.2.1*), so that it complies with the proposed I/O middleware to enable direct data transfer in an application transparent fashion.

4 Evaluation

This section first describes the experimental setup and experimental methodology for evaluating the proposed I/O middleware. It then presents experimental results and provides the relevant discussion. At last, we summarize the key points of our direct parallel data transfer approach.

4.1 Experimental Setup

Evaluation experiments to assess the advantages of the *SCALE-LETKF* system equipped with our current prototype middleware were conducted on the K computer [2]. The K computer is Japan's flagship supercomputer sporting 88,128 compute nodes (8 CPU cores each), with peak performance more than 10 petaFLOPS. The K computer took the first place of TOP 500 in 2011, and as of June 2015, it is ranked as the fourth fastest machine of the world [3].

As for the input data used in our experiments, we employ real world observations to test the efficiency of *SCALE-LETKF* when equipped with the proposed I/O middleware. In all experiments each MPI process was allocated to one compute node, and we logged the results related to I/O operations during the execution. Three real world test cases for regional weather analysis were used. In each measurement, *SCALE* is composed of up to 100 ensemble instances. *Test Case 1* and *Test Case 2* have 4 processes in each ensemble instance, but there are 48 processes in each ensemble instance of *Test Case 3*. *LETKF* consists of only one instance, but it contains the same number of processes as all *SCALE* instances in total. Note that every MPI process is allocated onto one computing node, and *openMP* is used to explicitly direct multi-threaded parallelism.

Table 1: Total Amount of Transferred Data in the Case Study.

Ensemble Size	**Test Case 1**	**Test Case 2**	**Test Case 3**
10	3, 468 MB	6, 720 MB	53, 328 MB
20	6, 936 MB	13, 440 MB	101, 808 MB
40	13, 872 MB	26, 880 MB	198, 768 MB
60	20, 808 MB	40, 320 MB	295, 728 MB
80	27, 744 MB	53, 760 MB	392, 688 MB
100	34, 680 MB	67, 200 MB	489, 648 MB

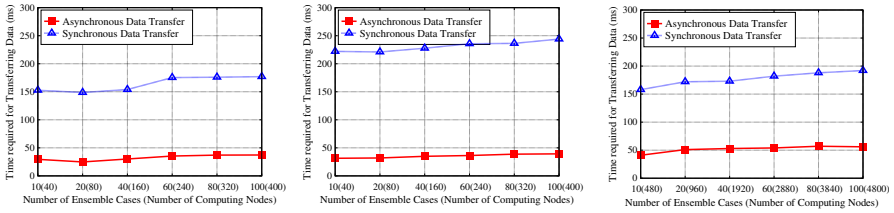
Table 1 summarizes the size of transferred data for the cases having different number of ensemble instances. Note that in our current execution model, each application instance corresponds to a separate MPI job.

4.2 Experimental Results

The main limitation of our current proof-of-concept I/O middleware is that we can run only one cycle of the *SCALE-LETKF* system. In other words, each *SCALE* process generates output data after simulation, which will be read by the corresponding *LETKF* process as input for assimilation.

Communication Time for Transferring Data. While running the selected three test cases, we first measured the communication time for transferring data between *SCALE* and *LETKF* after the computation of *SCALE*, as the function of increasing the number of ensemble instances from 10 to 100. Figure 4 shows the time required for transferring the data from *SCALE* to *LETKF* by using two direct data transfer schemes. The horizontal axis represents the number of ensemble instances and the vertical axis shows the time required for data transmission. As the experimental results imply, the communication time for transferring data between the two components remains essentially unchanged, even with the growing number of involved processes, which is due to the pair-wise communication pattern of the *SCALE-LETKF* system.

Another interesting observation is that the *Asynchronous Transfer* scheme requires less than $one-third$ time for transferring the data after the computation of data simulation, compared with *Synchronous Data Transfer*. In other words, the *LETKF* processes can start the computation of data assimilation quite earlier, when we employ the *Asynchronous Transfer* scheme. This is because the dirty data occupy a small proportion of all output data.

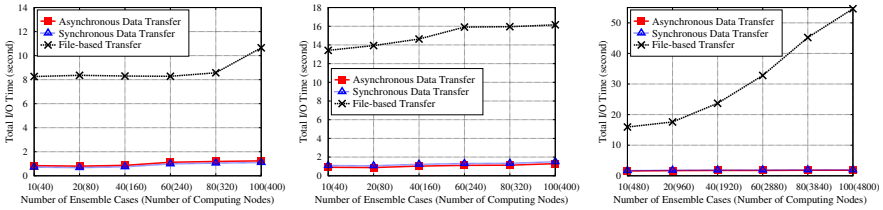


(a) Test Case 1 (b) Test Case 2 (c) Test Case 3

Fig. 4: Communication time needed for transferring data from *SCALE* to *LETKF* (both of them are supported by the proposed middleware).

I/O Acceleration. For comparison, we recorded the time required for I/O operations between the *SCALE* and *LETKF* processes by using both actual file I/O operations and the mechanism of direct data transfer. Figures 5(a), (b) and (c) indicate the time required for carrying out I/O operations between the two types of processes utilizing file I/O and the proposed mechanism, respectively. Note that the I/O time shown by the proposed mechanisms includes the time needed for memory operations, and the time required for transferring the data from *SCALE* to *LETKF*.

As the Figure depicts, the proposed mechanism can substantially reduce the time needed for I/O operations between *SCALE* and *LETKF* processes compared to the file I/O-based data transfer. For example, when the size of ensemble instances is 100 using the test case *Test Case 3*, the mechanism of

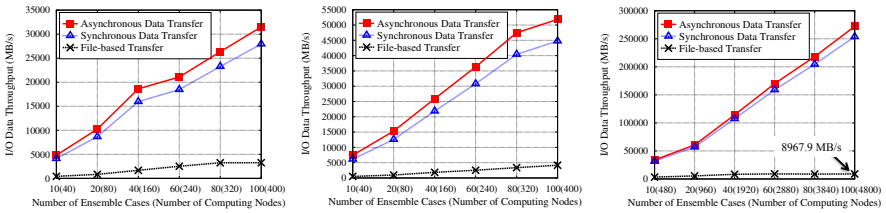


(a) Test Case 1 (b) Test Case 2 (c) Test Case 3

Fig. 5: I/O time contrasting file I/O and direct data transfer.

direct data transfer can yield over 30× speedup on I/O operations, which in turn implies that more time can be devoted to perform simulation and data assimilation, and that the total execution time can be consequently decreased. Furthermore, the file-based data transfer may require significantly increased I/O time due to contention on the parallel file system. The case of *Test Case 2* required 34.1% more time for conducting file I/O operations, compared with the case of *Test Case 1*, because the size of transmission data needed by the former case is two times of the size of transmission data of the latter one. In contrast, direct data transfer does not increase the transfer time significantly even for double size data.

Data Throughput. After verifying the proposed mechanism can indeed reduce the time needed for exchanging the data between *SCALE* and *LETKF* in our test cases, this section aims to measure the I/O data throughput while executing various test cases. Figures 6(a), (b) and (c) show the results about I/O data throughput reported by performing the tests with varying ensemble sizes, respectively. As seen, the proposed scheme of direct data transfer outperforms the scheme of file I/O-based data transfer, and it achieves from 758.3% to 2933.3% data rate improvements for the selected test cases. Particularly, improvements are getting remarkable while the ensemble size is getting larger that indicates more data are required to be processed.



(a) Test Case 1 (b) Test Case 2 (c) Test Case 3

Fig. 6: I/O data throughput utilizing file I/O and direct data transfer.

Another noticeable issue, implied by the figures, is the fact that the larger the amount of data is to be exchanged, the higher the benefits become by utilizing the direct data transfer method. In addition, compared with the scheme of *Synchronous Data Transfer*, the *Asynchronous Data Transfer* scheme can achieve more attractive performance improvements.

4.3 Summary

With respect to comparing direct data transfer and file I/O based data transfer, we emphasize the following two key observations. First, with increasing number of processes, direct data transfer yields better relative performance. Second, the more time reduction and higher data throughput can be achieved with the growing size of the involved data. In brief, we conclude that the proposed file I/O middleware is able to significantly reduce the time required by exchanging data between the component models in the *SCALE-LETKF* workflow system.

Furthermore, the implemented I/O middleware offers a general framework for inter-component data exchange in workflow systems, where individually developed applications are coupled together. By accelerating the execution of such systems, we believe our newly proposed middleware, facilitated with the direct data transfer functionality, is particularly important for systems with rigorous time constraints.

5 Concluding Remarks

This paper has proposed a general I/O middleware for Big Data processing, coupled workflows that are comprised of multiple individually developed components. Our framework enables direct parallel data transfer among component models in order to reduce data exchange time, which we applied to the *SCALE-LETKF* data assimilation based weather forecasting system.

Experimental results on the K computer using up to 4800 nodes have shown that the proposed mechanism can significantly reduce the time spent on I/O operations among *SCALE* and *LETKF*. Furthermore, we have demonstrated that the benefit of larger data throughput increases with the growing amount of data that is required to be processed.

The current implementation of the middleware relies on the MPI library for data transmission, but our long term vision is to implement data transfer on top of a Low Level Communication library (*LLC*), which will enable us to establish connections among arbitrary MPI jobs. LLC is part of the development plan of the post K supercomputer, the next generation flagship supercomputer in Japan.

References

1. Network Common Data Form (netCDF). www.unidata.ucar.edu/netcdf/ (2013).
2. RIKEN AICS: K computer, http://www.aics.riken.jp/en/k-computer/ (2011).
3. TOP500 Supercomputer Sites. http://www.top500.org/ (2015).
4. D. Reed, and J. Dongarra. Exascale Computing and Big Data. Communications of the ACM, Vol.58:56–68, June 2015.
5. N. Podhorszki, S. Klasky, and Q. Liu et al. Plasma Fusion Code Coupling Using Scalable I/O Services and Scientific Workflows. Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science *WORKS '2009*, Nov 2009.
6. B. Hunt, E. Kostelich, and I. Szunyogh. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter. Physica D: Nonlinear Phenomena, Vol.230(1): 112–126, 2007.

7. S. Nishizawa, H. Yashiro, and H. Tomita et al. Influence of grid aspect ratio on planetary boundary layer turbulence in large-eddy simulations. Geosci. Model Dev., Vol. 8(10):3393–3419, 2015.
8. S. Valcke, V. Balaji, and G. Riley et al. Coupling Technologies for Earth System Modelling. Geosci. Model Dev., Vol. 5: 1589-1596, 2012.
9. F. Chen, J. Dudhia. Coupling an advanced land surface-hydrology model with the Penn State-NCAR MM5 modeling system. Part I: Model implementation and sensitivity. Mon. Weather Rev., Vol.129(4):569–585, 2001.
10. Z. Janjic, and T. Black. An ESMF unified model for a broad range of spatial and temporal scales. Geophys. Res. Abstr., Vol. 9. No. 05025, 2007.
11. L. Liu, G. Yang, and B. Wang et al. C-Coupler1: A Chinese community coupler for Earth system modelling. Geosci. Model Dev. Discuss., Vol. 7(3): 3889–3936, 2014.
12. A. Craig, M. Vertenstein, and R. Jacob. A New Flexible Coupler for Earth System Modeling developed for CCSM4 and CESM1, Int. J. High Perform. C, Vol.26(1):31-42, 2012.
13. C. Zhang, L. Liu, G. Yang et al. Improving data transfer for model coupling. Geosci. Model Dev. Discuss., Vol. 8, pp. 8981–9020. 2015.
14. Masaru Kunii. Mesoscale Data Assimilation for a Local Severe Rainfall Event with the NHMLETKF System. Weather Forecasting, Vol. 29, pp. 1093–1105, 2014.
15. Jay Larson, Robert Jacob, and Everest Ong. T he Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models. Int. J. High Perform. C, Vol.19(3):277-292, 2005.
16. A. Craig, R. Jacob, B. Kauffman, and Y. He et al. CPL6: The New Extensible, High Performance Parallel Coupler for The Community Climate System Model. Int. J. High Perform. C, Vol. 19(3):309-327, 2005.
17. C. W. Armstrong, R. W. Ford, and G. D. Riley. Coupling integrated Earth System Model components with BFG2. Concurr. Comp-Pract. E., Vol. 21(6):767–791, 2009.
18. S. Valcke, R. Budich, and M. Carter et al. The PRISM software framework and the OASIS coupler. In proceedings of the 18 Annual BMRC Modelling Workshop, Nov. 28 - Dec. 1, 2006, Melbourne, Australia.
19. P.A. Browne, S. Wilson. A simple method for integrating a complex model into an ensemble data assimilation system using MPI. Environ. Modell. Softw., Vol.68:122–128, 2015.
20. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 2.2, September 2009.
21. G. Watson, W. Frings, and C. Knobloch, et al. Scalable control and monitoring of supercomputer applications using an integrated tool framework. In Proceedings ICPPW '2011, pp. 457-466, 2011.
22. F. Zhang, C. Docan, M. Parashar et al. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In Proceedings of IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS '2012), pp. 1352-1363, 2012.
23. S. Valcke, A. Craig, R. Dunlap, and G. Riley. Sharing experiences and outlook on Coupling Technologies for Earth System Models. Bull. Amer. Meteor. Soc., DOI:10.1175/BAMS-D-15-00239.1, 2015.
24. M. Dorier, M. Dreher, and T. Peterka et al. Lessons Learned from Building In Situ Coupling Frameworks. In Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization. ACM, pp. 19-24, 2015.
25. T. Miyoshi, K. Kondo, and K. Terasaki. Big Ensemble Data Assimilation in Numerical Weather Prediction. IEEE Computer, Vol. 48(11):15–21, 2015.