System Software Research Team

1. Team members

Yutaka Ishikawa (Team Leader)
Atsushi Hori (Senior Scientist)
Yuichi Tsujita (Research Scientist)
Keiji Yamamoto (Postdoctoral Researcher)
Kazumi Yoshinaga (Postdoctoral Researcher)
Akio Shimada (Research Associate)
Masayuki Hatanaka (Research Associate)
Norio Yamaguchi (Research Associate)
Toyohisa Kameyama (Technical Staff)

2. Research Activities

The system software team focuses on the research and development of an advanced system software stack not only for the "K" computer but also for towards exascale computing. There are several issues in carrying out future computing. Two research categories are taken into account: i) scalable high performance libraries/middleware, such as file I/O and low-latency communication, and ii) a scalable cache-aware, and fault-aware operating system for next-generation supercomputers based on many core architectures.

3. Research Results and Achievements

3.1. PRDMA (Persistent Remote Direct Memory Access)

The goal of this research is to design and evaluate an efficient MPI implementation for neighborhood communication by taking advantage of the Tofu interconnect, which has multiple RDMA (Remote Direct Memory Access) engines and network links per MPI process. The neighbor communication pattern is commonly used in the ghost (or halo) cell exchanges. For example, the SCALE-LES3, weather and climate models developed at RIKEN AICS, includes the multiple stencil computations. So, the neighborhood communication is a dominant communication pattern within the SCALE-LES3. Specifically, it is the two dimensional 8-neighbors ghost cell exchanges with periodic boundary conditions, which occupies about ten percent of the execution time. Nowadays, supercomputers using three-or-higher dimensional torus have been deployed, such as the Blue Gene / Q and the K computer. For instance, the Torus Fusion (called Tofu) interconnect employed by the K computer has 6 dimensional torus and mesh as a physical topology and its node controller has 4 RDMA engines and 10 network links. These networks are possible to improve the neighborhood communication performance when MPI ranks are properly mapped on the network topology and the transfer requests are properly scheduled on the multiple RDMA engines. Unfortunately, the

transfer-scheduling algorithm in the default MPI implementation provided on the K computer uses a simple round-robin method to distribute the transfer requests among the multiple RDMA engines. Therefore, our previous work has developed an RDMA-transfer scheduling algorithm, called *Modified-Bottom-Left*, to avoid the congestions on physical network links and the conflicts of receiver-side RDMA engines. In the latest MPI specification version 3.0 (hereafter referred to as "MPI-3.0"), neighborhood collective primitives were introduced. For example, the MPI_Neighbor_alltoallw primitive is a neighborhood collective version of MPI_Alltoallw. These primitives communicate with the user-defined processes instead of all participants on the communicator. Also, non-blocking collective primitives such as MPI_Ineighbor_alltoallw primitive are introduced in MPI-3.0. To support MPI-3.0 on K computer, we are porting MPICH-3.1, which is one of the major MPI implementations. Since the default neighborhood collective implementation in MPICH-3.1 is a generic implementation, we develop the optimized implementation of neighborhood collective on Tofu interconnect using our RDMA-transfer scheduler.

Proposal of RDMA-based collective communication, Cached-Multi-W

If all point-to-point user-data transfers defined in a neighborhood collective communication are replaced by RDMA Write (or RDMA Read) transfers as a whole, the RDMA-based transfers can progress the non-blocking collective communication without CPU intervention, and reduce extra copy overheads and memory consumption for data transfers due to the Zero-Copy feature. Therefore, an RDMA-based approach, called *Cached-Multi-W*, is proposed to further reuse a cached series of RDMA descriptors. In this approach (see Figure 1), once a neighborhood collective communication function such as MPI_Neighbor_alltoallw() is called, the arguments and a series of RDMA (Write) descriptors corresponding to the collective communication pattern are cached as a reusable entry. And then if the subsequent collective calls match the arguments with a cached entry, the calls reuse a cached series of RDMA descriptors, instead of generating and scheduling the RDMA descriptors.

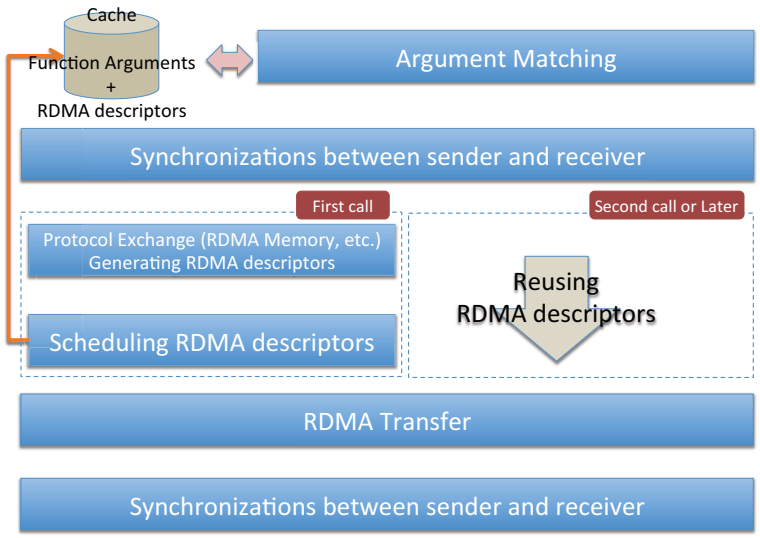


Figure 1 Cached-Multi-W (Cached Multiple RDMA Writes) approach

Implementation and Evaluation

The proposed *Cached-Multi-W* approach has been implemented for MPI_Neighbor_alltoallw() in MPICH-3.1 on K computer. This implementation uses a generator and scheduler of RDMA descriptors developed in PRDMA (Persistent Remote Direct Memory Access).

We measured two MPI_Neighbor_alltoallw implementations in a ghost cell exchange: (1) MPICH default implementation, and (2) the proposed Cached-Multi-W implementation. The communication pattern is a ghost cell exchange for two dimensional 9-point stencil computation, the cell size is 800 bytes (MPI_DOUBLE × 100), the ghost width is 2, and the cell shapes per process are 8x8, 16x16, and 32x32. In Figure 2-1 and Figure 2-2, the horizontal axis shows each call of MPI_Neighbor_alltoall(). The vertical axis shows the elapsed time for a call. The proposed Cached-Multi-W implementation is up to 77 % better than the MPICH original rendezvous implementation.

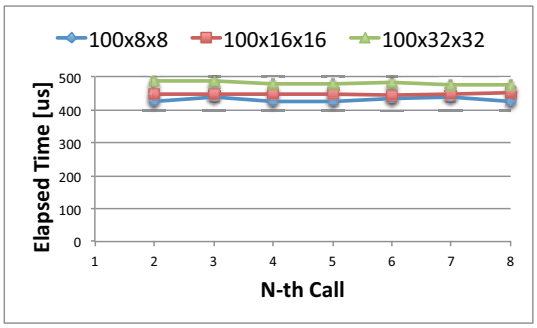


Figure 2 MPICH default implementation

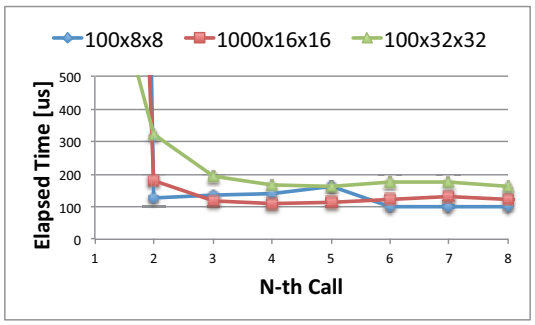


Figure 3 Cached-Multi-W implementation

3.2. New Process / Thread Model

Partitioned Virtual Address Space

From FY2012, we have been developing a new process / thread model that is suitable for the many-core architectures. The many-core architectures are gathering attention towards the next generation supercomputing. Many-core architectures have a large number of low performance cores, and then the number of parallel processes within a single node becomes larger on many-core environments. Therefore the performance of inter-process communication between the parallel processes within the same node can be an important issue for parallel applications.

Partitioned Virtual Address Space (PVAS) is a new process model to achieve high-performance inter-process communication on the many-core environments. On PVAS, multiple processes run in the same virtual address space as described in Figure 4 to eliminate the communication overhead due to the process boundaries that the current modern OSes introduce for inter-process protection. In PVAS, the data owned by the other process can be accessed by the normal load and store machine instructions, just like the same way accessing the data owned by itself. Then, high-performance inter-process communication is achieved.

We implemented the prototype of the PVAS process model in the Linux kernel in FY2012. We improved its quality and published it as open source software in FY 2013.

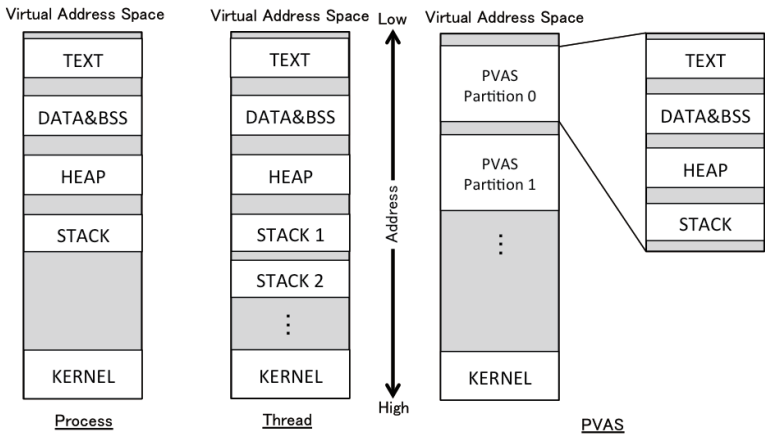


Figure 4 Partitioned Virtual Address Space

In FY2014, we evaluated the PVAS model by implementing a new BTL (a low-level communication layer of OpenMPI) to utilize the advantage of PVAS. In this implementation, the rendezvous protocol is optimized so that only one memory copy operation takes place. Contrastingly, in the current implementation of intra-node communication of OpenMPI, two memory copies must be involved to transfer a message. Figure 5 show the NPB performance comparison of the conventional sm BTL (BTL for shared memory) and our optimized PVAS BTL using the rendezvous protocol. The advantage of PVAS is not only to reduce the number of memory copies, but also to reduce the memory for page tables to map physical memory. In an all-to-all communication, for example, all processes in a node communicate with each other. This results in to map memory regions in $O(N^2)$ and to have $O(N^2)$ page table entries. Figure 6 shows the comparison of all-to-all memory consumption between sm BTL and PVAS BTL. As shown in this graph, conventional sm BTLs consumes memory $O(N^2)$, however, PVAS BTL consumes memory in $O(N)$.

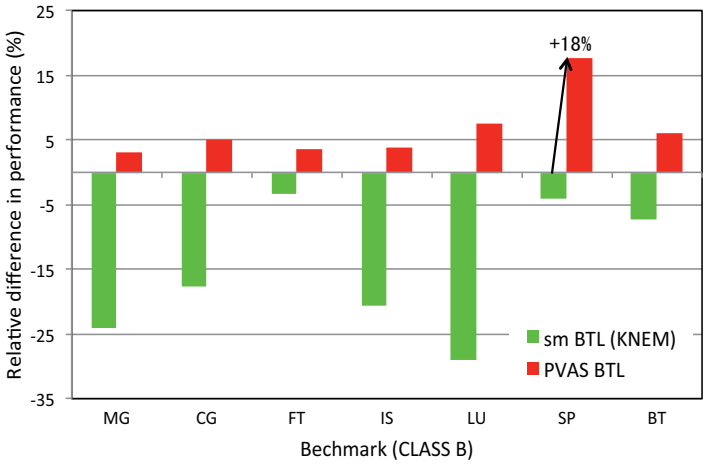


Figure 5 NPB Performance between sm BTL and PVAS BTL (Xeon Phi)

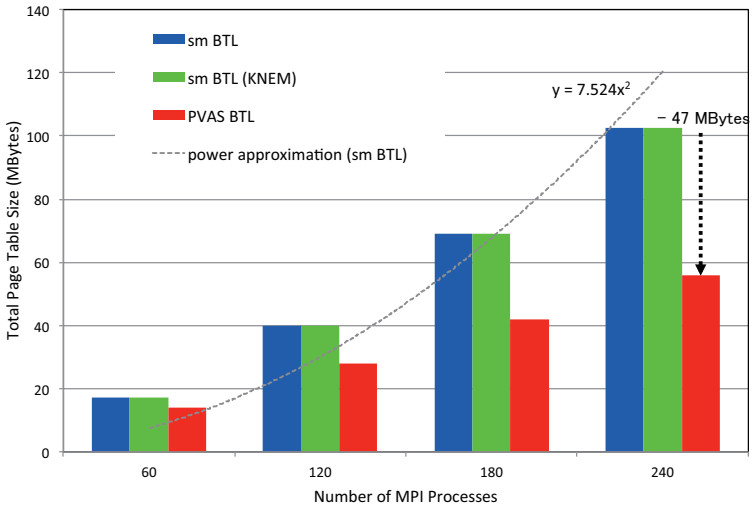


Figure 6 Alltoall Memory Consumption Comparison between sm BTLs and PVAS BTL (Xeon Phi)

3.3. Fault Resilience

With the increasing fault rate on high-end supercomputers, the topic of fault tolerance has been gathering attention. To cope with this situation, various fault-tolerance techniques are under investigation; these include user-level, algorithm-based fault-tolerance techniques and parallel execution environments that enable jobs to continue following node failure. Even with these techniques, some programs that have static load balancing, such as stencil computation, may underperform after a failure recovery. Even when spare nodes are present, they are not always substituted for failed nodes in an effective way.

There are some questions of how spare nodes should be allocated, how to substitute them for faulty nodes, and how much the communication performance is affected by such a substitution. The third question stems from the modification of the rank mapping by node substitutions, which can incur additional message collisions. In a stencil computation, rank mapping is done in a straightforward way on a Cartesian network without incurring any message collisions. However, once a substitution has occurred, the node-rank mapping may be destroyed. Therefore, these questions must be answered in a way that minimizes the degradation of communication performance.

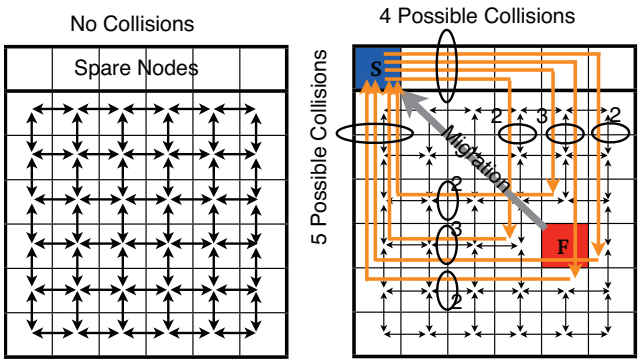


Figure 7 Message collisions by substituting a failed node (5P-stencil)

In FY2014, several spare-node allocation and node-substitution methods were studied, analyzed, and compared in terms of communication performance following the substitution. Three node substitution methods, 0D sliding, 1D sliding and 2D sliding are proposed (Figure 8).

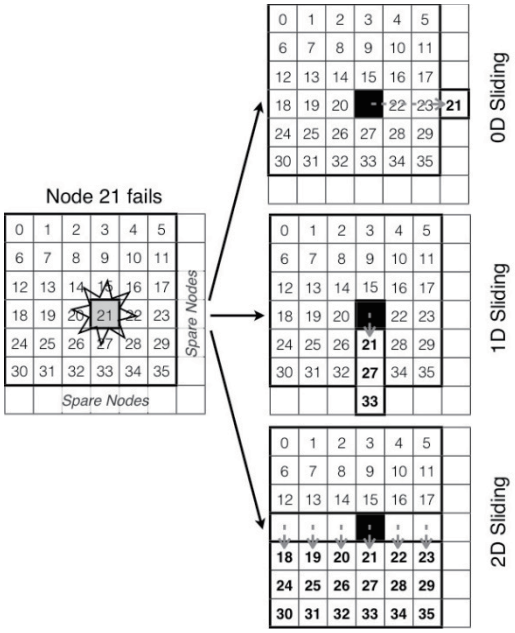


Figure 8 Proposed Three Failed Node Substitution Methods

It was revealed that when a failure occurs, the point-to-point (P2P) communication performance on the K computer can be slowed by a factor of three (Figure 9). On BG/Q, P2P performance can be slowed by a factor of five (Figure 10).

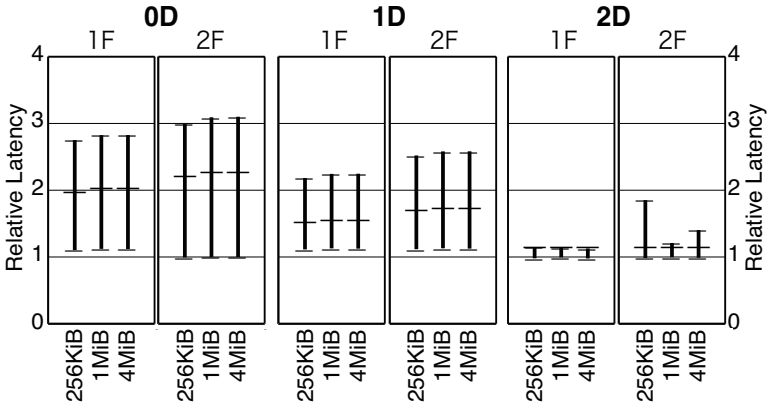


Figure 9 P2P Performance Degradation by Using Spare Node(s) – the K computer

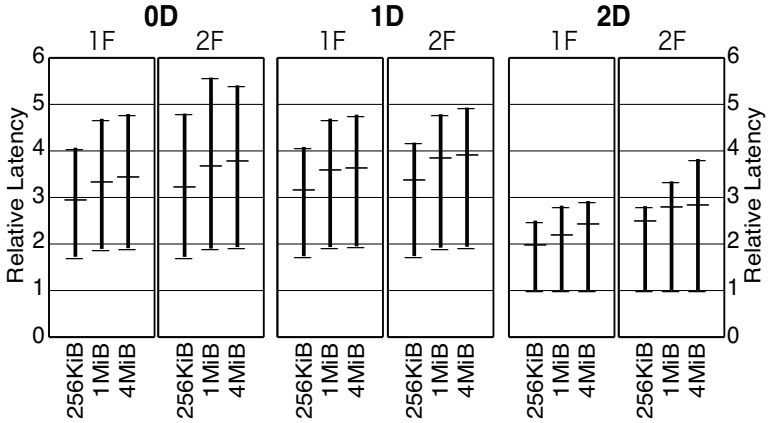


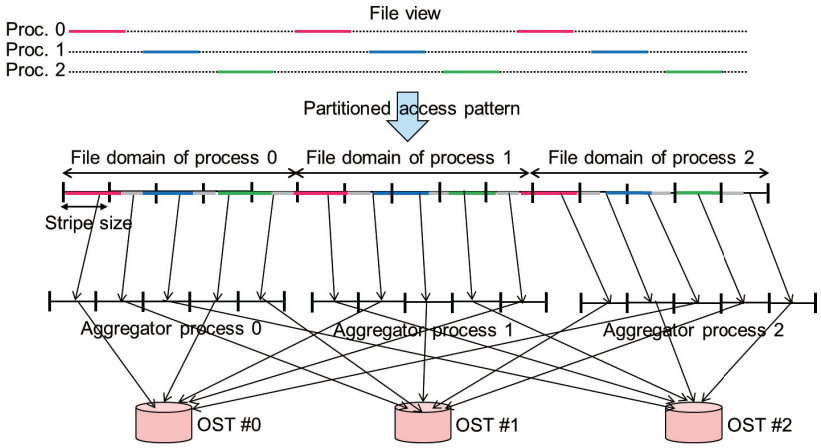
Figure 10 P2P Performance Degradation by Using Spare Node(s) - JUQUEEN (BG/Q)

3.3. Scalable MPI-IO Using Affinity-Aware Aggregation

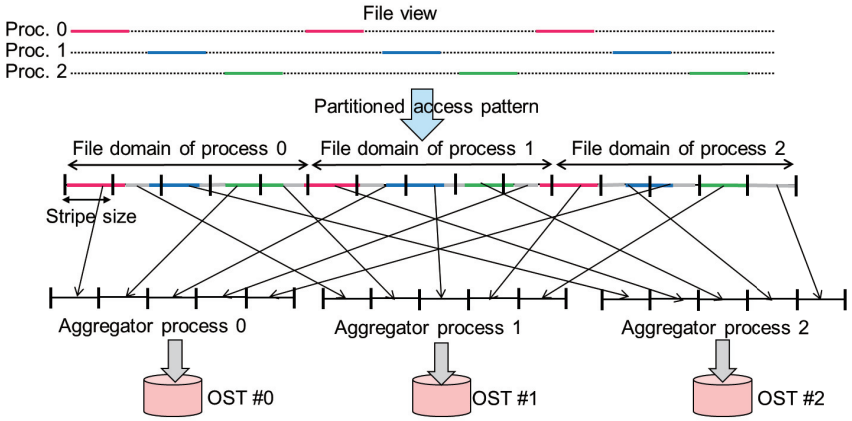
A commonly used MPI-IO library named ROMIO has the two-phase I/O (TP-IO) scheme to improve collective I/O performance for non-contiguous accesses. This research is addressing to optimize TP-IO implementation for further I/O performance improvements than the original one.

In the FY2014, ROMIO in the MPI library on the K computer has been arranged to have (1) affinity-aware data aggregation scheme and (2) I/O throttling approach.

Firstly, we focused to optimize data aggregation scheme. Figure 11 shows I/O flow of TP-IO in the original and optimized ROMIO implementation on the K computer.



(a) Original data aggregation scheme

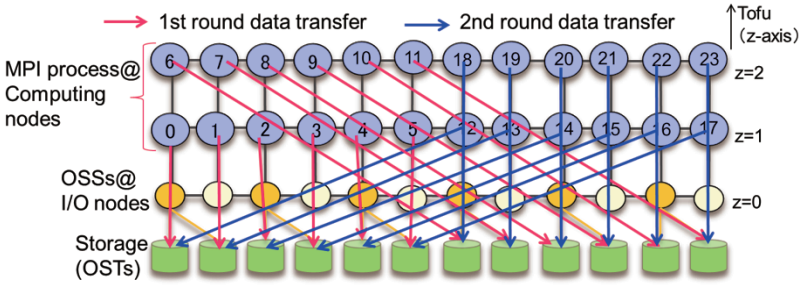


(b) Optimized data aggregation scheme

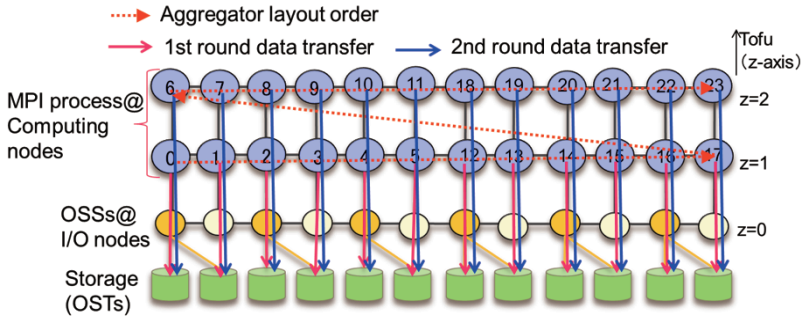
Figure 11 Original and optimized data aggregations in collective write operations

This figure depicts data flow from three MPI processes with the same number of aggregator processes. Figure 11 (a) illustrates TP-IO data flow done in the original implementation on the K computer. In this case, data stream from aggregators are going to each Object Storage Target (OST) of the FEFS file system. As a result, network contention occurs in such data transfer pattern. While in Figure 11 (b) that we adopted based on the similar optimization done for a Lustre file system, each aggregator collects data from every MPI process in order to form striped data layout. Therefore every aggregator just writes collected data to the target OST only, and network contention can be alleviated.

Furthermore, we have focused to have aggregator process layout which suits to the Tofu interconnect configuration of the K computer. Figure 12 shows examples of aggregator process layout and data transfer with striping-oriented aggregation only and with both striping-oriented aggregation and affinity-aware aggregator assignment.



(a) Striping-oriented only (default)



(b) Both striping-oriented & affinity-aware aggregator layout

Figure 12 Aggregator layout and data flow in TP-IO with 2x3x4 process layout

In this figure, data transfers in the first and second rounds of striping accesses on an FEFS file system are illustrated as a simple example. Numbers in circles stand for MPI rank. Since aggregator layout of the former case is based on MPI rank ascending order from zero, network contention or unbalanced network utilization may happen if the aggregator layout does not suit to FEFS's striping layout. On the other hand, the latter case adopts new aggregation layout whose order is independent of user's process layout. The new scheme implemented in a ROMIO library layer checks a 6-D position information, and arranges new groups consisting of MPI processes which are on the same Tofu z-axis. The new scheme finally deploys aggregator task to each MPI process in a round-robin manner crossing Tofu z-axis in order to form FEFS striping layout-aware aggregator layout as shown in Figure 12 (b). As a result, we can eliminate network contention or unbalanced network utilization remarked in Figure 12 (a).

Secondly, I/O throttling was adopted in the above optimized TP-IO implementation. According to previous study about POSIX-I/O on the FEFS file system done in our team, I/O throttling approach succeeded to improve I/O performance. Along with this approach, we implemented a function to control the number of I/O requests generated from computing nodes to a target OST of an FEFS file system in the ROMIO library. Here the I/O request generation is aligned to data access layout on each OST not to have unnecessary file seek operations. Since TP-IO carries out data exchange

phases between file read and write phases in a read-modify-write manner, we also implemented step-by-step data exchanges aligned to the I/O throttling scheme for further improvements. Thus MPI process that generates an I/O request can go to the next data exchange phase not to have a long waiting time for a forthcoming data exchange phase.

Performance evaluation was carried out using computing nodes ranged from 192 to 3,072 nodes. I/O performance evaluation was done by using the HPIO benchmark with non-contiguous access patterns on a local file system of the FEFS on the K computer. The number of nodes was arranged not to have any interference from other users' applications. In the K computer case, we specified the number of nodes in a 3-D manner node allocation, where we chose the following five patterns; 2x3x32, 4x3x32, 8x3x32, 8x6x32, and 8x12x32. We deployed one MPI process per one computing node, thus the number of MPI processes was the same with that of used computing nodes. Figure 13 shows I/O throughput values relative to the number of MPI processes.

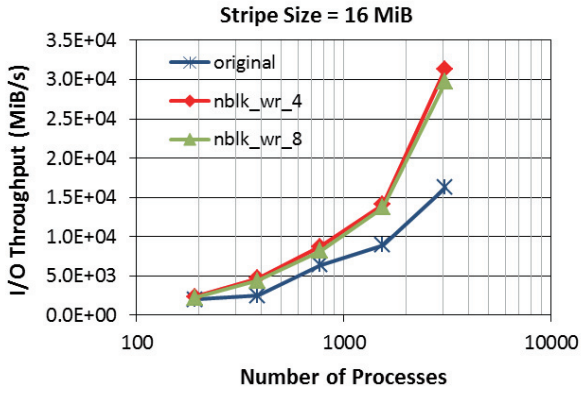


Figure 13 I/O throughput of Collective write

In this evaluation, we examined 4 and 8 for the number of I/O requests in the I/O throttling scheme indicated by "nblk_wr_4" and "nblk_wr_8", respectively in addition to the original implementation indicated by "original." The I/O throttling scheme with affinity-aware aggregation ("nblk_wr_4" and "nblk_wr_8") outperformed the original one and performed higher scalability. We have already observed that the number of I/O requests with 4 or 8 performed the best, but we have not realized why the numbers were the best at this moment. Examinations of properties of the I/O throttling scheme is our future work.

3.4. Big data processing on the K computer

This research is conducted by collaboration between the Data Acquisition team of RIKEN Spring-8 Center and the System Software Research team of RIKEN AICS. The goal of this project is to establish the path to discover the 3D structure of a molecule from a number of XFEL (X-ray Free Electron Laser) snapshots. The K computer is used to analyze the huge data transmitted from RIKEN Harima where SACLA XFEL facility is located.

In FY2012-2013, we developed parallel software running on the K computer to analyze images obtained by a light source, SACLA. The developed software consists of two components as shown in Figure 14. The first component is to select the representative images by a classic clustering computation. Thus, all images must be compared with all others. The second component is to classify the rest of images into the representative images. In this stage, we need to calculate for all possible combinations of representative images and rest of images.

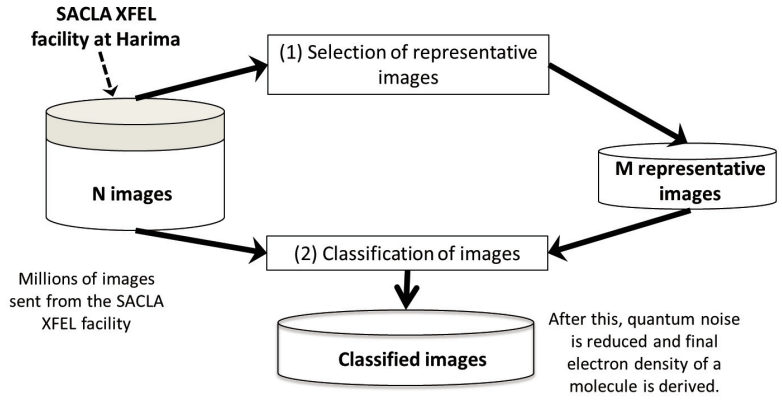
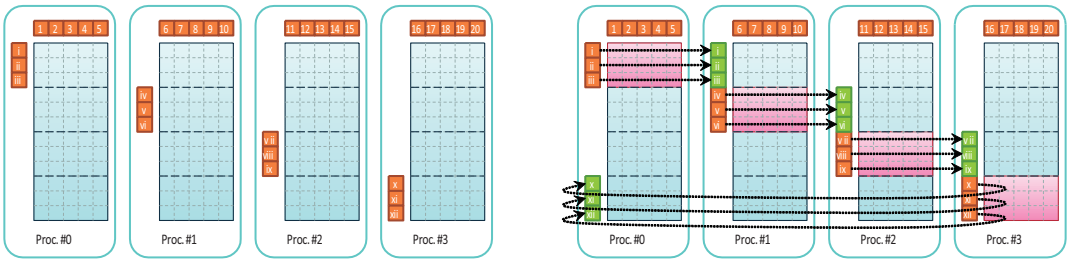
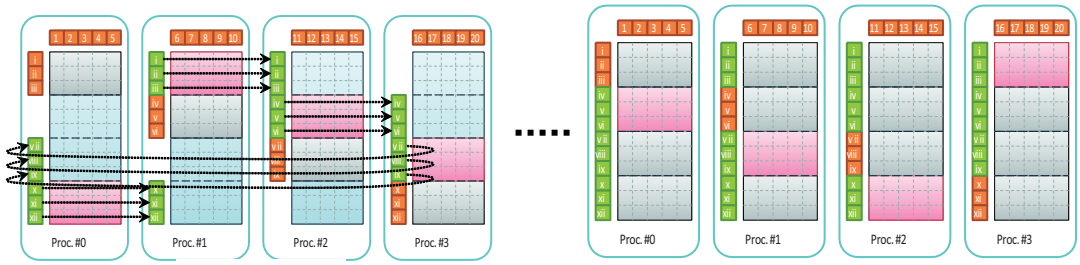


Figure 14 Block diagram of the procedure running on the K computer

In order to realize effective parallelization, the developed software keeps load balance between processes and reduces file input time by minimizing total size of the input from storage. Figure 15 shows the workflow for classification of images. The software reads the image file only once and read images are passed to neighbors in background at every calculation step. The orange squares in Figure 15 are images which are read from storage, and the greens are transferred from the neighbor process. The calculation is finished at N_p step, where N_p is number of processes.



Initial State　　Step 1



Step 2 Last Step (Step N_p, N_p is # of procs)

Figure 15 Calculation workflow for classification of images

In FY2014, based on the developed software, we designed a new framework, named *pCarp*, that describes any possible combination of two records in a dataset processed by all participating processes. This parallel processing is used not only our target application, but also used to analyze gene sequencing data, images obtained by electron microscopes, and so on. The framework users do not need to write any parallel program, but write just sequential program. All parallelizing tasks are performed by *pCarp*.

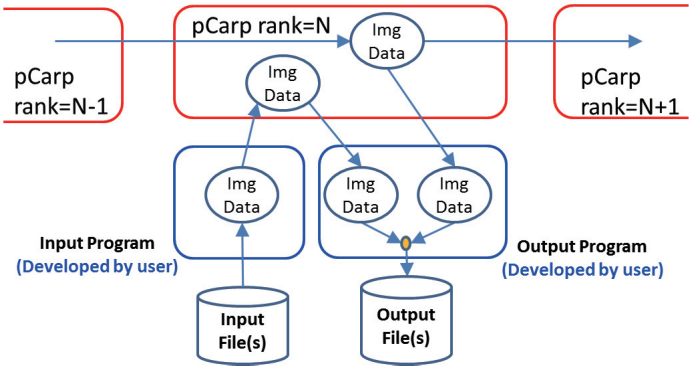


Figure 16 Decoupling Architecture of *pCarp*

```
main( argc, argv ) {

  ReadDataFile( data, size, &datacnt );

  for( i=0; i<datacnt; i++ ){
    carp_put_datasize( size[i] );
    carp_write( &data[i], size[i] );
  }
}
```

Input program

```
main( argc, argv ) {
  for( ... ) {
    carp_get_datasize( &size0 );
    carp_read( &data0, size0 );
    carp_get_datasize( &size1 );
    carp_read( &data1, size1 );
    result = kernel_code( data0, data1 );
    OutputData( result );
  }
}
```

Output program

Figure 17 Code skeletons of the input and output processes

Figure 16 shows the decoupled architecture of the image analyzing software for XFEL using *pCarp*. Each MPI process (*pCarp* process) creates two sub-processes by using the POSIX popen I/O

function, one to read a file and the other to process two data records and output its result to a file. Figure 17 shows the code skeletons of the input and output processes in Figure 16, the function having bold face names are the function provided by the *pCarp* framework. The input sub-process reads a file and passes read records to the parent *pCarp* process. In the *pCarp* process, records are preserved in memory and sent to its neighbor process as shown in Figure 15. The output sub-process reads two records from parent *pCarp* process and then computes those records. Finally its result is output. This procedure is repeated until the all data of the desired all-to-all computation is done. In those input and output programs, there is no need of calling the complex MPI functions at all.

In the evaluation of current prototype of *pCarp*, the execution time is much larger than the original program depending on the data size. The most of the additional time of *pCarp* comes from the pipe transmission. Improving this data transmission performance is our future work.

3.5. File Composition Library

We have been developing a user-level file I/O library, called file composition library, to reduce the heavy load on both the metadata and object storage servers of a parallel file system. The file composition library is assumed to use the SPMD (Single Program Multiple Data) execution model. In other words, it is assumed that all processes access each own file and issue the same I/O operations. The heavy load in the metadata is caused by issuing file open/create/close operations by a large number of processes. The file composition library gathers all files created by a job and make them a single file so that the metadata accesses are reduced. To mitigate the heavy load in the object storage, the file composition library limits the number of processes accessing a parallel file server simultaneously. In FY 2014, based on the experiences of developing the file composition library, a new file I/O library, called ftar (Fragmented tar), was designed. Unlike the file composition library, ftar uses the tar format. An ftar file keeps the tar format, but each file may be stored in fragments. Figure 18 shows an example of ftar file in which two files, f1 and f2, are stored in fragments.

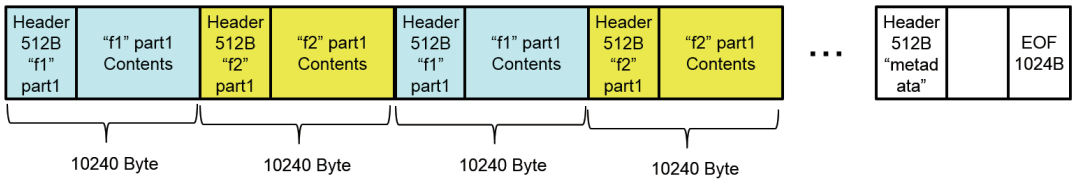


Figure 18 An Example of Ftar format

4. Schedule and Future Plan

- Communication Library

  Cached-Multi-W implementation on MPI_Neighbor_alltoallw primitive will be enhanced and

evaluated. This approach will also be applied to MPICH one-sided implementation using MPI_Put and MPI_Win_fence primitives.

- New Process / Thread Model

  Integratation the proposed task model with the McKernel which is under development by AICS System Software Development team is planned. Also, we are doing a collaborative research with ANL on the User-level process which was developed in last FY2013 to enhance the performance of irregular applications.

- Fault Resilience

  Based on the investigation in FY2014, we will start developing a framework to allow user applications to be fault-resilient easily.

- File I/O

  pCarp will be enhanced and distributed as open source. As mentioned in section 3.4. , the most of overhead of pCarp comes from data transmission using pipe. In order to reduce this overhead, we will try to implement pCarp with shared memory.

  The scalable MPI-IO will be adapted to the FEFS file system used in the K computer. Although the scalable MPI-IO implementation manages data exchanges in the two-phase I/O optimization using MPI_Isend and MPI_Irecv among MPI processes, we will try to apply MPI_Alltoallv for data exchanges as an alternative implementation for further performance improvement. I/O throttling scheme will be also examined in the collective data exchanges.

  Ftar will be developed and distributed as open source.

5. Publication, Presentation and Deliverables

(1) Journal Papers

1. 島田 明男, 堀 敦史, 石川 裕, 新しいタスクモデルによるメニーコア環境に適したMPI ノード内通信の実装, 情報処理学会論文誌, 情報処理学会, volume 56, 2015.

(2) Conference Papers

1. Yuichi Tsujita, Atsushi Hori, Yutaka Ishikawa: “Locality-Aware Process Mapping for High Performance Collective MPI-IO on FEFS with Tofu Interconnect,” In Proceedings of the 21th European MPI User’s Group Meeting, Workshop on Challenges in Data-Centric Computing, ACM, 2014.
2. Yuichi Tsujita, Atsushi Hori, Yutaka Ishikawa: "Affinity-Aware Optimization of Multithreaded Two-Phase I/O for High Throughput Collective I/O," In Proceedings of International Conference on High Performance Computing & Simulation, HPCS 2014, IEEE, 2014.
3. Yuichi Tsujita, Kazumi Yoshinaga, Atsushi Hori, Mikiko Sato, Mitaro Namiki, Yutaka

Ishikawa: "Multithreaded Two-Phase I/O: Improving Collective MPI-IO Performance on a Lustre File System," 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, 2014.

4. Yuichi Tsujita, Kazumi Yoshinaga, Atsushi Hori, Mikiko Sato, Mitaro Namiki, Yutaka Ishikawa: "Improving Parallel I/O Performance Using Multithreaded Two-Phase I/O with Processor Affinity Management," PPAM 2013 Revised Selected Papers, Lecture Notes in Computer Science, Vol. 8384, Springer, pp. 714-723, 2014.
5. Balazs Gerofi, Akio Shimada, Atsushi Hori, Takagi Masamichi, Yutaka Ishikawa: "CMCP: A Novel Page Replacement Policy for System Level Hierarchical Memory Management on Many-cores," In Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, ACM, 2014.
6. Mikiko Sato, Go Fukazawa, Akio Shimada, Atsushi Hori, Yutaka Ishikawa, Mitaro Namiki, “Design of Multiple PVAS on InfiniBand Cluster System Consisting of Many-core and Multi-core,” In Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014.
7. Atsushi Hori, Kazumi Yoshinaga, Atsushi Tokuhisa, Yasumasa Joti, Kensuke Okada, Takashi Sugimoto, Mitsuhiro Yamaga, Takaki Hatsui, Makina Yabashi, Yuji Sugita, Yutaka Ishikawa, Nobuhiro Go, “Decoupling Architecture for All-to-all Computation,” In Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014.
8. 畑中正行, 堀敦史, 石川裕, 京 Tofu における MPI-3.0 隣接集団通信の実装と評価, 情報処理学会, プログラミング 7(5), 2014.
9. 山口 訓央, 高木 将通, 畑中 正行, 堀 敦史, 石川 裕, HPC 向け高可搬通信ライブラリの設計と評価, 情報処理学会, 2014-HPC-145, No. 15, 2014.
10. 島田 明男, 堀 敦史, 石川 裕, 新しいタスクモデルによる MPI ノード内通信の高性能化, 情報処理学会, 2014-OS-130, No. 18, 2014.
11. 吉永一美,亀山豊久,堀敦史,石川裕,エクサスケールでの耐故障性実現に向けた代替ノード配置による通信性能の評価, 情報処理学会, 2014-HPC-144, No.16, 2014.
12. 吉永一美,亀山豊久,畑中正行,堀敦史,石川裕, 代替ノード利用手法による耐故障性実現に向けた通信性能の評価と検討, 情報処理学会, 2014-HPC-145, No.6, 2014.
13. 吉永一美,亀山豊久,堀敦史,石川裕,予備ノードを利用した故障後の実行継続手法の検討と評価, 情報処理学会, 2014-HPC-147, No.21, 2014.
14. 畑中正行, 堀敦史, 石川裕, 京 Tofu における隣接集団通信の袖通信最適化, 情報処理学会, 2015-HPC-148(35), 2015.
15. 辻田　祐一，堀　敦史，石川　裕，FEFS におけるストライピング処理を考慮した集団型 MPI-IO の実装，情報処理学会，2014-HPC-145，No. 35, 2014.
16. 辻田　祐一，堀　敦史，石川　裕，集団型 MPI-IO の高速化に向けた I/O リクエスト発行方式の最適化，情報処理学会,2014-HPC-146，No. 17, 2014.

17. User-level Process towards Exascale Systems (Akio Shimada, Atsushi Hori, Yutaka Ishikawa, Pavan Balaji), In 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, 一般社団法人情報処理学会, volume 2014, 2014.
18. メニーコアクラスタにおけるジョブ間の性能干渉について (堀 敦史, 亀山 豊久, 並木 美太郎, 石川 裕), 情報処理学会研究報告. HPC 研究会報告, 一般社団法人情報処理学会, volume 2014, 2014.

(3) Invited Talks

(4) Posters and presentations

1. Akio Shimada, Atsushi Hori, Yutaka Ishikawa, “Eliminating Costs for Crossing Process Boundary from MPI Intra-node Communication,” In Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014.

(5) Patents and Deliverables

Open Source Software Packages (http://www.sys.aics.riken.jp/releasedsoftware/index.html)

1. PRDMA (for the K computer)
2. GDB for McKernel
3. PVAS, M-PVAS and Agent (for the x86 and Xeon Phi CPUs)
4. sCarp and pCarp (for the K computer, FX10s and Linux clusters)