System Software Research Team

1. Team members

Yutaka Ishikawa (Team Leader)

Atsushi Hori (Senior Scientist)

Keiji Yamamoto (Postdoctoral Researcher)

Akio Shimada (Research Associate)

Yoshiyuki Ohno (Research Associate)

Masayuki Hatanaka (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, power-aware, and fault-aware operating system for next-generation supercomputers based on many core architectures.

Parallel file I/O is one of the scalability issues in modern supercomputers. One of the reasons is due to heavy metadata accesses. If all processes create and write different files, the metadata server receives so many requests by all processes not only at the creation time but also at writing data to each file. Three approaches have been conducted to mitigate this issue. One approach is to introduce a file composition technique that gathers multiple data generated by an application and stores these data into one or a few files in order to reduce the number of files accessed by processes. Another approach is to provide multiple metadata server in which the requests for metadata are sent to a metadata server resolved using hash function. The third approach is to provide a smart MPI-IO implementation for applications using MPI-IO functions.
Increasing number of cores and nodes enforces strong scaling on parallel applications. Because the ratio of communication time against local computation time increases, a facility of low-latency and true overlapping communication and computation communication is desired. A communication library, integrated to the MPI library implementation in K computer, has been designed and implemented, that utilizes DMA engines of K computer. Each compute node of K computer has four DMA engines to transfer data to other nodes. If a communication library knows communication patterns in advanced, it may utilize the DMA engines. Indeed, the feature of MPI persistent communication allows the runtime library to optimize data transfers involved

in the persistent communication using the DMA engines.

System software stack developed by our team is designed not only for special dedicated supercomputers, but also for commodity-based cluster systems used in research laboratories. The system will be expected to be used as a research vehicle for developing an exascale supercomputer system.

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 includes the multiple stencil computations for weather and climate models. 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. Also, it 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. For neighborhood communication on the Tofu interconnect, there are major two scheduling issues; (1) load imbalance and (2) network resource contentions. In the former case, the loads between RDMA engines may be imbalance if the transfer requests are distributed to RDMA engines in round-robin fashion. In the latter case, the network link is congested if a network link is shared by some outgoing messages at the same time. Also, Contention of the RDMA engine in the receiver side occurs if some incoming messages try to request the same RDMA engine of the receiver side at the same time. Especially, the sender must specify the receiver-side RDMA engine explicitly in the RDMA transfer request of the Tofu interconnect. However, it is not clear how to distribute the receiver-side RDMA engines among senders.

Proposed Scheduling Algorithm

One of the straightforward ways to solve these scheduling issues is to model as a two-dimensional strip-packing problem. For simplicity and speed, we selected the Bottom-Left heuristic algorithm to solve this packing problem. The Bottom-Left algorithm sorts the input RDMA commands (transfer requests), and packs them at the open bottom-most and left-most position in the scheduling table, such as (a), (b), and (c) in Figure 1. In addition, we added a constraint to avoid the network resource contentions into the basic Bottom-Left algorithm. This constraint assures that two or more RDMA commands using a same network link are not scheduled at the same time. So, in the Figure 1 (d), a rectangle indicates an RDMA command, and the rectangles with the same color indicates that these commands use the same network link. The command #4 is scheduled on the RDMA engine #2 because the contention happens if it is scheduled on the RDMA engine #3.

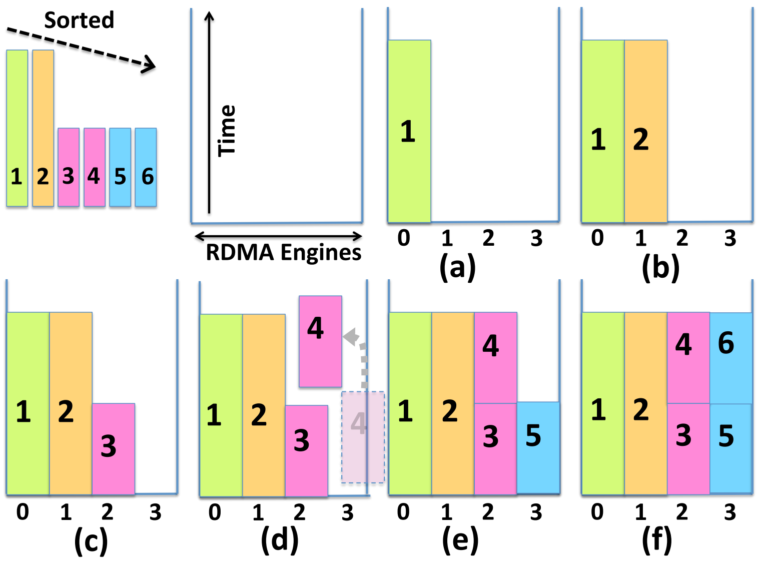


Figure 1. Scheduling by proposed Modified Bottom-Left algorithm.

Implementation and Evaluation

The proposed scheduler has been implemented and integrated into the persistent communication primitives in Open MPI on the K computer. This implementation is called PRDMA (Persistent Remote Direct Memory Access).

We measured three implementations in ghost cell exchange of SCALE-LES3: Original PC, Round-Robin, and Modified Bottom-Left. The first two implementations were measured to compare to proposed Modified Bottom-Left implementation. The first one, called Original PC, is the default Open MPI based implementation provided on the K computer. And the second one, called Round-Robin, is the same as Modified Bottom-Left except the scheduler uses a simple Round-Robin algorithm similar to Original PC. In Figure 2, the horizontal axis shows the aggregate transfer size in a ghost cell exchange corresponding to the grid size in SCALE-LES3. The vertical left axis shows the elapsed communication time for the forty one thousands exchanges, and vertical right axis shows the percentage change against the first two implementations. The Round-Robin is 20 to 70 % better than Original PC in communication time. The Modified Bottom-Left is up to 32 % better than Round-Robin and about 50 % better than Original PC.

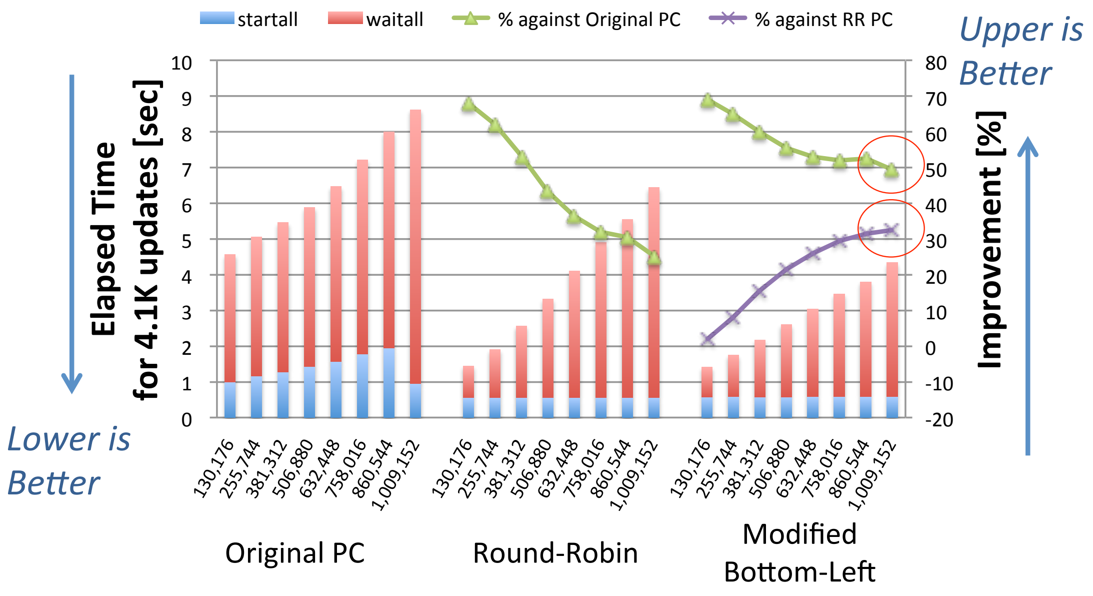


Figure 2. Evaluation of Three Scheduler implementations in Ghost Exchange.

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 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 3 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.

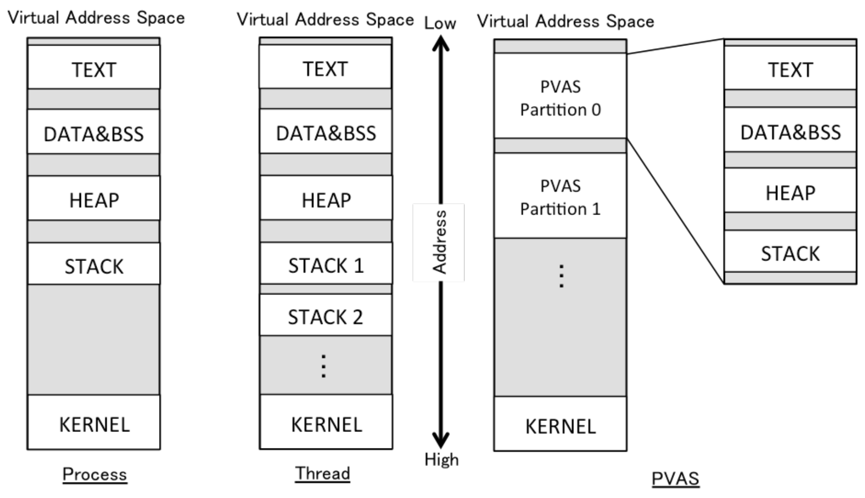


Figure 3. Semantics view of the conventional process model and PVAS.

User-level Process

Process oversubscription, which invokes a larger number of parallel processes within a computing node than the number of available CPU cores, results in better load-balance and latency hiding, and then it may improve the performance of multi-process programs. In this case, parallel processes within a computing node must be scheduled by the OS kernel. However, lightweight OS kernels that are developed for exascale systems may no longer support task scheduling, because it is one of the most resource consuming and noisy operation. In this situation, only one parallel process per CPU core is allowed, and then process oversubscription is impossible. To solve this problem, we developed user-level process. User-level process is a "process" that can be scheduled in user-space, then multiple user-level processes per CPU core can be allowed without help of OS task scheduler as described in Figure 4. Multi-process programs running on the lightweight OS kernels for exascale systems will be able to do process oversubscription by binding one parallel process to one user- level process even if those OS kernel does not support task scheduling.

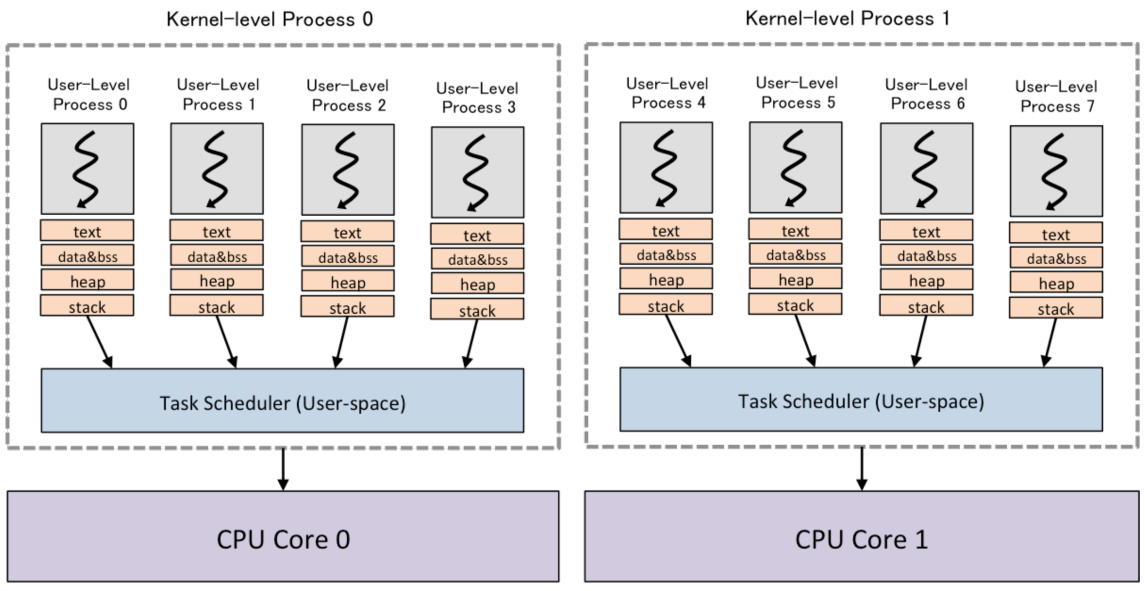


Figure 4. Semantic view of the User-level Process.

The use of user-level process also provides favorable side effects to the process oversubscription. When doing process oversubscription, the performance of context switch between parallel processes affects the overall performance of the program, because so much number of parallel processes may be invoked within a computing node. The context switch between user-level processes is faster than the context switch between traditional processes, because it is operated in user-space. As shown in Figure 5, the preliminary evaluation results show that the context switch between user-level processes is approximately 2.6 times faster than the context switch between traditional processes in the best case.

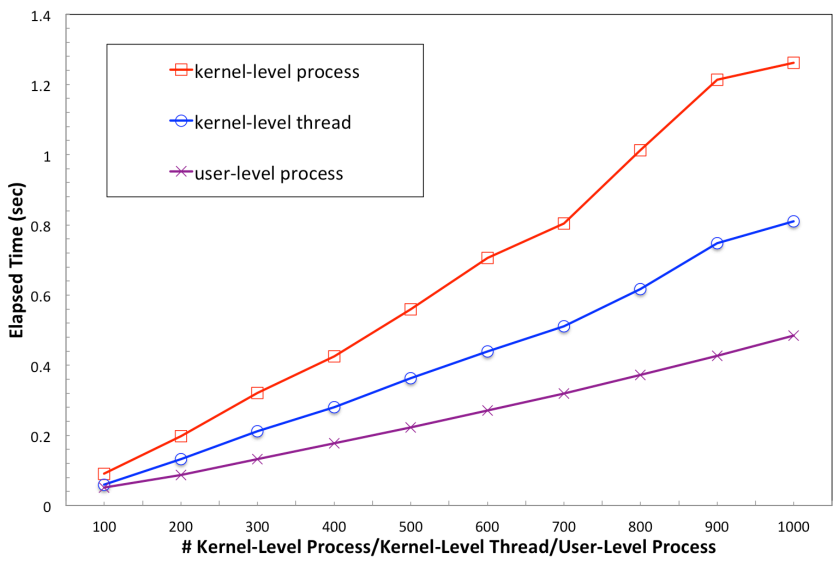
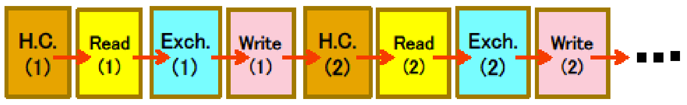


Figure 5. Elapsed time for operating 1000 times context switches.

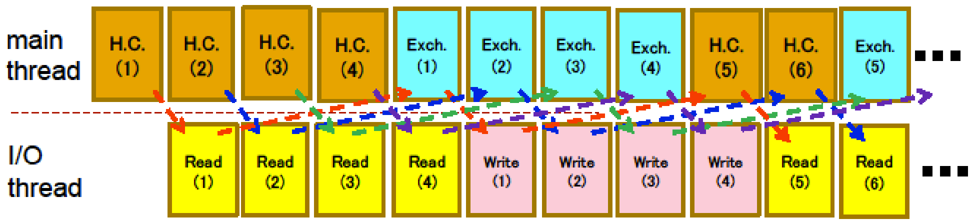
3.3. Scalable MPI-IO Using Multithreaded Scheme

A commonly used MPI-IO library named ROMIO has the two-phase I/O (TP-IO) scheme to improve collective I/O for non-contiguous accesses. This research is addressing to have a multithreaded scheme in the TP-IO in order to achieve higher performance than the original one in collective I/O for non-contiguous accesses.

In the FY2013, ROMIO in the MPICH2 library was arranged to have multithreaded operations using Pthreads. Figure 6 shows multithreaded the TP-IO operation scheme in parallel with its original scheme.



(a) Original TP-IO scheme



(b) Multithreaded TP-IO scheme

Figure 6. Original and multithreaded TP-IO in collective write operations.

Here "H.C." and "Exch." stand for hole check and data exchange phases, respectively, while "Read" and "Write" denote file read and write operations, respectively. Since every phase is processed independently in terms of access regions (described such as (1), (2) under each phase name), we can overlap read and write phases with hole check and data exchange phases as shown in Figure 6(b) using multithreaded way.

Figure 7 shows a functional diagram of the multithreaded implementation. Every MPI process invokes an I/O thread which plays file I/O using the pthread_create function. In processing I/O requests from user's MPI programs, I/O requests generated on every main thread are enqueued in the read queue in the hole check phase, followed by read operations on an I/O thread by dequeueing I/O requests from the read queue, and so on. Finally in the write phase, I/O requests are enqueued into the hole queue, then it will initiates the next hole check phase and so on. These operations are repeated until all the target regions are accessed. With the help of the multithreaded way, ideally we can achieve at most twice the original TP-IO throughput. For further performance improvements, the multithreaded scheme introduced multiple request slots in queues. The number of slots can be managed by a key-value pair in an MPI_Info object using the MPI_Info_set function as well as the size of collective buffer size.

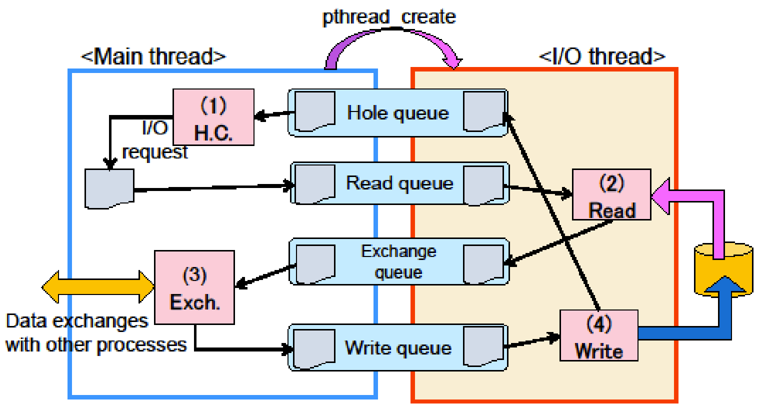
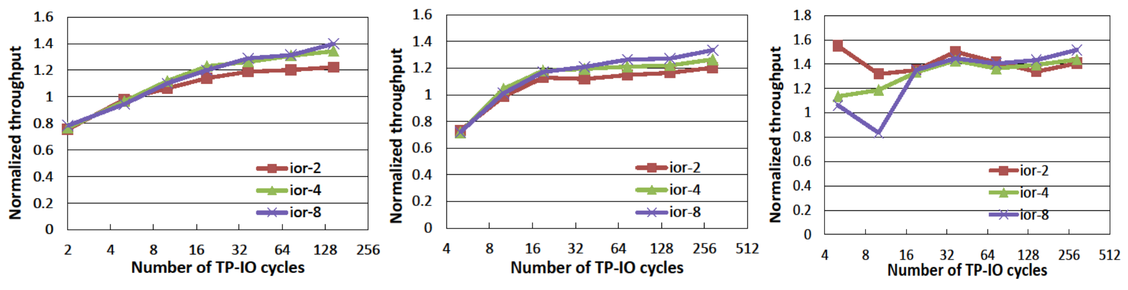


Figure 7. Functional diagram of multithreaded TP-IO.

Performance evaluation was carried out on a PC cluster system at the University of Tokyo, which is named as T2K. We used 64 PC nodes, which have 4 AMD Opteron 2.3GHz processors with 32 GiB memory using Linux kernel 2.6.18. Interconnections between PC nodes are Myrinet10G for MPI communications and the Lustre file system and Gigabit Ethernet for control. I/O operations were carried out on its Lustre filesystem version 1.8.9, which has 30 OSTs in total. In this evaluation, all the OSTs were utilized. We used HPIO benchmark, which supports unique derived data type generations required for non-contiguous I/O. ROMIO in MPICH2 version 1.4.1p1 with our multithreaded extension was evaluated. Figure 8 shows normalized write throughput relative to the original TP-IO with 64, 128, and 256 MPI processes.



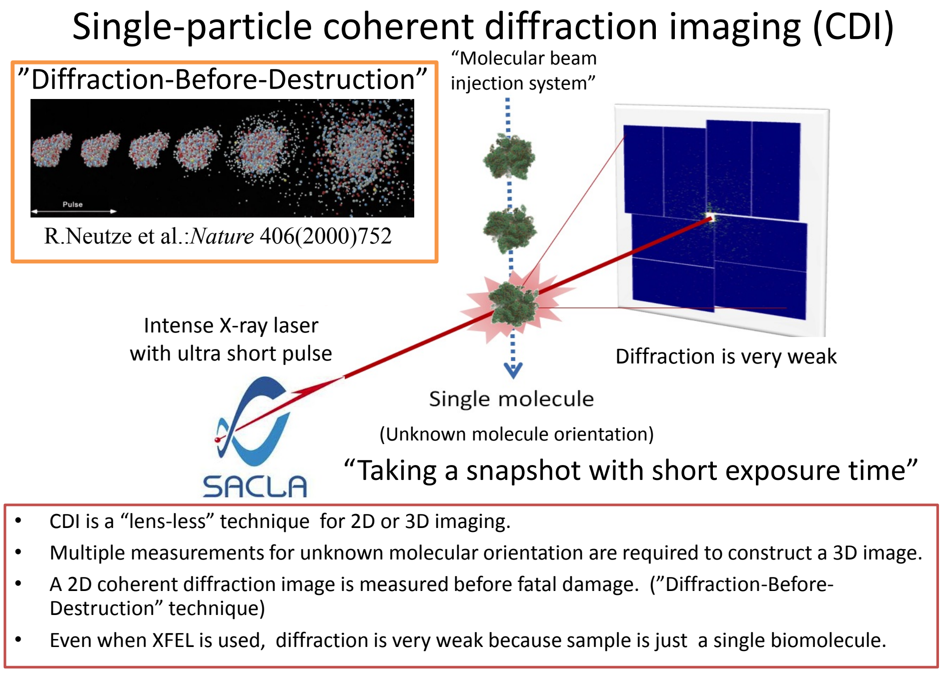
(a) 64 MPI processes (b) 128 MPI processes (c) 256 MPI processes

Figure 8. Normalized write throughput of multithreaded TP-IO relative to the original one.

In every case, data block was configured with 256B data blank in each 488B data region, and absolutely 1 GiB data by multiple data blocks were generated at every MPI process. File system cache was disabled by remounting Lustre file system prior to every MPI-IO operation run. We also changed the number of slots in queues as 2, 4, and 8, which are indicated as ior-2, ior-4, and ior-8, respectively in this figure. Horizontal axis of this figure stands for the number of TP-IO cycles, thus lower value means the larger collective buffer, while the larger value corresponds to smaller collective buffer size. Through this evaluation the multithreaded TP-IO achieved up to 60% improvements relative to the original one. Higher number of slots in queues led to higher performance gain, however it utilized larger memory resources. However, we can minimize utilized memory resources using this multithreaded scheme because this evaluation shows higher gains when we had around from 64 to 128 TP-IO cycles. It is also noticed that smaller number of TP-IO cycles led to smaller performance gain or performance degradation. This was due to imperfect overlap due to larger collective buffer size. Thus the multithreaded scheme was not effective in such case. However, such case needs larger memory resources, so the multithreaded scheme can have higher throughput gains without increasing memory utilization for collective buffer.

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. Figure 9 and Figure 10 show an outline indicating how a molecule 3D structure can be reproduced from images obtained by XFEL. Each image size is around 20 Mbytes, but may vary depending on the resolution of image sensor. The number of images required to develop a 3D structure of a molecule can be millions, resulting 20 PBytes of data size in total. Further, each image is classified into thousands of images to have every possible snapshot orientations and to reduce the quantum noise.


Single-particle coherent diffraction imaging (CDI)
"Diffraction-Before-Destruction"
Pulse
R.Neutze et al.:*Nature* 406(2000)752
"Molecular beam injection system"
Intense X-ray laser with ultra short pulse
Diffraction is very weak
Single molecule
(Unknown molecule orientation)
SACLA
"Taking a snapshot with short exposure time"

- CDI is a "lens-less" technique for 2D or 3D imaging.
- Multiple measurements for unknown molecular orientation are required to construct a 3D image.
- A 2D coherent diffraction image is measured before fatal damage. ("Diffraction-Before-Destruction" technique)
- Even when XFEL is used, diffraction is very weak because sample is just a single biomolecule.

Figure 9. How SACLA XFEL works.

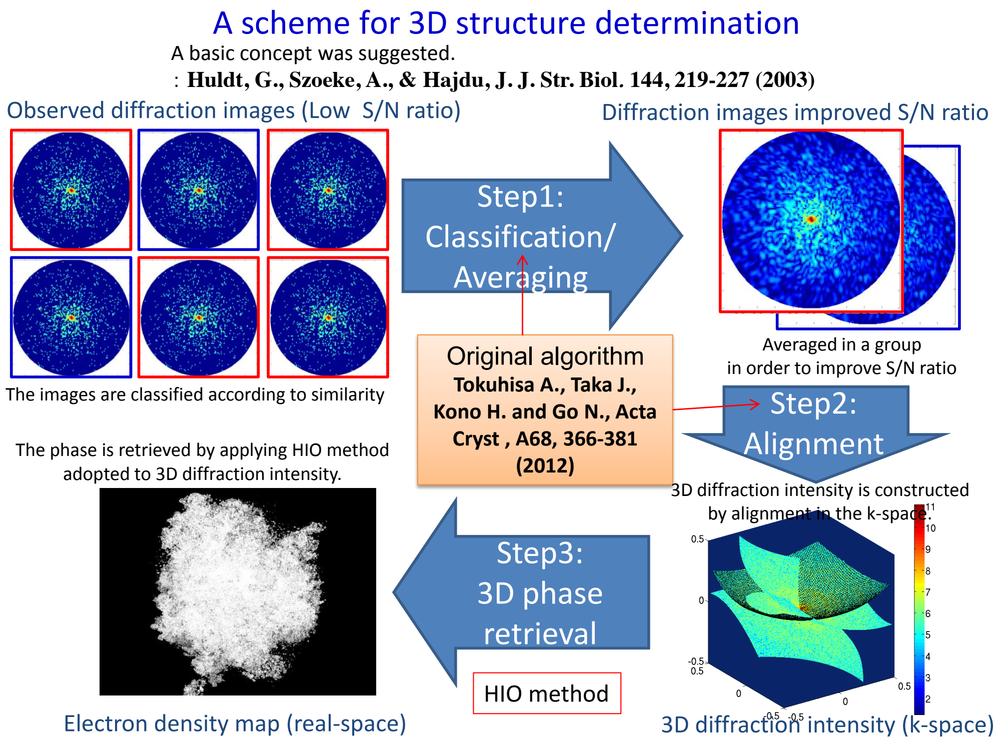

A scheme for 3D structure determination
A basic concept was suggested.
: Huldt, G., Szoeke, A., & Hajdu, J. J. Str. Biol. 144, 219-227 (2003)
Observed diffraction images (Low S/N ratio)
Diffraction images improved S/N ratio
Step1: Classification/ Averaging
The images are classified according to similarity
Averaged in a group in order to improve S/N ratio
Original algorithm
Tokuhisa A., Taka J., Kono H. and Go N., Acta Cryst , A68, 366-381 (2012)
Step2: Alignment
The phase is retrieved by applying HIO method adopted to 3D diffraction intensity.
3D diffraction intensity is constructed by alignment in the k-space.
Step3: 3D phase retrieval
HIO method
Electron density map (real-space)
3D diffraction intensity (k-space)

Figure 10. Obtaining electron density map from XFEL snapshots.

The developed software consists of two parts. One is to select representative images and another is to classify images using selected representative images as shown in. The classification is conducted by several FFT operations on each image to have correlation. The order of the number of this FFT operation to select thousands classification images is $O(M^2)$, followed by the classification of the rest of the images of $O(N*M)$ FFT operations, where M is the number of classification and N is the number of images. SACLA XFEL is going to produce 30 images in a second and the time to take one million images takes approximately 9 hours. There can be the cases where the snapshots are not well enough quality to analyze. In this case, the experiment must be stopped and tuned to obtain good quality images. Thus the image analysis must be done as soon as possible. This heavy computation, one million images should be analyzed as soon as possible, requires the power of the K computer. The Data Acquisition team at SCALA has been developing a classification algorithm, while the System Software Research team at AICS has been in charge of parallelization, performance tuning, and I/O.

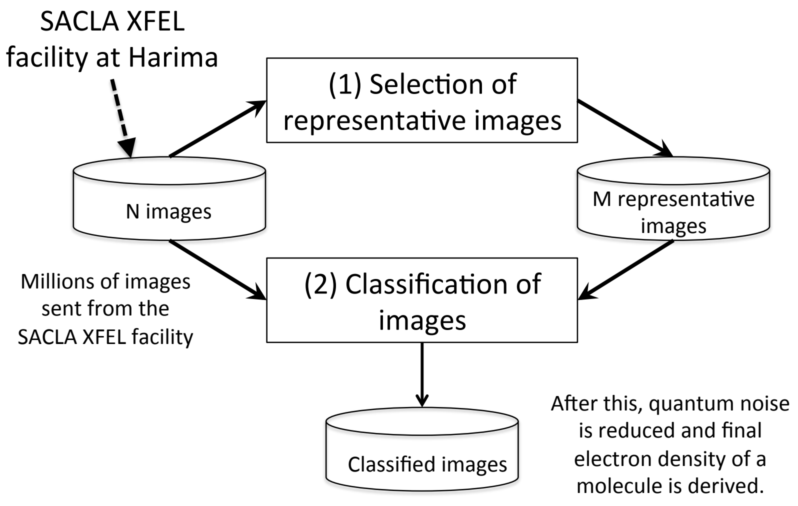

SACLA XFEL facility at Harima
(1) Selection of representative images
N images
M representative images
Millions of images sent from the SACLA XFEL facility
(2) Classification of images
Classified images
After this, quantum noise is reduced and final electron density of a molecule is derived.

Figure 11. Block diagram of the procedure running on the K computer.

In FY2013, we optimized classification software developed in FY2012. In order to realize effective parallelization, our software keeps load balance between processes and reduces file input time by minimizing total size of the input from storage.

Figure 12 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 12 are images which read from storage, and the greens are passed from neighbor process. The calculation is finished at N_p step, where N_p is number of processes. In addition, the software can utilize rank directory of K computer to reduce read time and increase the scalability (as shown in Figure 13). By using the software, classification for one million images can be finished in an hour on K computer.

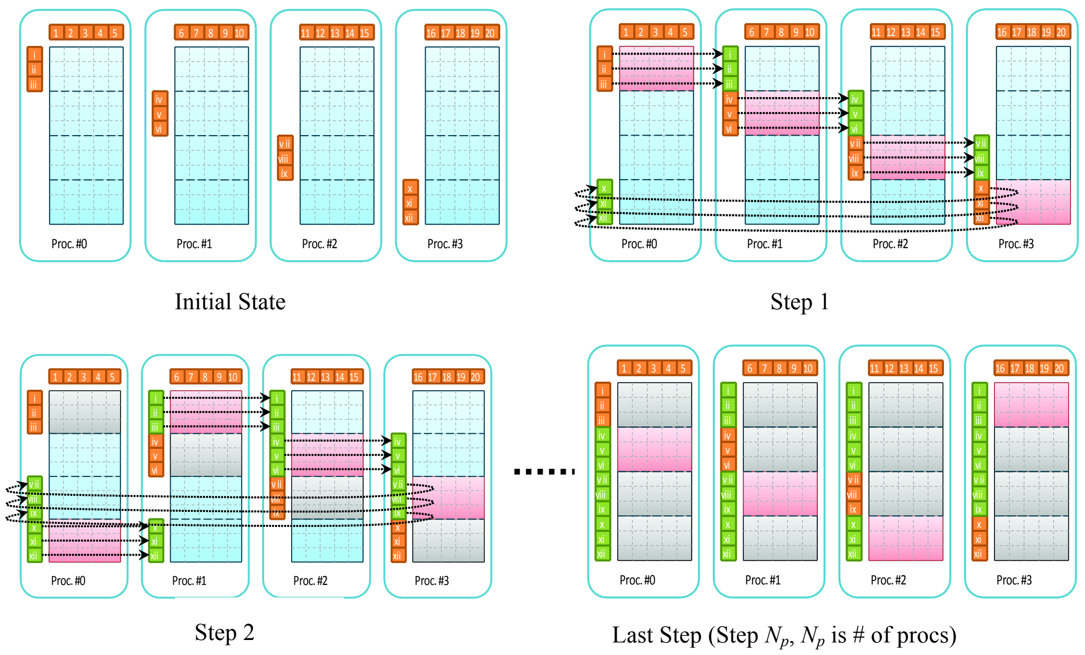


Figure 12. Calculation workflow for classification of images.

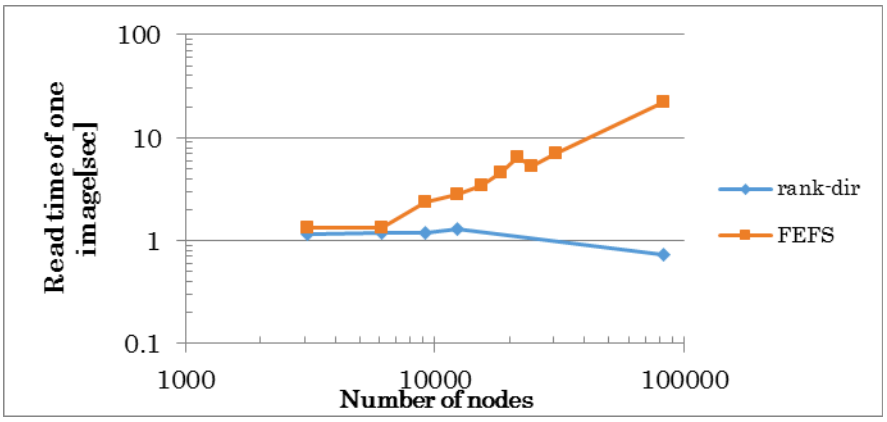


Figure 13. Read time of one image on the K computer.

3.5. File Composition Library

I/O system resources such as Meta-Data Server (MDS), Object Storage Server (OSS) and Object STorage (OST) are shared across the processes of a single parallel job. As HPC systems become large, the amount of I/O requests to each I/O system resource becomes larger, and load of them becomes heavier. The heavy loads on I/O system resources cause I/O performance degradation, and application performance degradation. It is one of the scalability issues of leadership-class high performance computing systems. We targeted the case that each process of the parallel job creates its own file and writes the file. Many parallel applications adopt this I/O pattern. Figure 14 shows the I/O performance of the K computer over the number of client processes, varying the striping counts and striping sizes (for example, "C1_S16m" indicates the striping count is set to one and striping size is set to 16MiB). As in this figure, the bandwidth tends to decrease when the number of client processes increases. Thus, the idea of proposed "File Coordination Library" is to limit the number of client processes accessing a parallel file server simultaneously.

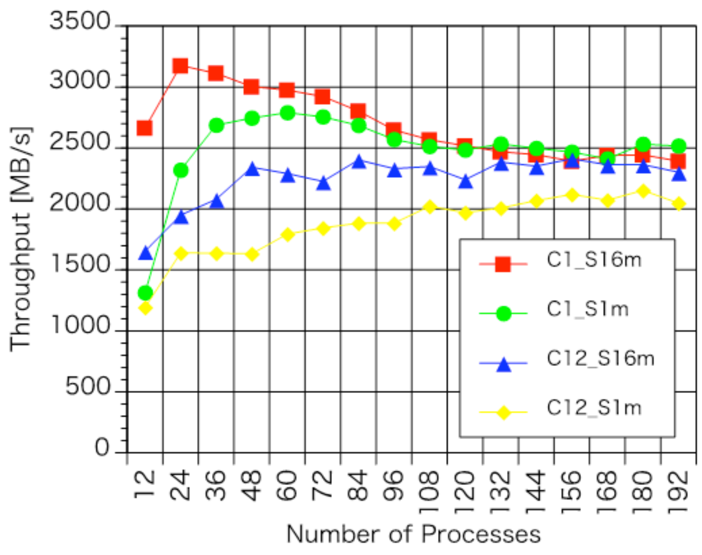


Figure 14. Write Performance over the number of client threads.

Figure 15 shows the I/O performance of the K computer with the proposed file coordination library (right graph) and without it (left graph). As shown in this figure, the average performance with the file coordination technique exhibits about 20% better.

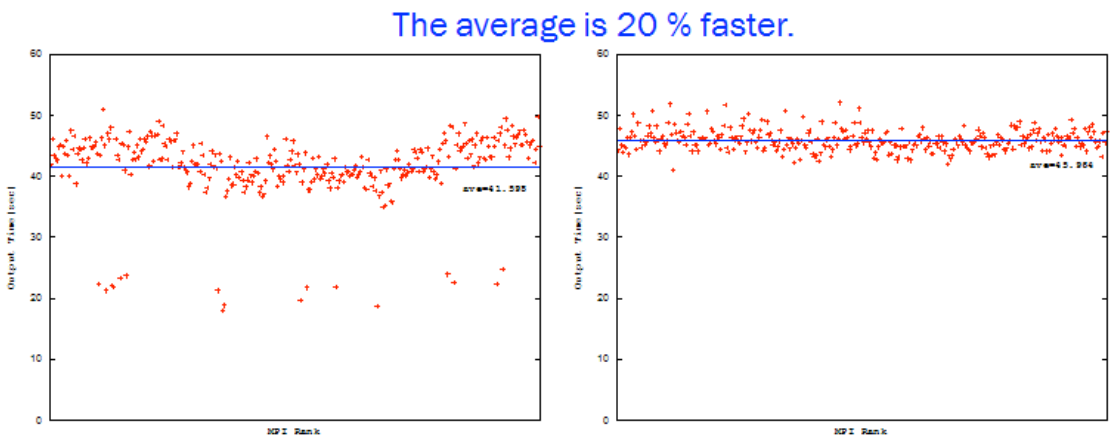


Figure 15. Write throughput of FClib.

4. Schedule and Future Plan

- Communication Library

  PRDMA (Persistent Remote Direct Memory Access) will be enhanced with a sophisticated data transfer scheduling algorithm, and it will be evaluated. MPICH3, an MPI implementation for the MPI-3 standard, will be continued to be ported to the K computer.

- File I/O

  The classification software, developed in FY2013, processes huge image files generated by SACLA XFEL. It will be open to the users of SACLA. Based on the experience on the design of the classification software, general file I/O functions for other classification applications will be designed. The scalable MPI-IO will be adapted to the FEFS file system used in the K computer.

- New process/thread Model

  A new process/thread model, PVAS and its user-level thread, has been designed and implemented in FY2013. In FY2014, an MPI runtime environment on PVAS will be implemented and evaluated.

5. Publication, Presentation and Deliverables

(1) Journal Papers

[1] High-speed classification of coherent X-ray diffraction patterns on the K computer for high-resolution single biomolecule imaging (Atsushi Tokuhisa, Junya Arai, Yasumasa Joti, Yoshiyuki Ohno, Toyohisa Kameyama, Keiji Yamamoto, Masayuki Hatanaka, Balazs Gerofi, Akio Shimada, Motoyoshi Kurokawa, Fumiyoshi Shoji, Kensuke Okada, Takashi Sugimoto, Mitsuhiro Yamaga, Ryotaro Tanaka, Mitsuo Yokokawa, Atsushi Hori, Yutaka Ishikawa, Takaki Hatsui, Nobuhiro Go), In Journal of Synchrotron Radiation, volume 20, 2013.

[2] A Hybrid Operating System for a Computing Node with Multi-Core and Many-Core Processors (M. Sato, G. Fukazawa, K. Yoshinaga, Y. Tsujita, A. Hori, M. Namiki), In International Journal Advanced mputer Science (IJACSci), volume 3, 2013.

(2) Conference Papers

[3] Multithreaded Two-Phase I/O: Improving Collective MPI-IO Performance on a Lustre File System (Yuichi Tsujita, Kazumi Yoshinaga, Atsushi Hori, Mikiko Sato, Mitaro Namiki, Yutaka Ishikawa), In Proceedings of PDP2014, Turin, February 12-14, 2014, IEEE CS, 2014.

[4] 京での大量データの並列相関計算を支援するソフトウェアの提案（吉永一美，徳久淳師，大野善之，亀山豊久，堀敦史，城地保昌，初井宇記，石川裕），In 情報処理学会研究報告．[ハイパフォーマンスコンピューティング]，一般社団法人情報処理学会，volume 2013, 2013.

[5] メニーコア混在型並列計算機向け大域仮想アドレス空間モデル Multiple PVAS の提案（深沢 豪，佐藤 未来子，吉永 一美，辻田 祐一，島田 明男，堀 敦史，並木 美太郎），In 情報処理学会第 141 回ハイパフォーマンスコンピューティング研究発表会，一般社団法人情報処理学会, volume 2013, 2013.

[6] RDMA スケジューリングによる MPI 通信の高速化（畑中正行，堀敦史，石川裕），In IPSJ

SIG Notes, Information Processing Society of Japan (IPSJ), volume 2013, 2013.

[7] メニーコア用 Agent プログラミング環境の提案 (Hori Atsuhi, Shimada Akio, Namiki Mitaro, Sato Mikiko, Fukazawa Go, Tsujita Yuishi, Ishikawa Yutaka), In IPSJ SIG Notes, Information Processing Society of Japan (IPSJ), volume 2013, 2013.

[8] ヘテロジニアス計算機上の OS 機能委譲機構 (佐伯 裕治, 清水 正明, 白沢 智輝, 中村 豪, 高木 将通, Balazs Gerofi, 思 敏, 石川 裕, 堀 敦史), Information Processing Society of Japan (IPSJ), volume 2013, 2013.

[9] メニーコア向けNUMA最適並列分散I/Oの予備検証 (小田和 友仁, 住元 真司, 堀 敦史, 石川 裕), Information Processing Society of Japan (IPSJ), volume 2013, 2013.

[10] 次世代高性能並列計算機のためのシステムソフトウェアスタック (石川 裕, 堀 敦史, Balazs Gerofi, 高木 将通, 島田 明男, 清水 正明, 佐伯 裕治, 白沢 智輝, 中村 豪, 住元 真司, 小田和 友仁), Information Processing Society of Japan (IPSJ), volume 2013, 2013.

[11] A Delegation Mechanism on Many-Core Oriented Hybrid Parallel Computers for Scalability of Communicators and Communications in MPI (Kazumi Yoshinaga, Yuichi Tsujita, Atsushi Hori, Mikiko Sato, Mitaro Namiki, Yutaka Ishikawa), In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE Computer Society, 2013.

[12] Proposing a new task model towards many-core architecture (Akio Shimada, Balazs Gerofi, Atsushi Hori, Yutaka Ishikawa), In Proceedings of the First International Workshop on Many-core Embedded Systems, ACM, 2013.

[13] Optimization of MPI Persistent Communication (Masayuki Hatanaka, Atsushi Hori, Yutaka Ishikawa), In Proceedings of the 20th European MPI Users' Group Meeting, ACM, 2013.

[14] Partially Separated Page Tables for Efficient Operating System Assisted Hierarchical Memory Management on Heterogeneous Architectures (Balazs Gerofi, Akio Shimada, Atsushi Hori, Yutaka Ishikawa), In CCGRID, 2013.

(3) Patents and Deliverables

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

[15] PRDMA (for the K computer)

[16] File Composition Library (for the K computer)

[17] GDB for McKernel

[18] PVAS, M-PVAS and Agent (for the x85 and Xeon Phi CPUs)