iv Reports on Research Activities

1. System Software Research Team

1.1. Team members

Yutaka Ishikawa (Team Leader)

Atsushi Hori (Senior Scientist)

Keiji Yamamoto (Postdoctoral Researcher)

Balazs Gerofi (Postdoctoral Researcher)

Akio Shimada (Research Associate)

Yoshiyuki Ohno (Research Associate)

Masayuki Hatanaka (Research Associate)

Toyohisa Kameyama (Technical Staff)

1.2. 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 exa-scale 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. Two 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. The other approach is to provide multiple metadata server in which the requests for metadata are sent to a metadata server resolved using hash function.

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 advance, it may utilize the DMA engines. Indeed, the feature of MPI persistent communication, standardized in MPI-1.0, allows the runtime library to utilize the DMA engines for data transfers involved in the persistent communication with restricted usage.

The 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 exa-scale supercomputer system. This is partially supported by JST CREST Post-Petascale research project.

1.3. Research Results and Achievements

1.3.1. 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 1 and Figure 2 show an outline indicating how a molecule 3D structure can be obtained from images obtained by XFEL. Each image size is around 20 Mbytes, but may vary depending on the resolution of image sensor. However, the number of images required to develop a 3D structure of a molecule is 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.

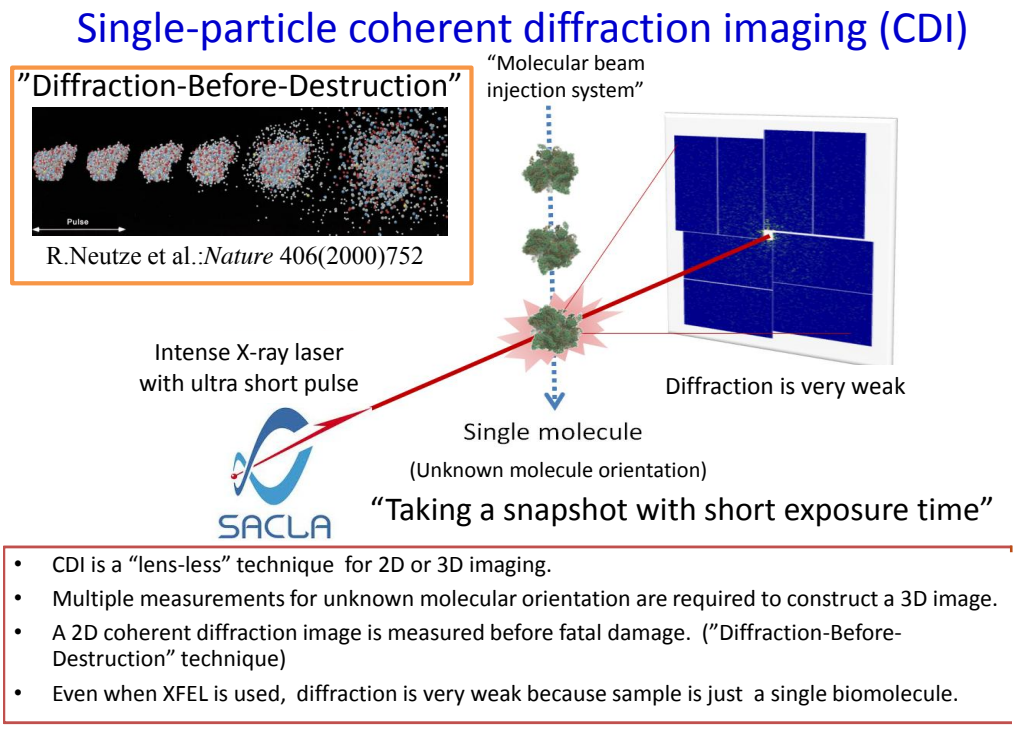

Figure 1 How SACLA XFEL works

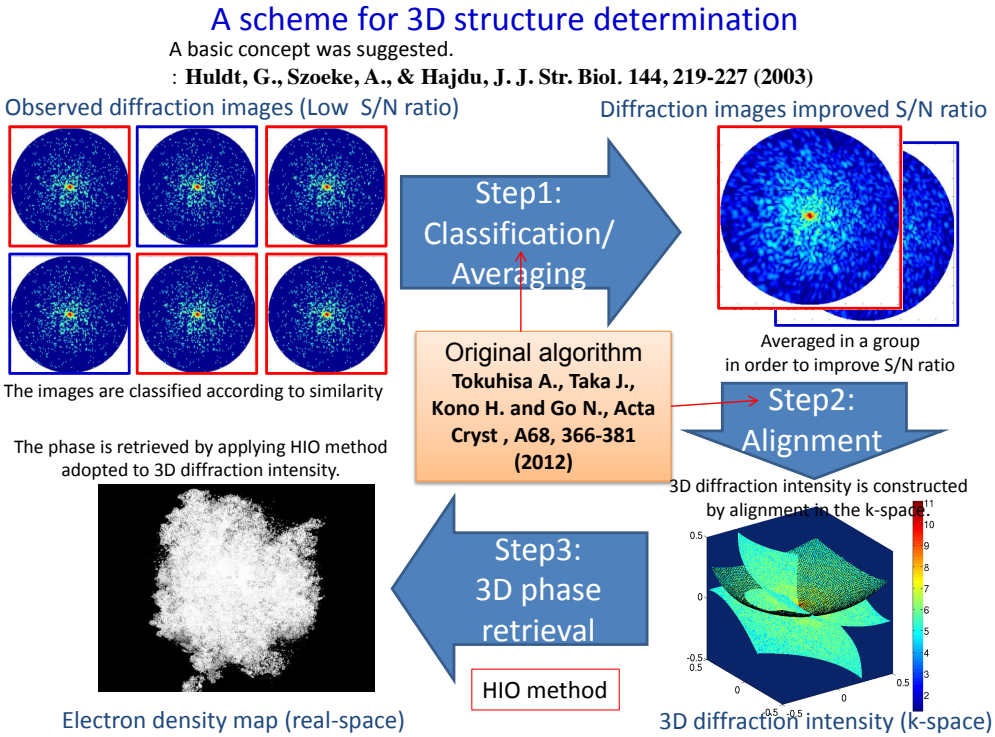

Figure 2 Obtaining electron density map from XFEL snapshots

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 Figure 3. 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. Since this is the first year of this project, the development of the software is the target. Performance tuning and I/O tuning are left in the following years.

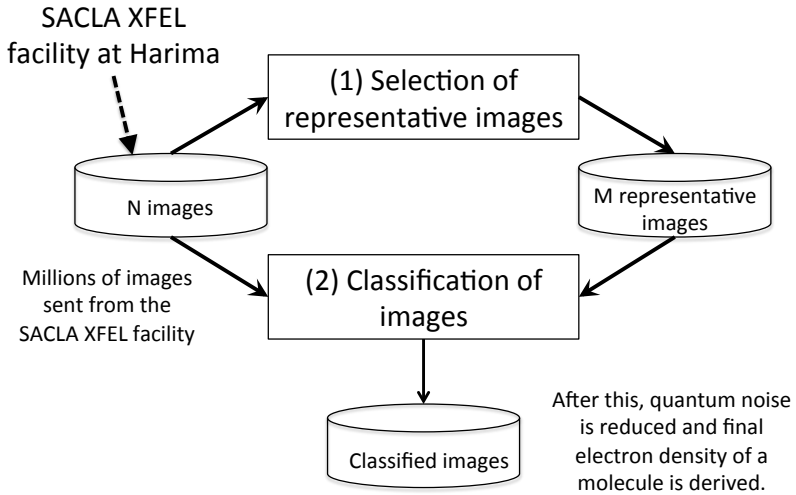

Figure 3 Block diagram of the procedure running on the K computer

1.3.2. File Composition Library

From FY2011, we have been developing a file I/O middleware called File Composition Library (FClib). FClib aggregates files created by parallel processes and stored to one of a few files. This results in reducing bottleneck of metadata operations on the metadata server.

In FY2012, FClib has been ported on the K computer based on the FY2011's implementation of commodity-based PC cluster. We measured the performance of FClib in case that each process creates/writes to one file into an aggregate file on shared directory. In order to compare with the regular file system provided by K computer, we measured the performance that each process creates/writes to one file to each rank directory. As shown in Figure 4, FClib performs 2.5 times faster than the original file I/O in the create operation, but 15-20 % of write throughput is dropped.

So various write access patterns have been measured to find out the source of performance degradation. It has been shown that if each compute node writes data to its associated directory, called rank directory, the write throughput is better than the case that writing data to a regular directory. This is because rank directories are provided by different file servers, and thus storage accesses are well distributed. We modified FClib so that each compute node accesses its rank directory. As a result, the write throughput of FClib is the same performance where each compute node accesses its rank directory (Figure 5).

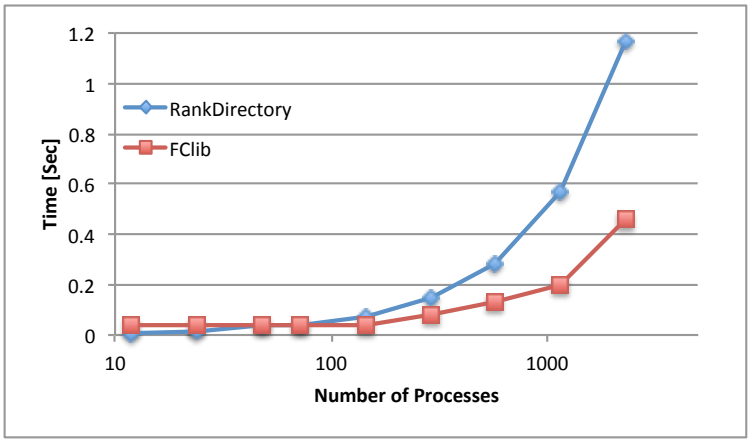

Figure 4 Elapsed time of file create

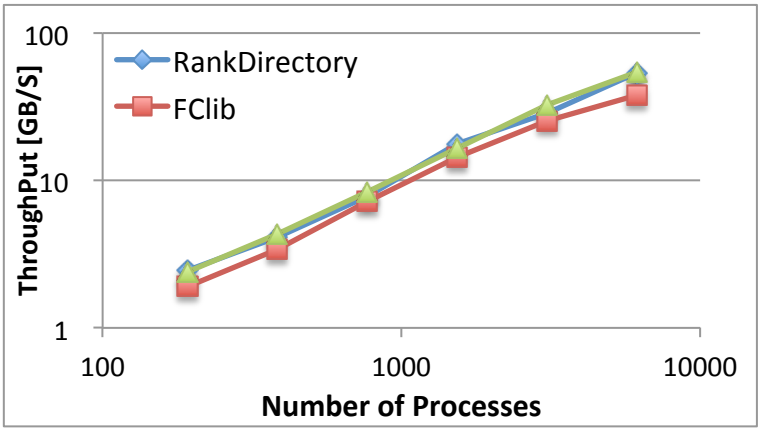

Figure 5 Write throughput of FClib

1.3.3. Hash-based Parallel File System

A parallel file system used in a modern supercomputer is basically composed of MDS (Metadata Server) and OSS (Object Storage Server). An MDS handles the file open, create and status operations. An OSS handles file read and write operations. Most current parallel file systems have only one MDS, and thus, such a system causes bottleneck of metadata operations requested by all compute nodes. A new scalable parallel file system based on a hash function has been designed and implemented from FY2011.

This system consists of multiple MDSs and OSSs as shown in Figure 6. Each MDS is responsible for metadata operations on a part of all files. The metadata set of files is determined by a hash value of the file name with path. The client determines the MDS of a file accessed in the client by a hash value of that file and path, and metadata operations for that file are sent to the MDS. The MDS informs the client to the location of OSSs for write/read operations in the client.

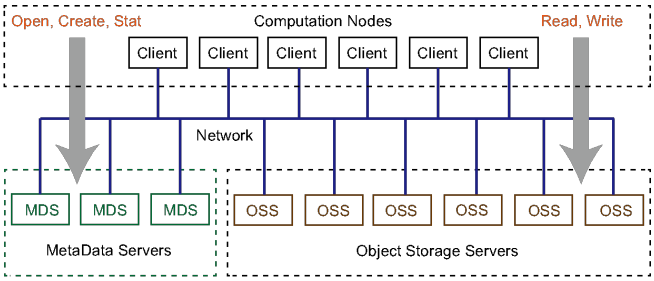

Figure 6 Hash-based Parallel File System

In FY2012, we implemented transactional metadata operations using software transactional memory for consistency of metadata. Two or more same operations are consistently executed at the same time. Evaluations show that file creation throughput of proposed file system is about 25 times higher than that of current Lustre File System in the case of 64 metadata servers as shown in Figure 7.

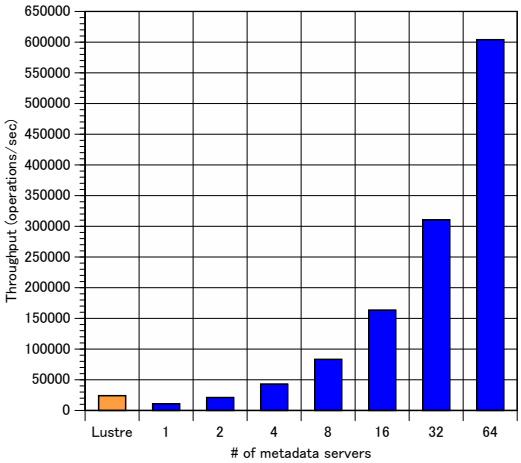

Figure 7 Throughput of the file creation

1.3.4. Persistent RDMA

The implementation of persistent communication provided in MPI has been reconsidered to provide low latency and true overlapping communication and computation. If the communication code is implemented using the persistent communication facility in the way that the end-points of both the sender and the receiver are set up by issuing MPI_Send_init and MPI_Send_recv primitives prior to actual communication triggered by the MPI_Start or MPI_Startall primitive. The same communication pattern is reused without reissuing the initialization. Thus, at the start of actual communications in persistent communication, the runtime system already knows all the communication patterns, i.e., peers and message sizes if both sender and receiver have issued persistent communication primitives. Such situations have

a chance for the communication library to utilize four DMA engines equipped in K computer and carry out true overlapping communication and computation.

FY 2012, a novel method to schedule DMA engines for K computer was designed and integrated into PRDMA (Persistent Remote Direct Memory Access) designed and implemented in FY 2011.

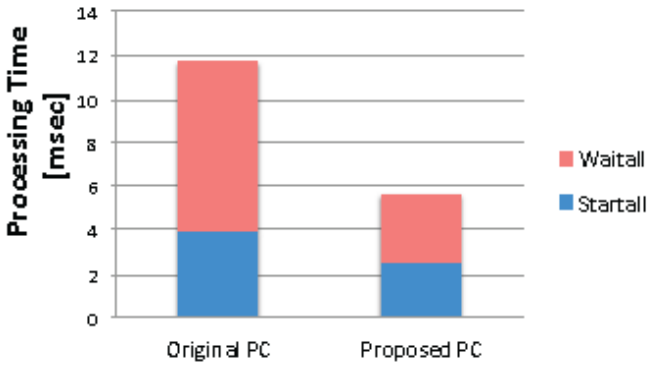

Figure 8 Comparison of original persistent communication and proposed one.

The proposed method was evaluated using a simple halo exchange program whose communication pattern is the same as a large eddy simulation code, SCALE-LES3. Figure 8 shows the evaluation results of using original and proposed persistent communication implementations. The proposed method is about two times faster than the original implementation.

1.3.5. New Process / Thread Model

We have been designing a new process / thread model that is suitable for the many-core architectures. The many-core architectures are gathering attention towards next generation supercomputers. Many-core architectures have a large number of low performance cores and the amount of the main memory per core is relatively small. On such environment, the system software should not consume a lot of memory.

The conventional process model has a problem in terms of the intra-node communication between parallel processes on many-core environments. It consumes a lot of memory in the kernel space for the intra-node communication using shared memory mapping scheme. In this scheme, shared memory regions are allocated on the memory, then each parallel process maps those shared memory regions to its own virtual address space. Parallel processes can communicate via those shared memory regions. However, a lot of page table entries are required to map the shared memory regions. For example, 200MB memory is consumed on the page table entries in the kernel space in case that parallel processes map the 1GB shared memory region to their own address space.

One of the solutions to solve this problem is to utilize a small shared memory region as just a intermediate buffer. In this method, parallel processes send and receive the data via the shared memory region allocated as an intermediate buffer. However, this method introduces two memory copies on the intra-node communication. These two memory copies can decrease the performance of the parallel application.

Partitioned Virtual Address Space (PVAS) is a new process model to achieve the low-cost intra-node communication on the many-core environments. In PVAS, multiple processes run in the same virtual address space as described in Figure 9 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 another process can be accessed by the normal load and store instructions, just like the same way accessing the data owned by itself. PVAS is designed not only to achieve high performance intra-node communication, but also to minimize the memory usage in the kernel space. PVAS processes do not have to map the shared memory regions for the intra-node communication because the address space boundaries between PVAS processes do not exist. Therefore, the page table entries to map the shared memory regions are not required.

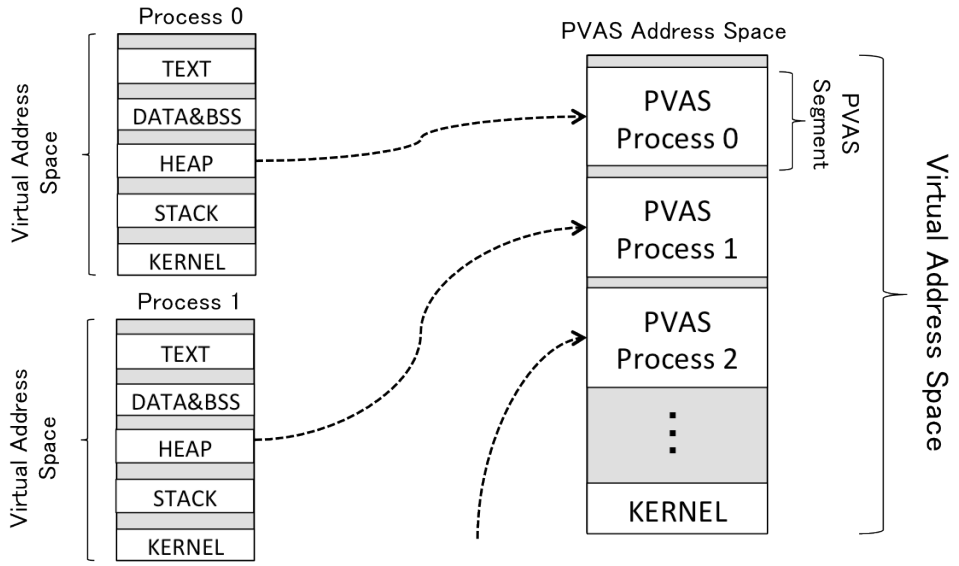

Figure 9 Semantics view of the Partitioned Virtual Address Space

We implemented the PVAS process model in the Linux kernel and modified the MPI communication library (MPICH2) to utilize the PVAS intra-node communication. Figure 10 shows the performance results of the ping-pong communication between a pair of processes utilizing MPI_Send() and MPI_Recv(). The latency of the PVAS implementation is faster than that of the Nemesis implementation that utilizes shared memory region as an intermediate buffer for the intra-node communication. Because the PVAS implementation introduces only one memory copy in the intra-node communication while the Nemesis implementation introduce two memory copies.

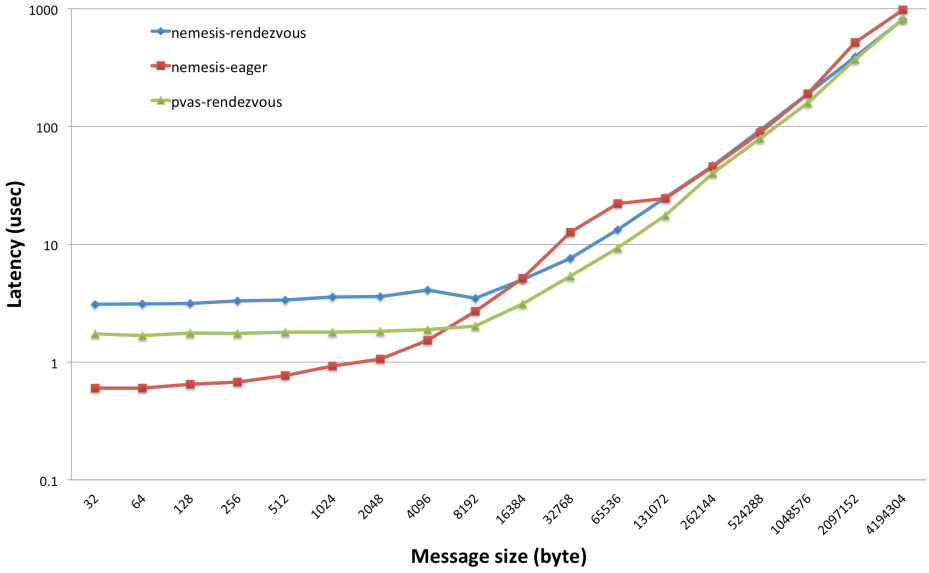

Figure 10 MPI ping-pong communication latency

We also evaluated the PVAS intra-node communication in PGAS language. We modified XacalableMP, which is one of the PGAS languages, to utilize the PVAS intra-node communication. XacalableMP implemented with the GASNet communication system for the intra-node communication. GASNet has two methods for the intra-node communication. First, GASNet-AM utilizes a shared memory region as just an inter-mediate buffer, and then two memory copies are required on the intra-node communication. In the other one, the GASNet-Shmem allocates a shared memory region as a global array and parallel processes map it to their own address space, therefore, only one memory copy is required on the intra-node communication.

Figure 11 shows the performance result of the NAS Parallel Benchmarks implemented by the XscalebleMP. A red bar represents the elapsed time for the intra-node communication and a green bar represents the elapsed time for the calculation.

The PVAS implementation is comparable with GASNet-Shmem, because it introduce only one memory copy as same as GASNet-Shmem. However, memory consumption of PVAS for the intra-node communication is fewer than that of GASNet-Shmem in theory.

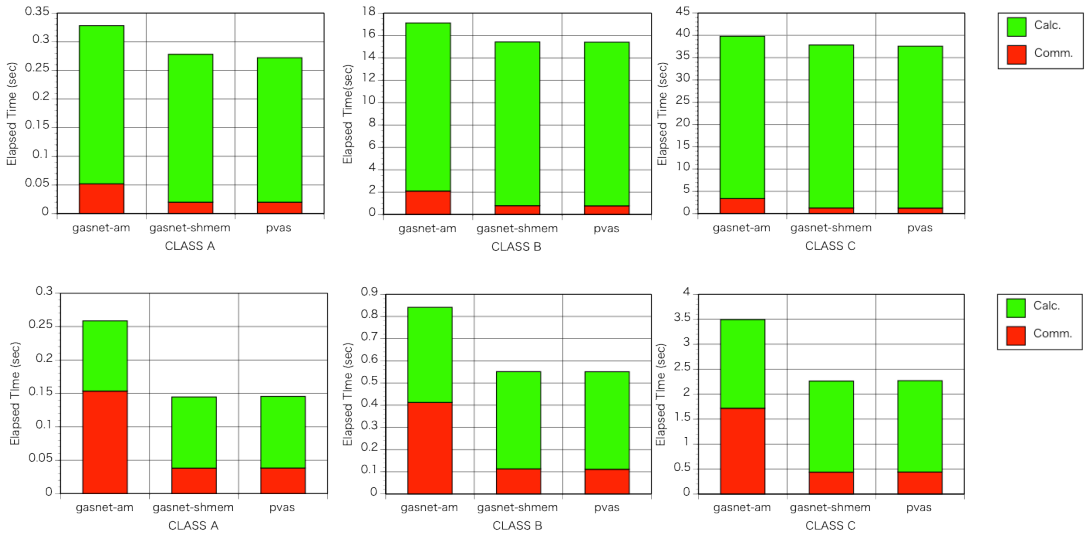

Figure 11 NAS Parallel Benchmarks

1.4. Schedule and Future Plan

PRDMA will be open to users in May of 2013. More optimization techniques will be developed to mitigate network contentions. FClib will be optimized for K computer, and will be open to users. File IO for SACLA XFEL will be redesigned based on the FY2012 research results. MPICH3, an MPI implementation for the MPI-3 standard, will be ported and enhanced in K computer.

The system software stack has been designed and implemented in cooperation with the University of Tokyo. In FY2013, PVAS and its related research topics will be much focused, and the operating system kernel will be mainly developed at the University of Tokyo.

1.5. Publication, Presentation and Deliverables

(1) Journal Papers

- None

(2) Conference Papers

1. PGAS Intra-node Communication towards Many-Core Architecture (Akio Shimada, Balazs Gerofi, Atsushi Hori, Yutaka Ishikawa), In 6th Conference on Partitioned Global Address Space Programing Model, 2012.
2. Yoshiyuki Ohno, Atsushi Hori, Yutaka Ishikawa, "File Composition Technique to Improve the Performance of Accessing a Number of Small Files", In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, volume I, pages 395--400, 2012

(3) Invited Talks

- None

(4) Posters and presentations

1. Partitioned Virtual Address Space, In The 1st HPC in Asia Workshop, 2012.
2. Yoshiyuki Ohno, Atsushi Hori, Yutaka Ishikawa, "File Composition Technique for Improving Access Performance of a Number of Small Files", 10th International Meeting on High-Performance Computing for Computational Science (VECPAR 2012), 2012.
3. Keiji Yamamoto, Atsushi Hori, Yutaka Ishikawa, “Distributed Metadata Management for Exascale Parallel File System”, SC12 The International Conference for High Performance Computing, Networking, Storage, and Analysis, 2012

(5) Patents and Deliverables

PRMDMA will be distributed in May 2013.