Programming Environment Research Team

1. Team members

Mitsuhisa Sato (Team Leader)

Hitoshi Murai (Research Scientist)

Miwako Tsuji (Research Scientist)

Masahiro Nakao (Postdoctoral Researcher)

Tomotake Nakamura (Research Associate)

Takenori Shimosaka (Research Associate)

Laurence Beaude (Research Trainee)

Swann Perarnau (Visiting Researcher, JSPS Research Fellow)

Masahiro Yasugi (Visiting Researcher)

Hitoshi Sakagami (Visiting Researcher)

Hiroaki Umeda (Visiting Researcher)

Horitz Helias (Visiting Researcher)

Susanne Kunkel (Visiting Researcher)

Tomoko Nakashima (Assistant (Concurrent))

2. Research Activities

The K computer system is a massively parallel system which has a huge number of processors connected by the high-speed network. In order to exploit full potential computing power to carry out advanced computational science, efficient parallel programming is required to coordinate these processors to perform scientific computing. We conducts researches and developments on parallel programming models and language to exploit full potentials of large-scale parallelism in the K computer and increase productivity of parallel programming.

In 2013FY, in order to archive these objectives above, we carried out the following researches:

1) We continued the development of XcalableMP(XMP) programming languages. XcalableMP is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations. The specification has been designed by XcalableMP Specification Working Group (XMP Spec WG) which consists of members from academia and research labs to industries in Japan. We have been working with XMP Spec WG to improve the specification. In this year, we studied the optimization of reflect communication, and one-sided communication, and applications of climate applications using the XcalableMP. As an extension to exascale computing, we are designing a new programming model and developing its compiler for emerging accelerator clusters, by combining XcalableMP and

OpenACC. W have released the version 0.7.0 in November 2013, and deployed it to the K computer. We submitted the results to SC13 HPCC class 2 competition, and awarded "HPCC Class2 Award".

2) We have working for Japan-France project FP3C, "Framework and Programming for Post Petascale Computing", from 2012, and have been developed the integrated programming environment, called FP2C, of XcalableMP and YML which is developed by the French team. In this year, we carried out performance evaluation of FP2C programming using the K computer.
3) For large-scale parallel applications, we investigate a new parallel communication library to support the communication between a set of multiple processes in Multiple processes Multiple Data (MIMD)
4) For the research for performance tuning tools for large-scale scientific applications running on the K computer, we are supporting the Scalasca performance turning and analysis tool developed by JSC, Germany, on the K computer. We have deployed it on the K computer as our AICS-supported software.
5) Aiming to explore runtime technologies of post-petascale computing, we studied the performance of work-stealing on the K computer for dynamic distributed load balancing. In this year, we designed a new load balancing efficiency metric to evaluate the performance of work-stealing.
6) We conducted several collaborations on the performance evaluation with JSC, University of Tsukuba and other groups.

In addition to the research activities, we conduct promotion activities to disseminate our software. To promote XcalableMP as a means for parallelization of programs, we made the XcalableMP workshop, seminars or lectures as follows.

- XcalableMP workshop (Nov. 1)
- FOCUS seminar (July 16, Sep. 18, Dec. 18)
- Earth Simulator user meeting (Oct. 17)

The seminar or lecture consists of both classroom and hands-on learning

3. Research Results and Achievements

3.1. Development of XcalableMP compiler

We are developing Omni XcalableMP that is an open-source XcalableMP compiler, in cooperation with Tsukuba University, and released the version 0.7.0 in November 2013.

3.1.1 Optimization of Reflect communication in stencil computation

The key changes in this version are the improvements of stencil communications in performance and conformance to the language specification, which is related with XMP's shadow and reflect directives [1]. First, three methods of data transfer in stencil communication were implemented:

- Based on the MPI's derived datatype: Array elements on distribution boundary are transferred in a batch as a derived datatype. The performance depends on the implementation of MPI, but it is an advantage that it can be applied in any environment.
- Based on parallel packing/unpacking: Data is packed to a buffer with multithreading before communication and unpacked from a buffer. This can be effective in multicore environments.
- Based on RDMA of the K compute: Using the extended interface of RDMA available on the K computer, stencil communications can be implemented more efficiently than using MPI. Each contiguous block of array elements is put to the corresponding location in the stencil area on the neighbor node.

Users can select with an environment variables which method is to be used in their XcalableMP program.

Secondly, functions of the reflect directives are fully implemented for XMP/Fortran. The function includes:

- the width clause
- the periodic modifier
- the async clause

3.1.2 Evaluation of productivity and performance of XcalableMP: HPCC Class2

In order to evaluate productivity and performance of XcalableMP, we have implemented HPC Challenge (HPCC) benchmarks. The HPCC benchmarks are a set of benchmarks to evaluate multiple attributes of an HPC system. The HPCC benchmarks consist of RandomAccess, Fast Fourier Transform (FFT), High Performance Linpack (HPL), and STREAM. We have implemented them by using XcalableMP. In addition, we have implemented the Himeno Benchmark, which is a typical stencil application. Table 1 shows the source lines of code (SLOC) of our implementations. The SLOCs of XcalableMP are smaller than those of the reference implementations by using MPI.

Table 1: Source lines of code of HPCC and HIMENO benchmarks.

	HPL	RandomAccess	FFT†	STREAM	HIMENO
XcalableMP	306	250	70	66	137
Reference	8,800	938	101	329	380

†Only kernel implementation

All benchmarks were compiled by using the Omni XMP compiler 0.7.0-alpha. In order to evaluate the performances of these benchmarks, we used all compute nodes at a maximum on the K computer. For comparison, we also evaluated the some reference implementations. For HPL, we compared our performance with the theoretical performance. Fig. 1 shows the performance results. The performances of XcalableMP implementations are almost the same as those of the reference implementations.

Through these implementations and performance evaluations, we have revealed that XcalableMP has good productivity and performance. We have submitted the results to the SC13 HPC Challenge Benchmark Class2 Competition, and we have awarded the HPC Challenge Class 2 Award.

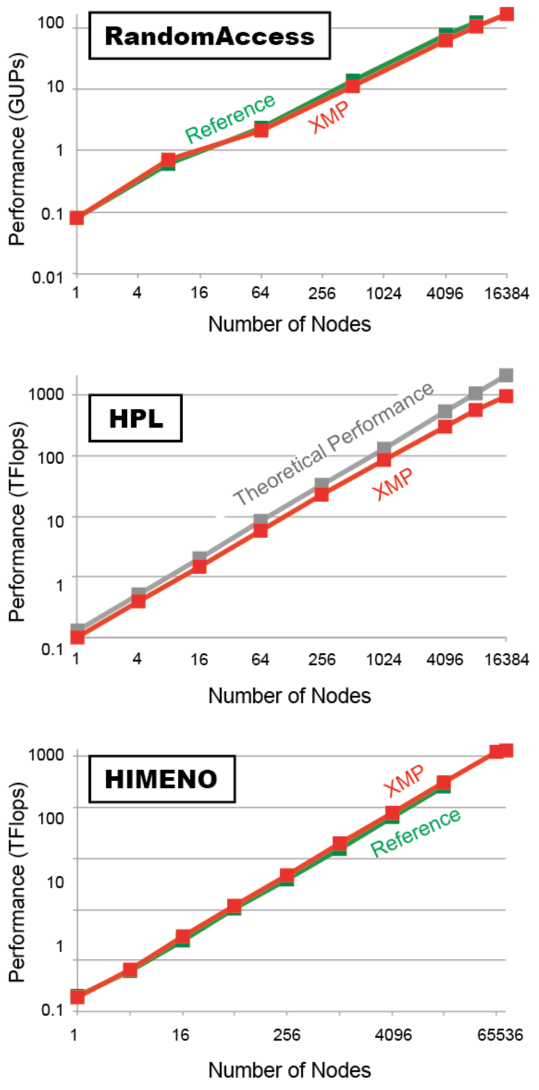


Fig 1. Performance results of HPCC benchmarks and HIMENO benchmark.

3.1.3 Evaluation of climate applications on the K computer

Exascale computing will enable us to predict a high-precision climate change by using very high-resolution model. In this study, we have ported following existing climate applications to the K computer and evaluated their performance.

- CGPOP Miniapp (CGPOP) : Mini application of global ocean modeling.
- NICAM : Global cloud resolving model.
- CICE : Sea ice model.

In order to improve productivity, we have used coarray syntax provided by XcalableMP for the porting. This study is a part of "G8 ESC - Enabling Climate Simulations at Extreme Scale," project with partners of US, France, Germany, Spain, and Canada.

The CGPOP needs a reduction operation by all compute nodes. We have implemented to use hardware support of the reduction operation provided by the K computer. Next, the all applications need a sleeve exchange operation. We have implemented the exchange operation by using RDMA API provided by the K computer. This operation is supposed to use the coarray syntax. Finally, we have added OpenMP directives in CGPOP to perform thread-parallelization. Fig. 2 shows the performance results. We achieve 84% speed up in CGPOP, and 7% speed up in NICAM, and 15% speed up in CICE. In addition, by using coarray syntax, the codes of them are simpler than those of original ones.

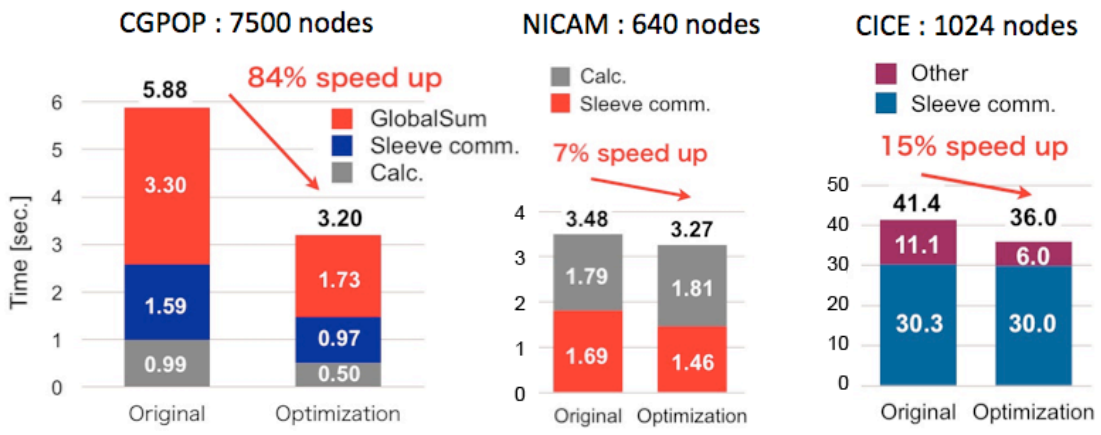
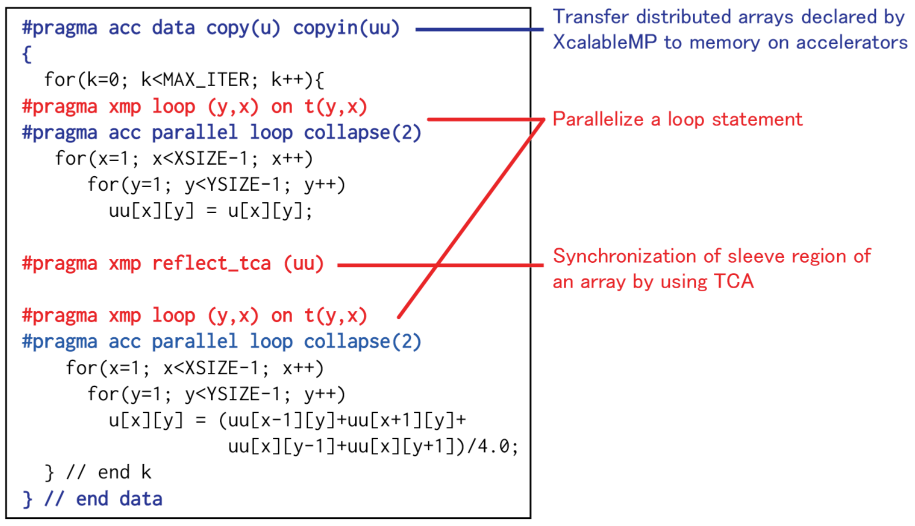


Fig 2. Performance results of Climate applications.

3.1.2 Extensions for Tightly Coupled Accelerators

We are designing a new programming model and developing its compiler for emerging accelerator clusters. Specifically, out target is accelerator clusters that are capable of direct communication between two accelerator devices on different nodes, called Tightly Coupled Accelerators (TCA). TCA is a next-generation device for communication between accelerators. In order to improve productivity of applications using TCA and accelerators, we have developed a new programming model as an extension of XcalableMP.

Our basic idea is combining XcalableMP and OpenACC. OpenACC is a directive-based API for offloading programs written in C, C++ and Fortran programs from a host CPU to an attached accelerator device. In our approach, XcalableMP directives are used to specify global-view-based parallel processing among nodes and OpenACC ones to control accelerator devices. The function of direct communication between accelerators is also provided as an extension to XcalableMP. This new programming model can significantly improve productivity in writing programs for accelerator clusters. Fig. 3 shows an example of the proposed programming model. Programmer only adds XcalableMP directives and OpenACC directives, and programmer can develop applications using TCA and accelerators.



```
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
   for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        uu[x][y] = u[x][y];

#pragma xmp reflect_tca (uu)

#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
} // end data
```

Transfer distributed arrays declared by XcalableMP to memory on accelerators
Parallelize a loop statement
Synchronization of sleeve region of an array by using TCA

Fig 3. Example of the proposed programming model for TCA.

We have evaluated its performance on HA-PACS/TCA cluster, which is the demonstration system with TCA. As a result, the performance of the proposed programming model is almost the same as that of the application that directly uses native APIs of TCA.

This research is supported by CREST, JST.

3.2 FP3C Project & FP2C Software

FP3C (Framework and Programming for Post Petascale Computing) project is a French-Japanese collaborative research project involving INRIA, CNRS, CEA, and Maison de la Simulation (in France) and University of Tsukuba, University of Tokyo, Institute of Technology of Tokyo, the University of Kyoto and RIKEN AICS (in Japan). The project started on September 1st, 2010, and continued until March 31th, 2014. The aim of the project was to establish software technologies, to propose languages and programming models to explore extreme performance computing beyond petascale computing.

It is expected that post-petascale systems will be a huge, heterogeneous and highly hierarchical architecture with nodes of general processing cores and accelerator cores. For the programming models currently considered, for example the hybrid programming model of MPI+OpenMP, it would be sometimes difficult to exploit such systems efficiently. It would be essential to use multiple programming methodologies across multiple architectural levels. Based on this new programming model, we have developed the FP2C (Framework for Post-Petascale Computing) software to develop and execute applications on large, hierarchical and heterogeneous systems.

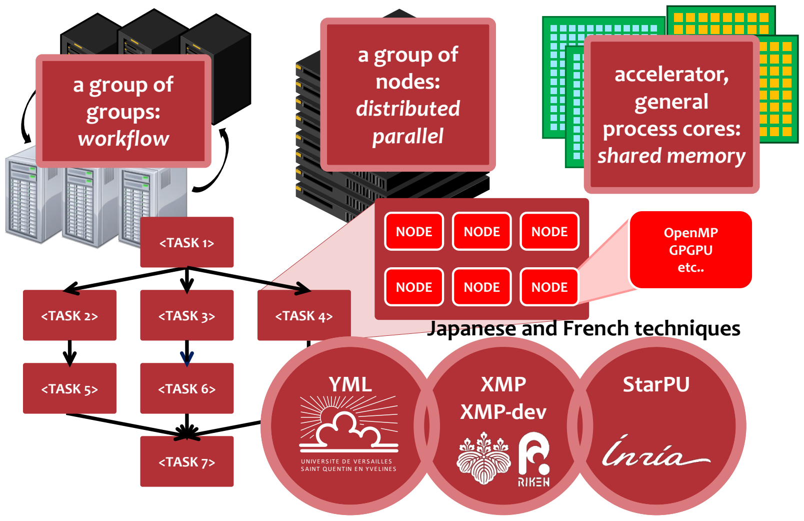

a group of groups: workflow
a group of nodes: distributed parallel
accelerator, general process cores: shared memory
<TASK 1>
<TASK 2>
<TASK 3>
<TASK 4>
<TASK 5>
<TASK 6>
<TASK 7>
NODE
OpenMP GPGPU etc..
Japanese and French techniques
YML
XMP XMP-dev
StarPU

Fig 4. Overview of FP2C software.

Fig. 4 shows the overview of FP2C, which consists of workflow (supported by YML), distributed parallel programming model (supported by XMP) and shared memory/GPPGU programming model (supported by XMP-dev/StarPU). Within a group of tightly connected nodes, we adopt distributed programming model, and within a node composed of processing/accelerator cores, we adopt shared-memory/GPGPU programming model. These hybrid distributed parallel programs are considered to be tasks of a workflow. Therefore, between the groups of node, there is a workflow programing model, to manage and control these tasks. In order to develop FP2C software to realize the hierarchical programming model, we combine many Japanese and French techniques described above during this collaborative project.

One advantage of FP2C over the traditional workflow is that we can extend tasks by introducing distribute parallel programming model into its tasks to speed-up the tasks and overall application. Another advantage of FP2C over the traditional distributed parallel programming model is that we can divide a large parallel program into several moderate sub-programs to avoid the communication cost in the large program. Additionally, we can construct a complicated program easily by combining existing parallel programs and parallel libraries (such as Scalapack) based on the workflow paradigm.

We have performed experiments on the K-computer with Block-Gauss-Jordan problem in order to investigate the different levels of hierarchical parallelisms. Therefore, in our experiments, while the matrix size is fixed, the number of blocks is varied. Also, while the total number of processors for a workflow application is fixed, the number of processers for each task is varied. As shown in the Fig. 5, the best result has been obtained from the mixture of different programming paradigms which realized based on our framework.

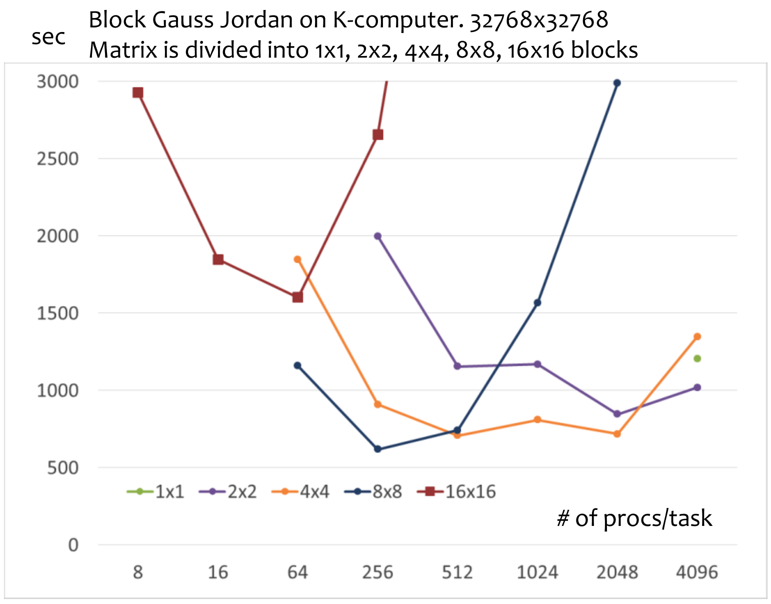


Fig 5. Performance of Block Gauss Jordan Method using FP2C on the K computer.

We have also implemented multiple implicitly restarted Arnoldi methods (MIRAM) with FP2C. The MIRAM is an eigen-solvers investigated by another group of the FP3C project. The MIRAM, which invokes two or more interacting restarted Arnoldi solvers, was particularly designed to be suited for environments that combine different parallel programming paradigms, i.e. coarse and fine grain parallelisms. Inside each solver, fine grain parallelism such as distributed parallel model is realized by XMP or MPI. The solvers communicate each other based on coarse grain parallelism realized by YML. The MIRAM for dense and sparse matrices are implemented with YML, XMP, MPI, and external libraries such as Petsc, Slepc, and PARPACK.

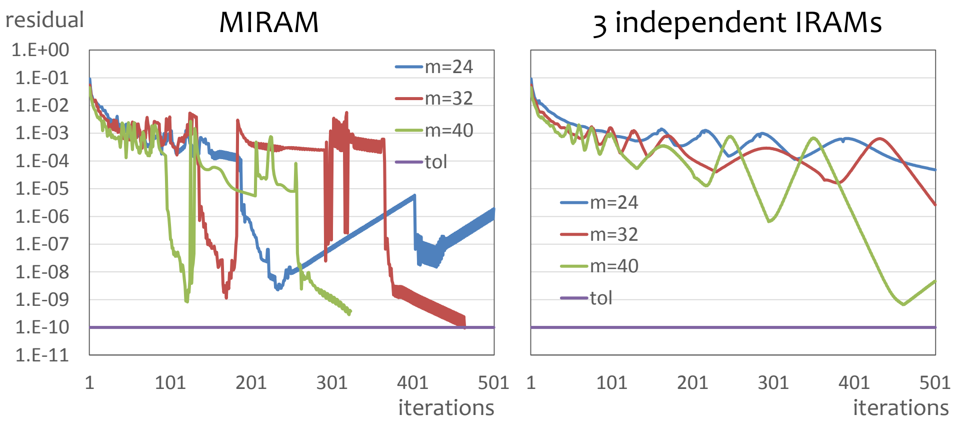


Fig 6. Performance of MIRAM using FP2C on the K computer.

Fig. 6 shows the experimental result of MIRAM for the sparse matrix called "Schenk/nlpkkt240" (Rows x Colmns = 27,993,600 x 27,993,600, 760,648,352 non-zero elements) provided by UF Sparse Matrix Collection. According to the evaluation on the K computer, 2 different kinds of speedup based on 2 different parallel models have been observed:

- Speedup convergence by the interactions of multiple solvers based on the cause grain parallelism
- Speedup each iteration based on the fine grain parallelism.

3.3 Communication Library between Multiple Sets of MPI Processes for MPMD model

An MPMD programming model is widely used as a master-worker program or a coupling program for multiple physical models. Recent high-end parallel computers have more than several thousand nodes. In order to utilize the parallel computers for an MPMD model, we proposed the communication library MPMPI between different multiple sets of MPI processes for the MPMD model.

We designed the basic MPMPI function MPMPI_Send and MPMPI_Recv. In order to effectively develop the interface specification of MPMPI_Send/Recv, we referred to the concept of some basic MPI functions. One of the features of the MPMPI functions is the exchange of distributed data between sets of processes having different shapes or different distribution methods. We also designed the MPMPI interfaces for the XcalableMP PGAS language. MPMPI_Send and MPMPI_Recv have been currently implemented in C language and the MPI_Send/Recv functions. They currently support one and two dimensional block distribution methods.

Fig. 7 shows the performances of the master-worker program with the Block Gauss Jordan algorithm using our library. We programs are implemented in XcalableMP/C (XMP/C). An experimental environment is the K computer system. Fig.7 shows the basic performance of MPMPI_Send/Recv. We assume that a whole matrix is stored in multiple main memories. For matrix products on each process of worker programs in the BGJ program, we utilized the thread parallelized LAPACK and the BLAS library provided by the K computer system. The sizes (n × n) of the input matrix A in the BGJ program are 16384 × 16384, 32768 × 32768 and 65536 × 65536. They are distributed onto 32 × 32 nodes in the master program. The numbers of nodes that one MPMPI communication uses are 2×2, 4×4, 8×8, 16×16 nodes.

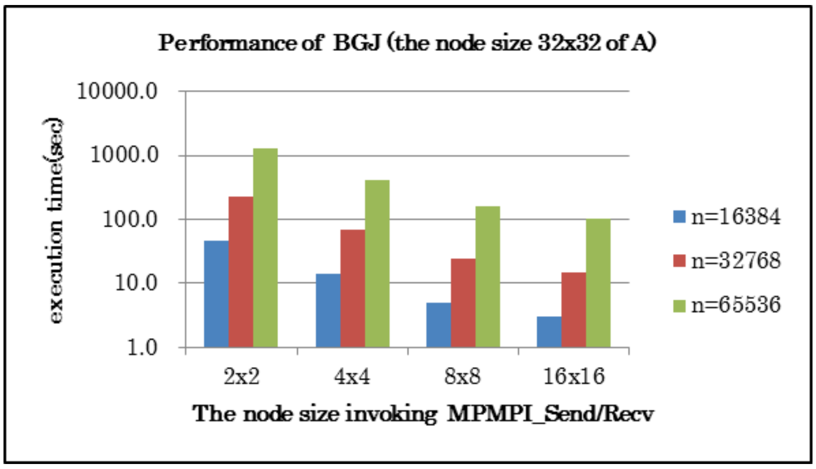


Fig 7. Performance of the master-worker program with Block Gauss Jordan algorithm using MPMPI

We found that the elapsed times of the master-worker program decrease as the numbers of nodes that one MPMPI communication uses increases. The percentages of the MPMPI communications are less than 10% for all cases.

3.4 Performance Study and Optimization of Distributed Load Balancing on the K computer

Distributing computations, in particular irregular ones, at such scale requires increasingly complex and dynamic load balancing systems. Work stealing is a provably efficient scheduling algorithm for such distributed, dynamic load balancing requirements. It is becoming increasingly popular, both for shared memory systems (intra-node load balancing) and in distributed settings (inter-node load balancing). We studied the performance of work-stealing on the K Computer in regards to a previously overlooked issue in High Performance Computing settings: the impact of large scale latencies. Using the publicly available, MPI-based implementation of the Unbalanced Tree Search (UTS) benchmark, we evaluated the performance of work stealing at the scale of several thousands compute nodes.

This implementation of UTS follows the general structure of any work stealing application. When work is available, a process retrieves a node from its stack. This node's data is then used to compute its children, which are pushed into the stack. If no work is available, a victim process is selected, and work is fetched from its stack. This process continues until all work is exhausted. Such condition is detected by a token-ring distributed termination algorithm. Fig. 8 presents the speedup of this same benchmark between 1024 and 8192 MPI Processes, for 3 process allocations (1 rank per node, 8 ranks per node in round robin, 8 consecutive ranks per node). This figure demonstrates that this UTS implementation does not scale past 2048 nodes. Additionally, it appears that the benchmark performance is severely impacted by the way in which processes are distributed among compute nodes. In particular, allocating successive ranks to different compute nodes results in the worse performance observed.

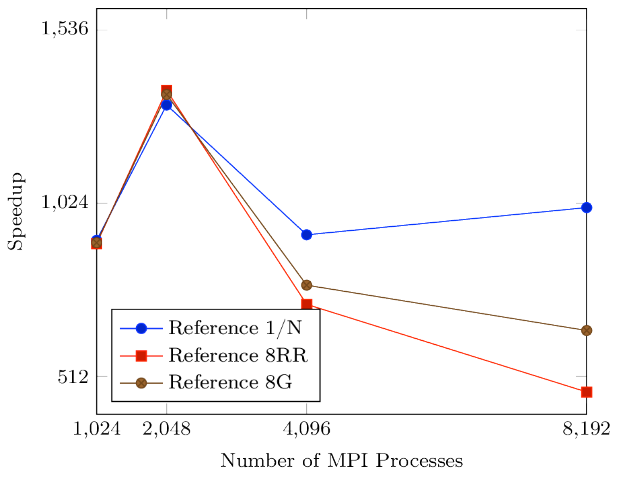


Fig 8. Speedup of the reference MPI work stealing implementation in the K computer.

To explain those results, we designed a new load balancing efficiency metric. The role of a dynamic load balancing scheme is to maximize the amount of processes having work at any given time. Thus, if a problem is comprised of enough work items, the state of an application should roughly be separated in three phases: the starting phase where work is distributed to all processes, the finishing phase during which work becomes scarce and the number of active processes decreases and the middle phase for which most processes are processing work. If work generation is irregular, as with UTS, balancing is also needed during the middle phase, but an efficient framework should be able to maintain a reasonable amount of processes busy. This intuition drives the definition of our performance metric. If one was to trace the active and idle phases of each process participating in the computation, it should be possible post-mortem to determine the number of active processes at any time during execution of the application.

We define here active phases as periods of time during which a process's stack contains work. Thus, in the chosen UTS implementation, all the time where a process is generating new nodes or handling MPI operations in between, (responding to steal requests for example) count as active. Similarly, a process is inactive if it does not have work locally. It should be noted that with such definition, most types of load balancing operations can be counted in either type of phase. Assuming there exists a trace of all processes indicating the time of each transition from one type of phase to the other, with the starting time of the application as t = 0, we now define the following metrics.

First, let workers(t) be the number of processes in an active phase at time t. From this number we derive the maximum number of workers at any given time during execution Wmax and an occupancy ratio O(t) = workers(t)/N, N being the number of processes executing the application. Second, let the starting latency be SL(x) = min(t,O(t)=x) /T, with T the total execution time of the application. This measure computes, for a given occupancy ratio, the first time it was exceeded during execution and represent it as a ratio of the total execution. Similarly, we define the ending latency as EL(x) = (T –max(t,O(t)=x))/T . Intuitively, the starting latency gives us the speed, relative to the total execution time of the application, at which a number of processes becomes active. The ending latency reflects a similar idea: how far away in the execution the framework is able to maintain a number of processes active. Fig. 9 shows the results of starting and ending latencies for an execution of the reference implementation with 8192 MPI processes, 1 process per node in the K computer. Data is limited to latencies lower than 10%.

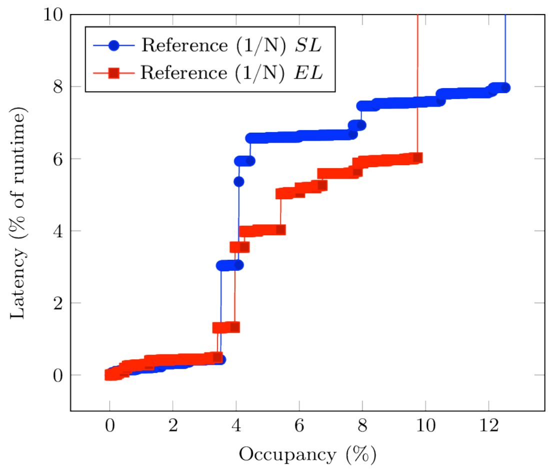


Fig 9. Starting and Ending latencies in the K computer.

On the K computer, latencies between nodes in the same blade are lower than inside the cube or across racks. Furthermore, the number of compute nodes inside a rack (only 96) means that an allocation of 8192 nodes can easily span across more than 80 racks, and in practice we observed that a communication between two processes can go through more than 10 hops.

To reflect these variations in response times for a steal, we designed a random selection strategy using a biased distribution. The idea is the following: while preserving the ability to steal any process, weight the probability of one process stealing another by the distance between those two. The farther a process is, the lower the probability of being chosen. As the Fujitsu MPI implementation provide extensions to query the 6D coordinates in the Tofu network of any MPI rank, we used the Euclidean distance between nodes to weight the probability. Fig. 10 shows the results improved by this approach, as indicated as "Tofu".

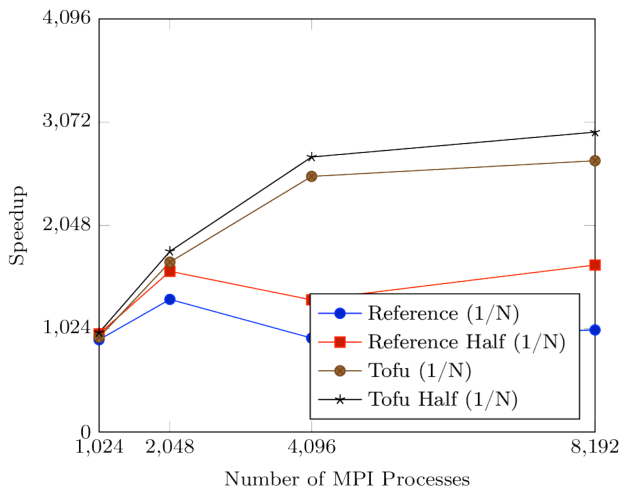


Fig 10. Speedup of the MPI work stealing implementations by using topology of Tofu network.

4. Schedule and Future Plan

One of important archivement of this year was that our XcalableMP submission was awarded HPCC Class2 Awards to be given for high productivity and performance. To extend this acheivement, the important action is to disseminate our XcalableMP to applications users. Actually, in this year, we organized several schools and hands-on, workshop with potential users. We will continue these promotion activities for the next year while we will study more optimization technique of XcalableMP compiler to improve the performance. As a research agenda especially for the K computer, we will contribute the scalability of large-scale applications for the K computer.

For the performance turning tools, we have been supporting Scalasca for the K computer. The next step will be to integrate with XcalableMP to make the performance tuning process easily. We have already done this partially for C, and will extend it to real large applications written in Fortran for more integrated programming environment for larege-scale parallel computing. Through these case studies, we will extend it for valuable performance analysis in the K computer.

The programming models for post-petascale will be investigated, including programming models and runtime techniques to support dynamic load balancing in large-scale parallel programs, and parallel programing supports for new communication devices, TCA, in advanced GPU clusters.

5. Publication, Presentation and Deliverables

(1) Conference Papers

[1] Hitoshi Murai, Mitsuhisa Sato. "An Efficient Implementation of Stencil Communication for the XcalableMP PGAS Parallel Programming Language", 7th International Conference on PGAS Programming Models, Edinburgh UK, Oct. 2013.

[2] Masahiro Nakao, Hitoshi Murai, Takenori Shimosaka, and Mitsuhisa Sato. "Productivity and Performance of the HPC Challenge Benchmarks with the XcalableMP PGAS Language", 7th International Conference on PGAS Programming Models, Edinburgh UK, Oct. 2013.

[3] Serge Petiton, Mitsuhisa Sato, Nahid Emad, Christophe Calvin, Miwako Tsuji and Makarem Dandouna, Multi-level programming Paradigm for Extreme Computing, Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo,2013.10.27-31, Paris France.

[4] Miwako Tsuji, Mitsuhisa Sato, Maxime Hugues and Serge Petiton, Multiple-SPMD Programming Environment based on PGAS and Workflow toward Post-Petascale Computing, Proceedings of the 2013 International Conference on Parallel Processing (ICPP-2013), 480--485,2013.10.01-04, Ecole Normale Superieure de Lyon Lyon France.

[5] Miwako Tsuji, Makarem Dandouna and Nahid Emad,Multi level parallelism of Multiple implicitly/explicitly restarted Arnoldi methods for post-petascale computing,Proceedings of the 8th IEEE International Conference on P2P Parallel Grid Cloud and Internet Computing (3PGCIC-2013),158--165,2013.10.28-30,University of Technology of Compiegne Compiegne France.

[not refereed, in Japanese]

[6] Masahiro Nakao, Hitoshi Murai, Takenori Shimosaka, Akihiro Tabuchi, Yuetsu Kodama, Taisuke Boku, Mitsuhisa Sato, "Extension of XcalableMP for Tightly Coupled Accelerators", IPSJ SIG Technical Report, Mar. 2013 (in Japanese).

[7] Masahiro Nakao, Mitsuhisa Sato, "Implementation of NICAM by using PGAS model language on the K computer", Vol.2013-HPC-140, pp.1-5 Aug. 2013 (in Japanese).

[8] Masahiro Nakao, Mitsuhisa Sato, "Performance measurement of CGPOP Mini-application on the K computer", IPSJ SIG Technical Report, May 2013 (in Japanese).

[9] T. Shimosaka, M. Sato, T. Boku, W. Tang. Evaluation of the nuclear fusion simulation code GTC-P on the K computer. IPSJ SIG Technical Report, May. 2013 (in Japanese).

(2) Invited Talks

[10] Mitsuhisa Sato, Issues for Exascale Accelerated Computing - system architecture and programming, 7th Int'l. Conf. on PGAS Programming Models.

(3) Posters and Presentations

[11] Masahiro Nakao, Hitoshi Murai Takenori Shimosaka Mitsuhisa Sato. “XcalableMP for Productivity and Performance in HPC Challenge Award Competition Class 2”, SC13 The 2013 HPC Challenge Awards BoF, Denver, Colorado, USA, Nov., 2013.

[12] Masahiro Nakao, “Performance and productivity of XcalableMP”, 5th cross-disciplinary symposium of computational science – development, assimilation, and construction of new knowledge”, Nov. 2013 (in Japanese).

[13] Nahid Emad, Leroy Drummond, Miwako Tsuji and Makarem Dandouna, Tuning Asynchronous Co-Methods for Large-scale Eigenvalue Calculations 16th SIAM Conference on Parallel Processing for Scientific Computing, 2014.02.18--21, Portland Marriott Downtown Waterfront Portland OR USA, 2014.

[14] T. Shimosaka, H. Murai and M. Sato, A communication library between multiple sets of MPI processes for a MPMD model, pp. 147–148(poster), EuroMPI2013, 2013.

[15] Swann Perarnau, Mitsuhisa Sato, Weighted Distribution for Random Victim Selection in Distributed Work Stealing, HPC in Asia Workshop, International Supercomputing Conference (ISC), Leipzig Germany, June 2013.

(4) Patents and Deliverables

[16] Omni XcalableMP compiler ver. 0.7for the K computer (registered as AICS-supported software)

[17] Scalasca performance analysis tool for the K computer (registered as AICS-supported software)