Chapter 2

Programming Environment Research Team

2.1 Members

Mitsuhisa Sato (Team Leader)

Hitoshi Murai (Research Scientist)

Miwako Tsuji (Research Scientist)

Masahiro Nakao (Research Scientist)

Jinpil Lee (Postdoc Researcher)

Yuetsu Kodama (Senior Research Scientist)

Hidetoshi Iwashita (Research Associate)

Shinichi Ito (Research Associate)

Makoto Ishihara (Agency Staff)

Masahiro Yasugi (Senior Visiting Researcher)

Hitoshi Sakagami (Senior Visiting Researcher)

Brian Wylie (Visiting Researcher)

Christian Feld (Visiting Researcher)

Kengo Nakajima (Senior Visiting Researcher)

Tomoko Nakashima (Assistant)

2.2 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 2015FY, in order to archive these objectives above, we carried out the following researches:

(1) We are working on the development and improvement of XcalableMP (XMP) programming languages. XcalableMP is a directive-based language extension, designed by XcalableMP Specification Working Group (XMP Spec WG) including some members from our team as a community effort in Japan. It allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations. In this year, we have improved Coarray functions in Fortran. The feature of coarray of the Omni XcalableMP compiler is implemented for the K compiler. In addition, some benchmarks and an application are parallelized with XcalableMP and their performance is evaluated on the K computer.

(2) As an extension of XcalableMP to exascale computing, we are proposing a new programming model, XcalableACC, for emerging accelerator clusters, by integrating XcalableMP and OpenACC. We continue working on the language design and the compiler development of XcalableACC. This research is funded by JST CREST project on "post-petascale computing".

(3) Co-design for HPC is a bidirectional approach, where a system would be designed on demand from applications, and applications must be optimized to the system. We started the design of tools for co-design, including the SCAMP profiler for the network of large scale systems.

(4) As the post-K computer will be a large-scale multicore-based system, we are investigating programming models for manycore-based parallel systems as a next version of XcalableMP, including dynamic tasking and load balancing as well as advanced PGAS models for distributed memory systems.

(5) We conducted several collaborations on the performance evaluation with JSC, University of Tsukuba, Kyusyu Institute of Technology and other groups. In the collaborations with Kyusyu Institute of Technology, a task parallel language Tascell was evaluated on the K computer. We are developing tools for performance analysis of large-scale parallel programs, by enhancing a tuning tool Scalasca, which is being developed by JSC, for the K computer. This tool is used for performance analysis of real applications, in collaboration with their developers.

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 and LENS workshop (Oct. 29, 30)
- Tutorial of XMP at Osaka University (Oct 23)
- Tutorial of XMP at University of Tsukuba (Dec 9)
- FOCUS seminar on XMP (Jan 8)

The seminar or tutorials consist of both classroom and hands-on learning

2.3 Research Results and Achievements

We are developing Omni XcalableMP that is an open-source XcalableMP compiler, in cooperation with the university of Tsukuba. The latest version 0.9.2 has been released in November, 2015

2.3.1 Improvement of the coarray feature of XcalableMP

Coarray Fortran (CAF) is a parallel language that is a language extension of Fortran. To support the local view, XMP contains coarray features, which were adopted from Coarray Fortran (CAF) defined as part of Fortran 2008 standard. Based on experience of the implementation of Omni XMP C compiler, we have implemented and improved main part of CAF specification into XMP Fortran compiler.

We performed two imporvements for memory allocation / registration and Communication.

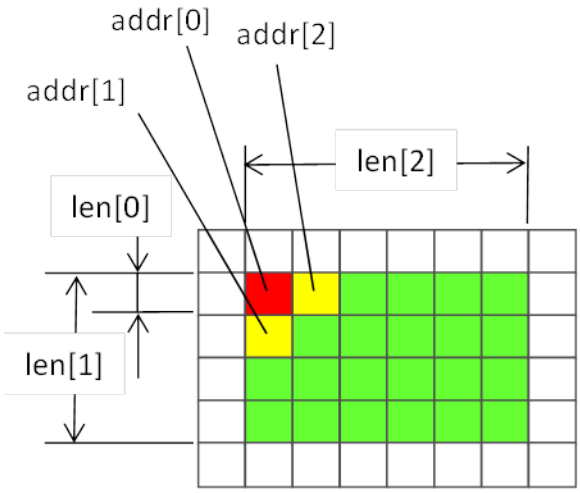

Figure 2.1: Parameters that determines data contiguity

Improvement of Memory Allocation / Registration

Coarrays are variables that can be accessed from other nodes. To allow remote nodes to access the local data, the address of the data must be registered at runtime with the low-level communication library, e.g., Tofu library in case of the K computer. To reduce runtime overhead of this operation for all coarrays, we made a mechanism that registers all static coarrays just before the execution of the program. The compiler generates the initializer for all static coarrays appearing in the program file at compile time and generates the caller calling the initializer at linkage time.

Improvement of Communication

To reduce communication latency overhead, contiguous data should be transferred simultaneously. For partially contiguous multidimensional array data, the length of the contiguous portion and the periodic pattern should be detected at compile time or at runtime. We made an algorithm and implemented on the compiler and the runtime library. Figure 2.1 shows an example of partially-contiguous communication data (colored elements) and major parameters in the algorithm. The parameters, lengths and addresses of data elements, are analyzed by the compiler and forwarded to the runtime library to find the contiguity.

Experimental Results

(1) Himeno benchmark

We ported Himeno benchmark program written with MPI to four different CAF programs. They used the same Fujitsu Fortran compiler with the same options including automatic thread parallelization. While the MPI version has 610 lines excluding comment and empty lines, the CAF versions have 402 to 415 lines, 32% to 34% shorter.

The result on grid size XL (1024 512 512) is shown in Figure 2.2. Two CAF programs are respectively 5% and 2% faster than the MPI version in average.

(2) NAS Parallel benchmark

We ported NAS Parallel benchmarks CG, EP, FT and MG written in MPI for CAF respectively. Figure 2.3 shows the result of CG Class-C as an instance and summarizes the history of the CAF program tuning. Finally CAF program V49 exceeds the original MPI version in performance. On EP, the CAF version is only 2% less performance in average than the original without tuning. On FT, the first version of CAF program extract more than 93% of the performance of the original in all evaluation ranges of Class B, C and D. Besides the CAF program has still room for performance tuning. On MG, the final version of CAF provides almost the same performance as the original in average.

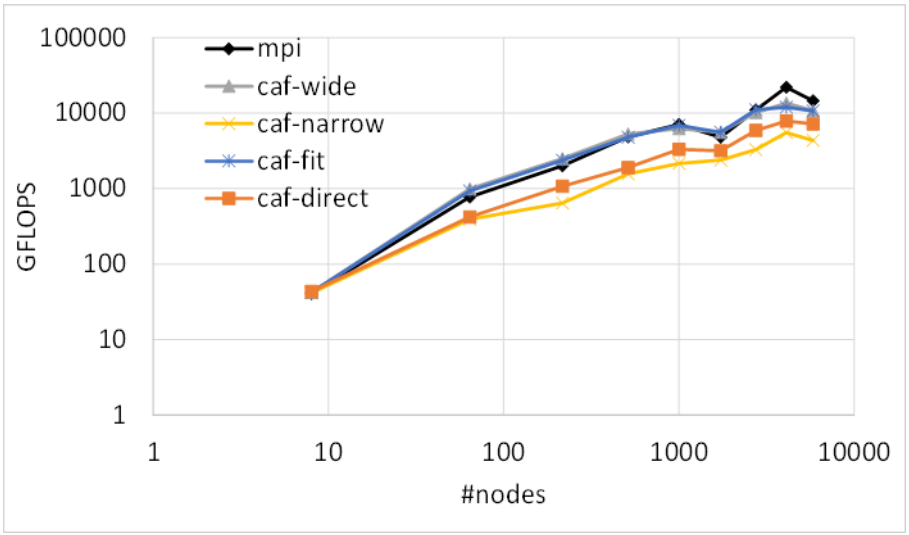

Figure 2.2: Four different CAF programs vs. the original MPI on Himeno benchmark

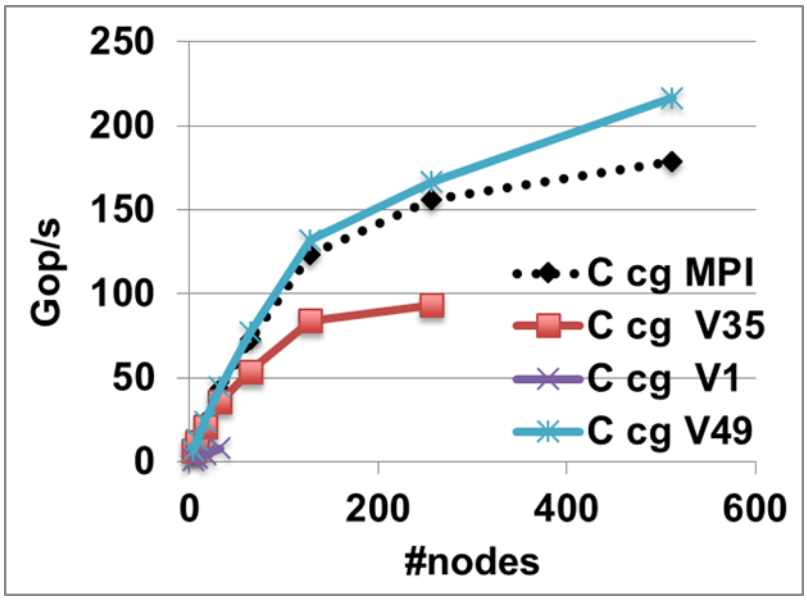

Figure 2.3: Porting and performance tuning of CAF program on NPB CG Class-C

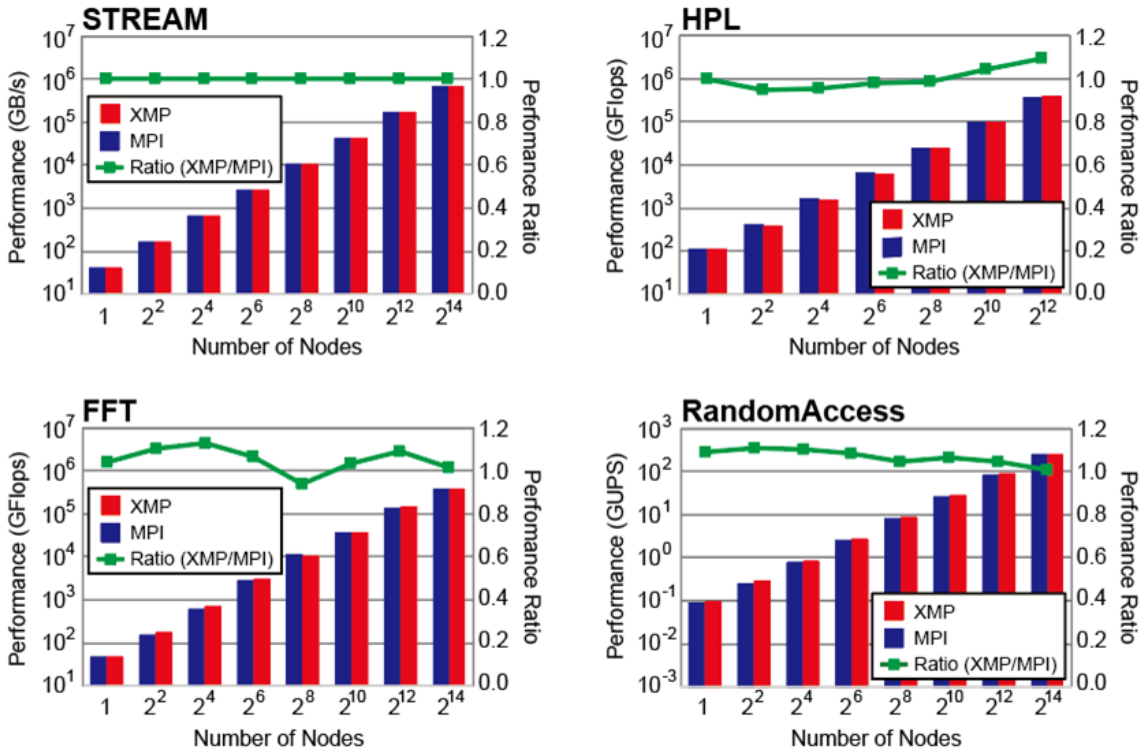

Figure 2.4: Performance of HPC Challenge Benchmark with XcalableMP

	STREAM	HPL	FFT	RandomAccess
XcalableMP	62	517	201	226
MPI	329	8,800	787	938

Table 2.1: SLOC of HPC Challenge Benchmark with XcalableMP

2.3.2 Performance Evaluation of the HPC Challenge Benchmark with XcalableMP

To evaluate productivity and performance of XcalableMP, we have implemented four benchmarks, namely STREAM, HPL, FFT, RandomAccess, in the HPC Challenge Benchmark Suite by using XcalableMP.

The figure 2.4 shows that the performance results of the XMP implementations. For a comparison purpose, we have also evaluated the performances of the MPI implementations which are reference implementations. The horizontal axis means that the number of compute nodes, the left vertical axis means that the performance corresponding to the bar, the right vertical axis means that the ratio of the performance of the XMP implementation to that of the MPI implementation corresponding to the line. When the performance ratio is greater than 1, the performance of the XMP implementation is better than that of the MPI implementation. The figure shows that the performances of XMP are almost the same as those of MPI.

The table 2.1 shows that source lines of code (SLOC) of the benchmarks in XMP and MPI. The table shows that the SLOCs of the XMP implementations are much less than those of the MPI implementations.

2.3.3 Performance of Three-dimensional Fluid Simulation with XcalableMP

The three-dimensional Eulerian fluid code written in Fortran, IMPACT-3D, which performs compressible and inviscid fluid computation to simulate converging asymmetric flows related to laser

#core	lx=ly=lz	only Z	both Y and Z		all of X, Y and Z		
		nz	ny	nz	nx	ny	nz
256	1024	32	8	4	4	4	2
2048	2048	256	16	16	8	8	4
16384	4096		64	32	16	16	8
131072	8192		128	128	32	32	16

Table 2.2: Simulation parameters

fusion, is parallelized by three different domain decomposition methods, namely the domain is divided in (1) only Z direction, (2) both Y and Z directions and (3) all of X, Y and Z directions using by using directives only for the "global-view" programming model of XcalableMP (XMP). The program is also hand-coded with MPI using the same domain decomposition methods, and the performance difference between XMP and MPI codes is evaluated on the K computer.

As one node consists of 8 cores in the K computer, one process is dispatched onto each node and each process performs parallel computations with 8 threads, which are explicitly described by OpenMP in both XMP and MPI programs. We run both XMP and MPI codes with three different decomposition methods and evaluate the weak scaling on the K computer using Omni XcalableMP/Fortran compiler 0.7.0 and Fujitsu Fortran K-1.2.0.15. A number of cores for execution and corresponding simulation parameters are summarized in Table 2.2. lx, ly, lz are Fortran array size of first, second, third dimension, and nx, ny, nz are a number of division in X, Y, Z direction, respectively.

Performance is measured by a hardware monitor installed on the K computer, and three indexes are obtained. The total number of floating point operations is counted by the hardware monitor and is interpreted to MFLOPS using elapsed time. Finally it is divided by theoretical peak MFLOPS and output as MFLOPS/PEAK value. The average amount of transfer data per second between memory and CPU is also monitored. It is divided by theoretical peak memory access throughput and output as Memory throughput/PEAK value. The hardware monitor counts the number of instructions, and the number of SIMD instructions is divided by the total number of instructions to obtain SIMD execution usage.

MFLOPS/PEAK values for all 6 cases, namely (MPI, XMP) x (only Z, both Y and Z, all of X, Y and Z) are shown in Fig. 2.5 (a). Performance of XMP codes is as same as that of MPI codes, and small differences among three decomposition methods are found. But we can get only 8 9 % of peak performance of the K computer. From the hardware monitor, we found that SIMD execution usage was less than 5% in all cases, and this could degrade the performance. Most cost intensive DO loops in IMPACT-3D include IF statements, which are needed to correctly treat extremely low velocity and flow direction change regardless of XMP and MPI codes, and the IF statement prevents the native Fortran compiler from generating SIMD instructions inside the DO loop. Thus relatively low performance is obtained.

As the true rate of the IF statement is nearly 100% in IMPACT-3D, speculative execution of SIMD instruction causes almost no overhead. So forcing the compiler to generate the SIMD instructions could be useful to enhance the performance, and it can be done with "simd=2" compiler option. All codes are recompiled with that option and rerun. SIMD execution usage increases up to around 50% in all cases, and we can expect performance improvement. MFLOPS/PEAK values for all cases are shown in Fig. 2.5 (b). MPI code performance is improved and we can get up to 20% of the peak performance. XMP code performance is also improved, but these are below 15% even XMP code performance is almost same as MPI code performance without "simd=2" option. Although Memory throughput/PEAK values of MPI codes are 55%, those of XMP codes are only 37% and this low memory throughput is one of candidates for low sustained performance.

In the converted code by the XMP/F compiler, all Fortran arrays are treated as allocatable arrays even the original code uses static arrays. The allocatable array prevents the native Fortran compiler from optimizing the DO loop with prefetch instructions because the array size cannot be determined at compilation time, and it could cause low memory throughput. All Fortran static arrays in the hand-coded MPI code for the decomposition method of all of X, Y and Z directions

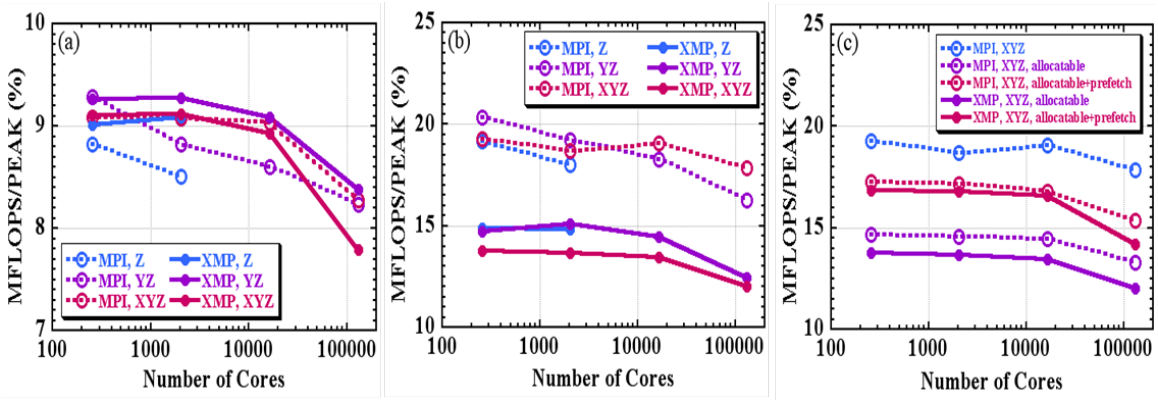

Figure 2.5: Performance comparison between MPI and XMP on the K computer with (a) no optimization for three decomposition methods, (b) SIMD optimization for three decomposition methods and (c) allocatable array optimization for the decomposition method of all of X, Y and Z directions.

are just replaced by allocatable arrays and we check a performance difference. Performance of the MPI code is shown in Fig. 2.5 (c) for static arrays (blue dash) and allocatable arrays (purple dash). MFLOPS/PEAK values are dropped from 20% to 15%, and this performance degradation without the prefetch instructions is confirmed. To force the native Fortran compiler to perform the prefetch optimization, we can use additional "prefetch_stride" compiler option. All codes are recompiled with "simd=2" and "prefetch_stride" options and rerun. Performance improvements by this compiler option are shown in Fig. 2.5 (c) for both MPI (purple dash to red dash) and XMP (purple solid to red solid) codes. MFLOPS/PEAK values are improved by 2 3% with the prefetch optimization.

2.3.4 Design of SCAMP (SCAlable Mpi Profiler) as a co-design tool for large-scale network

Co-design for HPC is a bidirectional approach, where a system would be designed on demand from applications, and applications must be optimized to the system. In order to co-design the network of large scale systems, it is important to evaluate the communication performance of applications. The trace driven simulator estimates the network performance based on trace files. Firstly, user should run their application on a real system in parallel to obtain the trace files from all processes. These trace files should contain MPI function calls, and their arguments and time stamps, etc. Then, the performance of a virtual system is estimated by using the trace files. While the trace driven simulator is straightforward, sometimes it is not appropriate for the simulation of large parallel systems since it is difficult to obtain the number of trace files for the future system if the current system is smaller than the future one. In order to tackle this scaling-problem in the trace driven simulator, we propose a method called SCAMP (SCAlable Mpi Profiler), which creates a large number of pseudo trace files based on the small number of trace files obtained from a small system and drives the network simulator using the pseudo trace files to estimate the performance of the large systems.

According to the experiments using SCAMP and using K-computer, as shown in Figure 2.6, SCAMP overestimates the performance of benchmarks, i.e. the runtime estimated by SCAMP is shorter than the real runtime on K-computer. The reason is that while we have focused only on the network performance, the computation time would change as the number of nodes increases.

2.3.5 Performance evaluation of Tascell on the K computer

Tascell is a task parallel language that supports distributed memory environments. A Tascell worker spawns a real task only when requested by another idle worker. The worker spawns a task after restoring its oldest task-spawnable state by temporarily backtracking. This mechanism eliminates the cost of spawning/managing logical threads. It also promotes the reuse of workspaces and improves the locality of reference since it does not need to prepare a workspace for each concurrently runnable logical thread. Furthermore, a single Tascell program can run efficiently on shared and distributed memory environments.

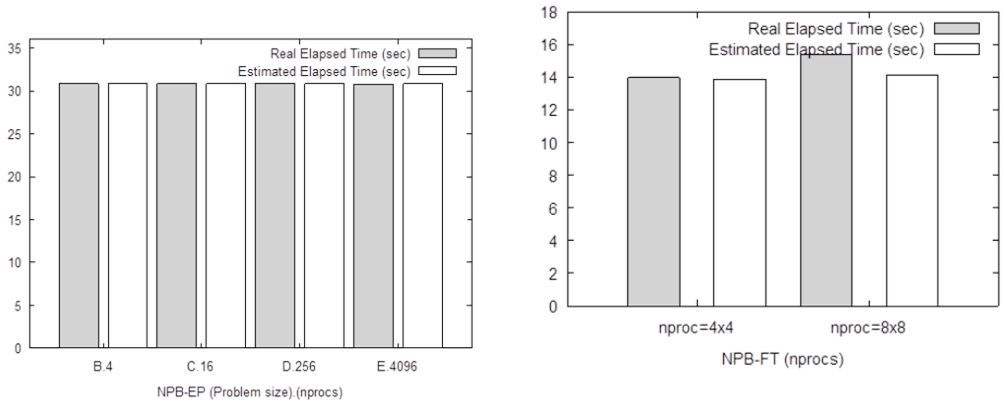

Figure 2.6: Comparison between real time and estimated time by SCAMP

This study aims to evaluate Tascell on massively parallel systems; in particular, we employed 1024 nodes of the K Computer with 8192 cores in total. In addition, we revised the implementation of Tascell to get it working on such systems.

In the conventional implementation of Tascell, inter-node communication is realized by TCP/IP communication via message routing servers called Tascell servers. This implementation is suitable for dynamic addition of computation nodes and wide-area distributed environments. On the other hand, Tascell servers often become communication bottlenecks. Furthermore, in recent supercomputer environments, there may be no appropriate places for deploying Tascell servers, and TCP/IP may not be available for inter-node communication; it is hard or impossible to run the conventional implementation in such an environments.

Therefore, we implemented inter-node communication in Tascell using MPI, which is supported by most practical supercomputer systems. At the same time, we adopted a server-less implementation in order to overcome the deployment and bottleneck problems, excluding the support of wide-area distributed environments. Note that programmers can write Tascell programs without concern about the underlying communication layer.

We evaluate the performance of our MPI-based implementation on the K computer using 7168 workers (7 workers x 1024 nodes). The result is shown in Figure 2.7.

In order to enable our implementation to work with the MPI implementation on the K computer and many other MPI implementations, it only requires the MPI_THREAD_FUNNELED support level, in which only the main thread can make MPI calls, and the two-sided communication paradigm. With such minimum requirements, our MPI-based implementation successfully realized both high performance and deadlock freedom.

2.4 Schedule and Future Plan

From this year, we started the study of the programming models for post-petascale, including programming models and runtime techniques to support manycore. We already propose XcableACC as a solution for accelerator-based system, which is to be explored in the JST CREST project. As the post-K computer will be a large-scale multicore-based system, we will investigate programming models for manycore-based parallel systems including dynamic tasking and load balancing as well as advanced PGAS models for distributed memory systems.

As in recent years, an important action for XcalableMP project is to disseminate our XcalableMP to applications users. As in last years, we organized several schools and hands-on, workshop with potential users also in this year. We will continue these promotion activities 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.

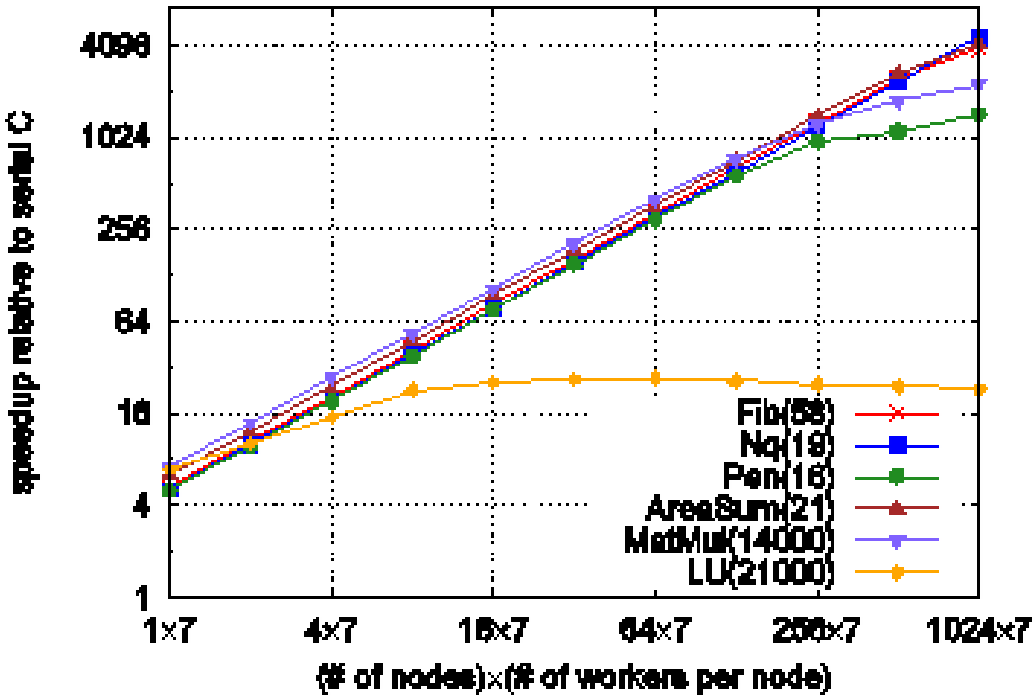

Figure 2.7: Evaluation results (Speedup) of Tascell programs on the K computer

2.5 Publications

Journal Articles

[1] A. Tabuchi et al. “Evaluation of A PGAS Language XcalableACC for Accelerator Cluster”. In: *IPSJ Trans. on Advanced Computing Systems, Vol. 9, No.1* (2016). (in Japanese), pp. 17–29.

[2] M. Ikei and M. Sato. “Design of a PGAS Runtime System for Efficient Parallel Stencil Computations on Multi-node Many-core Processors”. In: *IEICE Transaction D. vol. 99, No.2* (2016). (in Japanese), pp. 138–151.

[3] T. Odajima et al. “Hybrid Communication with TCA and InfiniBand on a Parallel Programming Language for Accelerators XcalableACC”. In: *IPSJ Trans. on Advanced Computing Systems, Vol. 4, No.8* (2015). (in Japanese), pp. 61–77.

Conference Papers

[4] H. Sakagami and H. Murai. “Performance of Three-dimensional Fluid Simulation with XcalableMP on the K computer”. In: *Proc. of the 30th International Supercomputing Conference*. 2015.

[5] Hidetoshi Iwashita, Masahiro Nakao and Mitsuhisa Sato. “Preliminary Implementation of Coarray Fortran Translator Based on Omni XcalableMP”. In: *The 9th International Conference on Partitioned Global Address Space Programming Models (PGAS2015)*. 2015.

[6] Miwako Tsuji, Serge G. Petiton and Mitsuhisa Sato. “Fault Tolerance Features of a New Multi-SPMD Programming/Execution Environment”. In: *Proceedings of First International Workshop on Extreme Scale Programming Models and Middleware (SC 2015)*. 2015.

[7] Tetsuya Odajima, Taisuke Boku, Toshihiro Hanawa, Hitoshi Murai, Masahiro Nakao, Akihiro Tabuchi and Mitsuhisa Sato. “Hybrid Communication with TCA and InfiniBand on A Parallel Programming Language XcalableACC for GPU Clusters”. In: *Workshop Series on Heterogeneous and Unconventional Cluster Architectures and Applications (HUCAA)*. 2015.

Invited Talks

[8] Mitsuhisa Sato. *Challenges for Parallel Programming Models and Languages of post-petascale and exascale computing*. IWOMP 2015. 2015.

[9] Mitsuhisa Sato. *FLAGSHIP 2020 project –Development of Japanese National Flagship supercomputer "post K" –*. US-Japan Joint Institute for Fusion Theory Workshop on Innovations and co-designs of fusion simulations towards extreme scale computing. 2015.

[10] Mitsuhisa Sato. *FLAGSHIP 2020 project –Development of Japanese National Flagship supercomputer "post K" –*. The satellite symposium of ICQC 2015. 2015.

Posters and Presentations

[11] Iwashita, Nakao, Sato. *Implementation and Evaluation of Coarray Fortran Compiler based on XcalableMP translator*. IPSJ SIGHPC Meeting 2015-HPC-150. (in Japanese). 2015.

[12] Nakao, Murai, Iwashita, Shimosaka, Boku, Sato. *Implementation and Evaluation of HPC Challenge Benchmarks using PGAS language XcalableMP*. IPSJ SIGHPC Meeting 2015-HPC-148. (in Japanese). 2015.

[13] Nakao, Murai, Iwashita, Tabuchi, Boku, Sato. *Implementation and Evaluation of HP Challenge Benchmarks using PGAS language XcalableACC*. IPSJ SIGHPC Meeting 2015-HPC-151. (in Japanese). 2015.

[14] Sugiyama, Lee, Murai, Sato. *Design of OpenMP using light-weight thread library Argobots*. IPSJ SIGHPC Meeting 2015-HPC-150. (in Japanese). 2015.

[15] Tabuchi, Nakao, Murai, Boku, Sato. *Performance Evaluation of XcalableACC for Cluster with Accelerator*. IPSJ SIGHPC Meeting 2015-HPC-150. (in Japanese). 2015.

[16] Tsugane, Nakao, Lee, Murai, Sato. *A proposal of Dynamic Task Construct in PGAS language XcalablMP*. IPSJ SIGHPC Meeting 2015-HPC-151. (in Japanese). 2015.

[17] Tsuji and Sato. *Application replay environment for Performance Evaluation of Single Node*. IPSJ SIGHPC Meeting 2015-HPC-150. (in Japanese). 2015.

[18] Tsuji, Lee, Boku, Sato. *SCAMP: A proposal of Network performance estimation method using MPI pseudo profile*. IPSJ SIGHPC Meeting 2016-HPC-15. (in Japanese). 2016.

Patents and Deliverables

[19] *Omni XcalableMP compiler ver. 0.9.3 (registered as an AICS-supported software).*