Chapter 2

Programming Environment Research Team

2.1 Members

Mitsuhisa Sato (Team Leader)

Yuetsu Kodama (Senior Research Scientist)

Hitoshi Murai (Research Scientist)

Miwako Tsuji (Research Scientist)

Masahiro Nakao (Research Scientist)

Jinpil Lee (Postdoc Researcher)

Tetsuya Odajima (Postdoc Researcher)

Manabu Yagi (Postdoc Researcher)

Itaru Kitayama (Technical Staff)

Masahiro Yasugi (Senior Visiting Researcher)

Hitoshi Sakagami (Senior Visiting Researcher)

Brian Wylie (Visiting Researcher)

Christian Feld (Visiting Researcher)

Hidetoshi Iwashita (Visiting Researcher)

Hiroko Takahashi (Assistant)

2.2 Overview of Research Activities

In order to exploit full potential computing power of large-scale parallel system such as the K computer to carry out advanced computational science, efficient parallel programming is required to coordinate these processors to perform scientific computing. Our team conduct researches and developments on parallel programming models and language to exploit full potentials of large-scale parallelism in the large-scale parallel system and increase productivity of parallel programming.

In 2019FY, in order to archive these objectives above, we carried out the following researches:

(1) We continued 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. In this year, we have evaluated XcalableMP benchmark programing on Fugaku. The performance of XcalableMP on the Fugaku is enhanced by the manycore processor and a new Tofu-D interconnect. As an application for XcalableMP, we investigated the graph algorithm for the the Order/Degree problem. For an extension of XcalableMP to exascale computing, we are working on XcalableACC for emerging accelerator clusters, by integrating XcalableMP and OpenACC.

(2) Since the Fugaku is a large-scale multicore-based system, we are investigating programming models for manycore-based parallel systems as XcalableMP 2.0. We focus especially on the integration of dynamic tasking with PGAS programming model. In this year, we continued the design of programming models for task parallelism and PGAS.

(3) We investigated the programming model for accelerators including Field Programmable Gate Array (FPGA), which is expected as one of the promising technologies for future acceleration technology. In this year, we investigate OpenMP technologies for GPU and FPGA, and the task parallel programming model to combine several kinds of accelerators such as GPU and FPGA.

(4) As a part of Flagship 2020 project, we are developing several tools for co-design, including performance analysis tools and simulators. In this year, we have improved the accuracy of the Gem5 simulator for Fujitsu A54FX used in the Fugaku system.

(5) We conducted several collaborations on the performance evaluation with JSC, University of Tsukuba, Kyusyu Institute of Technology and other groups. In this year, we carried out the performance study on NEST brain simulator developed by JSC, using Scalasca. Prof. Yasugi's group of Kyusyu Institute of Technology developed a new approach to fault-tolerant language systems without a single point of failure for irregular parallel applications.

In addition to the research activities, we conducted promotion activities to disseminate our software. To promote XcalableMP as a means for parallelization of programs, we organized the XcalableMP workshop as follows:

- The 6th XcalableMP workshop (Nov. 1st, 2019, at University of Tsukuba)

2.3 Research Results and Achievements

2.3.1 XcalableMP on Fugaku

In this section, we report our early experience and the preliminary performance of XcalableMP on Fugaku. The Fugaku is a huge-scale system with general-purpose manycore processors. The node processor is a single chip, named Fujitsu A64FX, which consists of 48 cores with 2 or 4 cores dedicated for OS activities, 32 GiB HBM2 memory, with TofuD interconnect, and a PCI express controller in the chip together. The Fugaku system consists of 158,976 nodes in 432 racks. The Fugaku is scheduled to be put into operation for public service around 2021. In 2020, the installation is completed, and the system partially serves the early-access program.

XcalableMP is available as a parallel programming language for the Fugaku, supported by R-CCS team with Fujitsu. C and Fortran are supported as base languages with XcalableMP 1.2 compliant.

We report the preliminary performance of XcalableMP program running on the Fugaku [1].

We used the following versions:

- Omni XcalableMP Version: 1.3.2, Git Hash:6d23f46
- Language specification: 1.2.25

The performance of XcalableMP on the Fugaku is enhanced by the manycore processor and a new Tofu-D interconnect.

[1] The reported results were obtained on the evaluation environment in the trial phase. Note that the performance is not guaranteed at the start of its operation.

2.3.1.1 Performance of XcalableMP Global View Programming

We executed the IMPACT-3D, described in Chapter 6, for the evaluation of XcalableMP global view programming in the Fugaku, using up to 512 nodes. The scalability on Fugaku is shown in Figure 2.1, comparing to the K computer. The program is parallelized by hybrid XMP-OpenMP parallel programming: An XMP node is assigned to a node, and 48 OpenMP threads are running within a node. The problem size is $512 \times 512 \times 512$ with 3-dimensional block distribution. The compile option is "-Kfast".

As shown in the figure, we found a good scalability in Fugaku, and the performance is better than that by MPI thanks to the optimized XMP runtime for communications in the stencil computation.

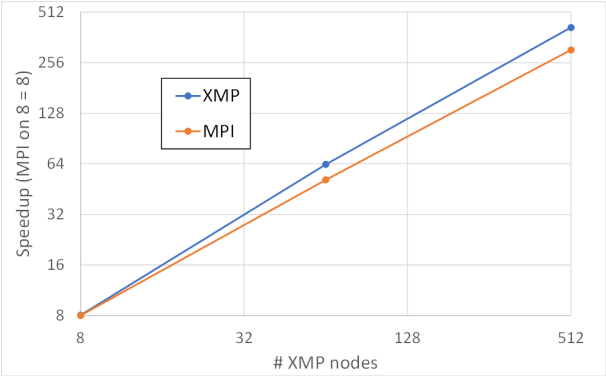

Figure 2.1: Speedup of Impact3D on Fugaku and Performance comparing to K computer

2.3.1.2 Performance of XcalableMP Local View Programming

Fugaku has a customized interconnection, called Tofu-D, which supports hardware-supported RDMA (Remote Direct Memory Access) operations. We implemented the XMP runtime library to make use of Tofu-D for one-sided communication for the XMP local view programming. The library is implemented by using a low-level communication layer, uTofu API, provided by Fujitsu.

For performance evaluation of XMP local view programming, we used CCS QCD and NTChem-MINI taken from the coarray version of Fiber Miniapp Suite.

To run CCS QCD mini-application, 8 XMP nodes are assigned to one node, running in a flat XMP mode. The size and conditions are as follows:

- Target data: Class 2 (32 x 32 x 32 x 32) (strong scaling)
- Compiler options: -Kfast,zfill,simd=2
- Timing region: sum of "Clover + Clover_inv Performance" and "BiCGStab(CPU:double precision) Performance" of the built-in timing feature

Figure 2.2 shows the speedup of the Fugaku, comparing to the performance of the K computer. The XMP version archives almost same performance of the MPI version. Note that the reason of the performance degradation of the XMP version on the K computer is the overhead of allocation for allocatable coarray used as a buffer for communication. It is improved by removing this overhead by using the uTofu communication layer.

The NTChem-MINI is a mini-application taken from NTChem, a high-performance software package for molecular electronic structure calculation. An XMP node is assigned to one node, and within a node, BLAS functions are executed using 48 cores. The size and conditions are set as follows:

- Target data: taxol (strong scaling)
- Compiler options: -Kfast,simd=2
- Timing region: RIMP2_Driver of the built-in timing feature

As shown in Figure 2.3, the XMP versions archive almost the same performance of the original MPI versions.

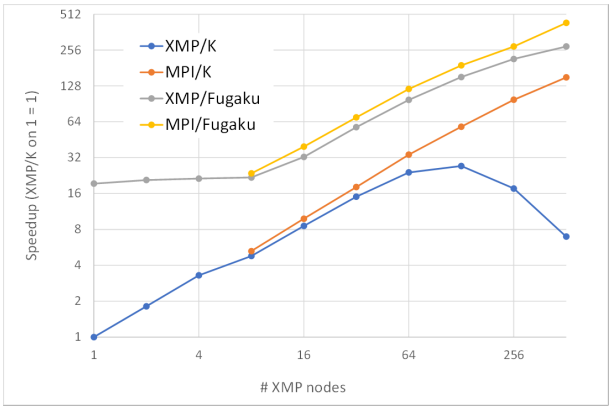

Figure 2.2: Speedup of CCS QCD on Fugaku and Performance comparing to the K computer

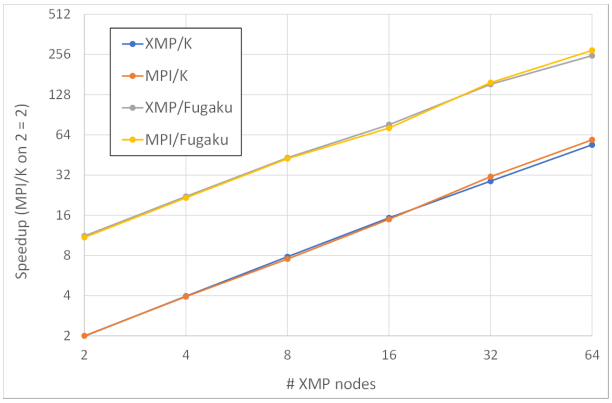

Figure 2.3: Speedup of NTChem-MINI on Fugaku and Performance comparing to the K computer

2.3.2 Accuracy Improvement of Memory System Simulation for Modern Processor

For the purpose of developing applications for supercomputer Fugaku at an early stage, RIKEN has developed a processor simulator. This simulator is based on the general-purpose processor simulator gem5. It does not simulate the actual hardware of a Fugaku processor. However, we believe that sufficient simulation accuracy can be obtained since it simulates the instruction pipeline of out-of-order execution with cycle-level accuracy along with performing detailed parameter tuning of out-of-order resources. In order to estimate the accurate execution time of a program, it is necessary to simulate with accuracy not only the instruction execution time, but also the access time of the cache memory hierarchy. Therefore, in the RIKEN simulator, we expanded gem5 to match the performance of the cache memory hierarchy to that of a Fugaku processor. In this simulator, we aim to estimate the execution cycles of one node application on a Fugaku processor with accuracy that enables relative evaluation and application tuning.

2.3.2.1 RIKEN simulator for A64FX

The RIKEN simulator currently supports only one CMG simulation, and multithreaded execution of up to 12 cores is possible. In order to estimate the accurate execution time of a program, it is necessary to simulate with accuracy not only the instruction execution time, but also the access time of the cache memory hierarchy. Therefore, in the RIKEN simulator, we expanded gem5 to match the performance of the cache memory hierarchy to that of the A64FX. The main extensions are as follows. These functions were developed for the RIKEN simulator, but many of them are general functions and can be applied to other processors.

- The L1 cache and L2 cache capacity, associativity, line size and latency were set in the gem5 parameter file according to the actual settings of the A64FX.

- In gem5, it is assumed that load and store operations can access the L1 cache in the same cycle. On the other hand, the A64FX enables two load or single store operations in a cycle. The RIKEN simulator enabled the same controls as those of the A64FX.

- In gem5, the L1 cache access in order to maintain L2 cache such as cache fill from L2 cache to L1 cache and write-back from L1 cache to L2 cache is controlled independently of the L1 cache access from the core. Therefore, the performance of the L1 cache will be enhanced. By performing exclusive control between them, the RIKEN simulator was able to simulate the L1 cache performance of the A64FX accurately.

- In gem5, when the core accesses to the L1 cache exceeds cache alignment, the access is divided into multiple accesses, and the overhead occurs. The A64FX is designed so as not to cause performance degradation even when accessing across cache lines. The RIKEN simulator also supports the unaligned cache access without overhead.

- In gem5 for ARM ISA, software prefetch was supported, but only prefetch for read access targeting L1 cache was implemented. Prefetch for write access is important for optimizing memory access, so the RIKEN simulator supports it. This feature has already been reported to the gem5 developers and accepted. Software prefetch targeting L2 cache is also important for optimizing memory access. The RIKEN simulator also supports it.

- Gem5 has several hardware prefetching capabilities, but there are only simple prefetches such as queued prefetch and stride prefetch. Furthermore, they only support prefetch for read access from next level memory hierarchy. The A64FX supports the hardware prefetch which extended the prefetch of K computer, but the RIKEN simulator implements the following prefetch that is based on the prefetch of K computer with small modification matched to the A64FX. When a cache miss is detected, a prefetch entry is generated. When a subsequent memory access matches the prefetch entry, two consecutive prefetch requests are generated both for L1 cache and L2 cache. When the prefetch distance reaches a set value, it is switched to single prefetch. This prefetch distance is set individually for L1 cache and L2 cache. The hardware prefetch supports both read access and write access.

- Gem5 supports shared L2 cache with multiple cores, but since default L2 cache is a single bank, L2 access could be a bottleneck if the number of cores increases. Since the L2 cache is designed as a module in gem5, it is possible for users to set multiple banks of L2 cache by adding descriptions individually. The RIKEN simulator has been expanded so that the number of banks can be changed just by specifying parameters.

- In gem5, L2 cache requests are handled by FCFS (First Come First Serve) manner. In the A64FX, the L2 cache has a mechanism to keep fairness between requests from each core. In the RIKEN simulator, L2 cache has been extended to allow FCFS and RR (Round Robin) policies to be selected.

- In gem5, the bus width between L1 cache and L2 cache is one parameter, and it is assumed that the transfer throughput from L1 cache to L2 cache and that from L2 cache to L1 cache are the same. On the other hand, in the A64FX, these two transfers throughputs are different. The RIKEN simulator has been extended to specify these two bus widths as different parameters. The same applies to the bus width between the L2 cache and memory.

- Gem5 supports HBM1 as main memory, but HBM2 is not yet supported. The RIKEN simulator added HBM2 parameters based on the HBM1 parameters. Although standard features of gem5 could not completely match the memory interleaving method used with the A64FX, the RIKEN simulator achieved almost the same memory performance as that of the A64FX by combining this feature with other parameters, such as burst length.

2.3.2.2 Evaluation

The RIKEN simulator was evaluated using several programs to find out how accurate it is with the execution time of the A64FX. The evaluation target is the test chip of the A64FX prototype, and does not indicate the performance of the final Fugaku processor. Also, the compiler used for generating the program to be executed was a prototype version of the compiler for the Fugaku from Fujitsu, which is the April 2019 version. It is the same version of the compiler used with the test chip evaluation.

Listing 2.1: Example source code of the kernel program (double addition)

```
subroutine calc01_add_r8(n,iter,dist,y,x1,x2)
  real*8 y(n), x1(n), x2(n)
  integer n, iter, i, j, dist
  do j = 1, iter
     do i = 1, n
        y(i) = x1(i) + x2(i)
     end do
  enddo
end subroutine calc01_add_r8
```

First, we compared the execution times of various kernel programs on a single core with that of the test chip. An example of a kernel program used for evaluation is shown in Listing 2.1. There are four types of kernels: basic arithmetic functions, type conversions, numerical functions, and mathematical functions. The basic arithmetic functions include seven arithmetic operations: addition, subtraction, multiplication, product-sum, division, reciprocal and square root. The type conversions include seven conversions: conversion from double precision to single precision and 32-bit integer and its inverse conversion, and conversion from the double precision of 'aint', 'nint' and 'anint' that are built-in functions of Fujitsu Fortran. The numerical functions include four functions of absolute value, maximum, minimum and sign. The mathematical functions include six functions of 'cos', 'sin', 'exp', 'exp10', 'log' and 'log10'. Evaluation was performed on a total of 24 kernels for double precision.

These kernel programs were compiled with the '-Kfast' option. We compared the execution time on the RIKEN simulator with that on the A64FX test chip. In the Fujitsu compiler, all of these kernels are executed by 8 SIMD for double precision and 16 SIMD for single precision, mathematical functions are inlined, and optimization by software pipeline is applied. Divisions and reciprocals are also calculated using reciprocal instructions that are pipelined rather than using non-pipelined division instructions.

The evaluation results are shown in Figure 2.4 for double precision. The bar graph represents the inverse of operation throughput (the number of cycles required for SIMD-length operation) evaluated by the RIKEN simulator, and corresponds to the left vertical axis. The orange point is the ratio of the difference in execution time between the test chip result and the RIKEN simulator result, and corresponds to the right vertical axis. An execution time difference of 10% indicates that the execution time of the RIKEN simulator is 10% longer than that of the test chip, and -10% indicates that the execution time of the test chip is 10% longer than that of the RIKEN simulator.

Next, in order to evaluate the L2 cache and memory performance in multithreaded execution, the performance of Stream Triad for two data sizes were compared by changing the number of threads.

The results for L2 cache throughput, using Stream Triad for the size within the L2 cache are shown in Figure 2.5. The left graph is results using hardware prefetch, and the right graph is results using software prefetch. The bar graph shows the total L2 cache throughput in the RIKEN simulator when the number of threads is changed from 1 to 12 for the same data size. The orange dots show the percentage difference in execution time between the RIKEN simulator and the A64FX test chip. The horizontal axis corresponds to the number of threads, the left vertical axis corresponds to the throughput, and the right vertical axis corresponds to the percentage difference.

Using hardware prefetch, although the result is relatively scalable with the increase in the number of threads, some of the difference are over 10%, and the variation is large. The A64FX results are more scalable up to 12 threads. When the number of threads is small, the RIKEN simulator is faster than the A64FX. This may be because the L1 hardware prefetch requests the L2 cache without checking the L1 cache in the RIKEN simulator, and the load on the L1 cache may be smaller. On the other hand, there is another problem on implementation of hardware prefetch for L2 cache in the RIKEN simulator. In the RIKEN simulator, the occupancy rate of crossbar between L1 cache and L2 cache is very high due to hardware prefetching to L2 cache, so we are continuing to develop to reduce it.

Using software prefetch, although the result is scalable with the increase in the number of threads, the throughput is saturated around 10 threads in the RIKEN simulator. On the other hand, the A64FX shows scalable performance up to 12 threads. As a result, the execution time difference for 12 threads is slightly larger at 8%, but otherwise it is 2% or less.

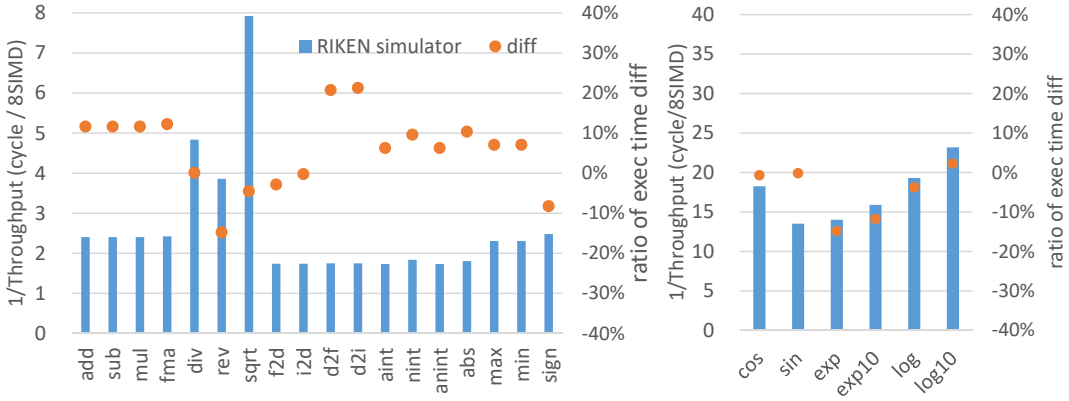

Figure 2.4: Execution throughput of kernel benchmark (double precision) in the RIKEN simulator and the execution time difference ratio of the test chip

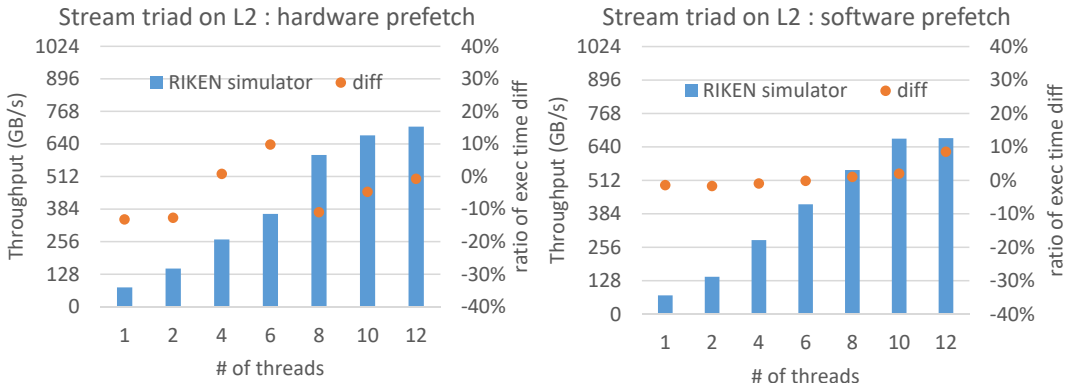

Figure 2.5: L2 cache throughput of Stream Triad in the RIKEN simulator and the difference in execution time with the test chip

2.3.3 Development optimization algorithm for grid graph in Order/Degree problem

We have developed a solver for the Order/Degree Problem as one of the real applications of XMP. The Order/Degree problem is the problem of finding the undirected graph with the smallest diameter and ASPL (Average Shortest Path Length) from the set of undirected graphs that satisfy a given number of vertices (Order) and degree (Degree). The problem is an abstraction of the network topology in large-scale systems such as supercomputers and data centers as a graph theory problem, and it is known that it can be used for designing a network with low latency. The categories of the Order/Degree problem include a general graph in which vertices can be arranged freely and a grid graph in which vertices are arranged on nodes of a two-dimensional grid. In the grid graph, the distance between vertices is defined as the Manhattan distance, which is because a parallel computer system or an intra-chip network that does not allow diagonal wiring is assumed. Since the maximum length of the edge can be set, the design can take into account the maximum length of the network cable, etc. Last year we tackled with general graphs, but this year we do with grid graphs.

We indicate that the symmetry of the graph can be used to improve both the solution search performance of Simulated Annealing and the calculation time. Figure 2.6 shows some examples of the symmetry. The graphs are 6 x 6 grid, degree is 3, and maximum length is 2. Hereinafter, it is represented as (width x height, degree, max length) = (6 x 6, 3, 2). When a graph is rotated (360/g) degrees and becomes the original graph, we define that the graph has the symmetry.

Figure 2.7 shows parts of the evaluation of the solution search performance for grid graph. The "number of groups" on the horizontal axis means the number of symmetries (g). The vertical axis is an ASPL of the actually found graph and a value in parentheses is the diameter. For comparison purpose, dot line is ASPL in random graph. The results show the solution search performance tends to be higher as the value of the number

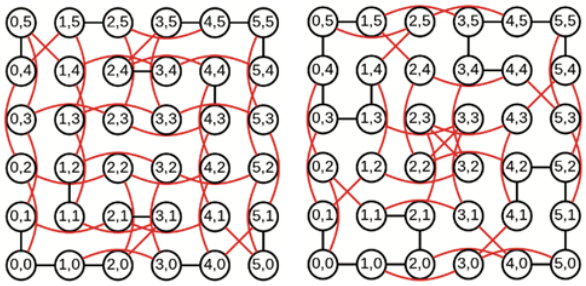

Figure 2.6: Examples of Symmetric Graph

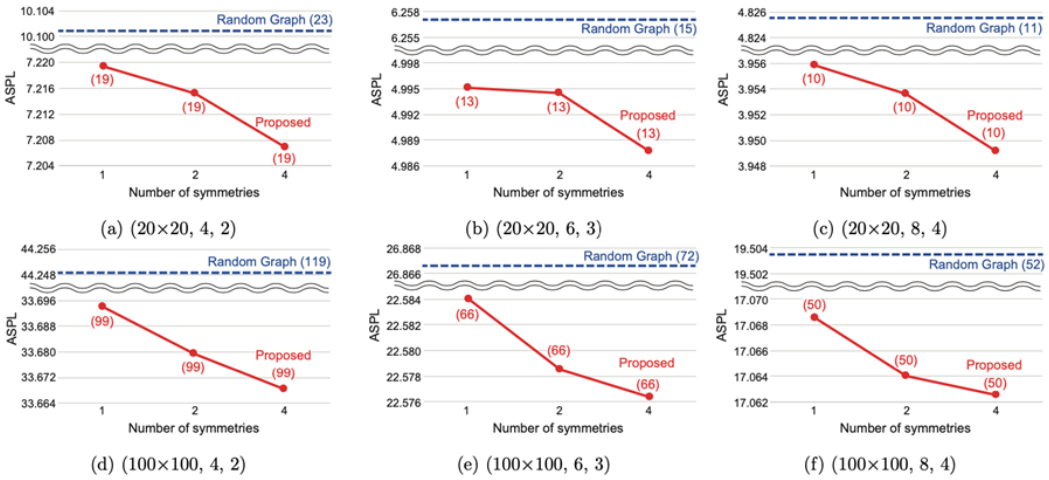

Figure 2.7: Evaluation results of the solution search performance for grid graph

of groups is larger.

We submitted the results to Graph Golf, an international competition on Order/Degree problem, and we won the award in 2019.

2.3.4 HOPE: A Fault-Tolerant Parallel Execution Model Based on Hierarchical Omission

This work presents a new approach to fault-tolerant language systems without a single point of failure for irregular parallel applications. Work-stealing frameworks provide good load balancing for many parallel applications, including irregular ones written in a divide-and-conquer style. However, work-stealing frameworks with fault-tolerant features such as checkpointing do not always work well.

This work proposes a completely opposite work omission paradigm and its more detailed concept as a hierarchical omission-based parallel execution model called HOPE. HOPE programmers' task is to specify which regions in imperative code can be executed in sequential but arbitrary order and how their partial results can be accessed. HOPE workers spawn no tasks/threads at all; rather, every worker has the entire work of the program with its own planned execution order, and then the workers and the underlying message mediation systems automatically exchange partial results to omit hierarchical subcomputations.

Figure 2.8 shows the results of HOPE with faults comparing to HPE and Tascell without faults. Efficiency (upper half) and execution times (lower half) of parallel systems using multiple workers on the K computer, in addition to execution times of serial C programs (lower half). We use 1 to 1024 nodes, each of which has eight cores and employs seven workers. HOPE FT(4) stands for HOPE with FaulT injection into one out of 4 workers.

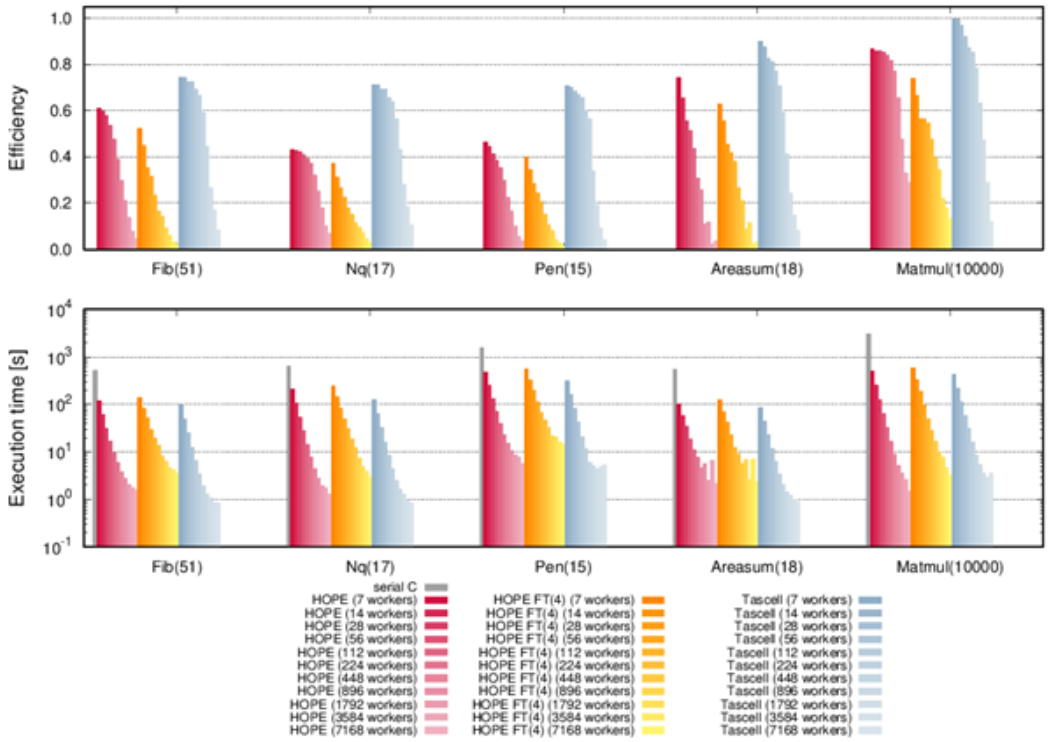

Figure 2.8: Results of HOPE with faults

Even with fault tolerance, the HOPE framework provides parallel speedups for many parallel applications, including irregular ones.

This work was done by Prof. Yasugi, Kyusyu institute of Technology, as a collaboration with our team.

2.4 Schedule and Future Plan

We have supported a production-level XcalableMP compiler, and evaluated a set of XcalableMP benchmark programs. XcalableMP (version 1.x) will be available for the Fugaku users.

We are working on the next version, XcalableMP 2.0, for cutting-edge high performance systems with many-core processors by multitasking with integrations of PGAS model and synchronization models for dataflow/multitasking executions. In this new programming model, the execution of the program is decomposed into several tasks executed according the dependency between tasks. This model enables less overhead of synchronization eliminating expensive global synchronization, overlap between computation and communication in manycore, and light-weight communication by RDMA in PGAS model. Especially for Fugaku, this model allows to exploit parallelism for both tasks and SIMD.

Our long-term goal is to establish the programming model for high-performance computing systems in Post-Moores era. Recently, the slow-down in progress of the semiconductor fabrication technologies is pointed out as a serious problem of the future computer system, and it is said that the post-Moore era is coming in near future after Moores low ends. Since modern high-performance computing systems has improved its performance by advanced semiconductor technologies, it is necessary to reconsider overall structure of computer architecture including hardware and software for the evolution the architecture of system, in order to improve performance in the post-Moores era.

We will extend the task parallel programming model to combine several kinds of accelerators such as GPU, FPGA and special-purpose processors with large-scale general-purpose manycore systems. It enables some task to be offloaded into the accelerators such as FPGA and accelerators as well as each core in modern manycore processor. We consider this configuration as a general global architecture of the future system as some part of system will be specialized for high performance and power efficiency. Our programming model will make it easy to adopt the existing computational science program to the new systems.

2.5 Publications

2.5.1 Articles/Journal

[1] Masahiro Nakao, Tetsuya Odajima, Hitoshi Murai, Akihiro Tabuchi, Norihisa Fujita, Toshihiro Hanawa, Taisuke Boku, Mitsuhisa Sato. "Evaluation of XcalableACC with Tightly Coupled Accelerators/InfiniBand Hybrid Communication on Accelerated Cluster," International Journal of High-Performance Computing Applications, 33(5) (2019).
[2] Ksander Ejjaaouani, Oivier Aumage, Julien Bigot, Michel Mhrenberger, Hitoshi Murai, Masahiro Nakao, Mitsuhisa Sato. "InKS: a programming model to decouple algorithm from optimization in HPC codes", The Journal of Supercomputing, 76(6): 4666-4681 (2020)

2.5.2 Conference Papers

[1] Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato: Parallelization of All-Pairs-Shortest-Path Algorithms in Unweighted Graph. HPC Asia 2020: 63-72
[2] Yuetsu Kodama, Tetsuya Odajima, Akira Asato, Mitsuhisa Sato: Accuracy Improvement of Memory System Simulation for Modern Shared Memory Processor. HPC Asia 2020: 142-149
[3] Yutaka Watanabe, Jinpil Lee, Kentaro Sano, Taisuke Boku, Mitsuhisa Sato, "Design and Preliminary Evaluation of OpenACC Compiler for FPGA with OpenCL and Stream Processing DSL", HPC Asia 2020, workshops, pp. 10-16.
[4] Jinpil Lee, Yutaka Watanabe, Mitsuhisa Sato: OpenMP Task Generation for Batched Kernel APIs. IWOMP 2019: 262-273
[5] Masahiro Yasugi, Daisuke Muraoka, Tasuku Hiraishi, Seiji Umatani, and Kento Emoto: HOPE: A Parallel Execution Model Based on Hierarchical Omission. Proceedings of the 48th International Conference on Parallel Processing (ICPP 2019), Kyoto, Japan. pp. 77:1-77:11, August 2019.

2.5.3 Posters

[1] Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato, Yoshimichi Andoh, Susumu Okazaki. "Performance improvement of MODYLAS using Remote Direct Memory Access on the K computer," International Conference on Parallel Processing, Kyoto, Japan, Aug. 2019.

2.5.4 Invited Talks

[1] Mitsuhisa Sato, "Challenges for Unified Parallel Programming Models for Accelerated Clusters", 4th International Workshop on Performance Portable Programming models for Manycore or Accelerators (P$\hat{3}$MA) in ISC2019, Frankfurt, June 20th, 2019.

2.5.5 Software

- Omni XcalableMP compiler ver. 1.3.3, Nov. 27, 2020 (registered as an R-CCS-supported software)