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)
Hidetoshi Iwashita (Research Associate)
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)
Hiroko Takahashi (Assistant)

2.2 Research Activities

Our team conducts researches and developments on parallel programming models and language to exploit full potentials of parallelism in large-scale distributed memory systems such as the K computer and increase productivity of parallel programming.

In 2017FY, in order to archive these objectives above, we carried out the following researches:
CHAPTER 2. PROGRAMMING ENVIRONMENT RESEARCH TEAM

(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. 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 continued the improvement of Coarray functions in Fortran for the K computer and others.

(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. This research is funded by JST CREST project on post-petascale computing. As we continued working on the language design and the compiler development of XcalableACC, we have been evaluating the performance using several benchmark program and actual application such as QCD. In this year, by additional funding of SPPEXA project which is an international collaboration with French and Germany, we have been working on MUST verification tool to XMP for correctness checking of user applications.

(3) As the post-K computer will be 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 have been working on the performance evaluation on the prototype implementation.

(4) 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 are working on the design of tools for co-design, including the SCAMP profiler for the network of large scale systems.

(5) In order to exploit the full performance of large-scale systems, we had been proposed a multi SPMD programming model, which use multiple programming models across different architectural levels and had been developed a software to realize the proposed programming model by adopting a scientific workflow environment and a PGAS language called XcalableMP (XMP). we continued the improvement such as fault tolerant features, and the evaluation using applications. This work has been carried out under the collaboration of MDLS, France.

(6) We conducted several collaborations on the performance evaluation with JSC, University of Tsukuba, Kyusyu Institute of Technology and other groups. As for performance tuning tools, We continue the collaboration with JSC for a tuning tool Scalasca, which is being developed by JSC, for the K computer. In this year, we carried out the performance study on NEST brain simulator developed by JSC, using Scalasca.

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 made the XcalableMP workshop, seminars or tutorial as follows:

- The 5th XcalableMP workshop (Oct. 31)
- Tutorials on XMP (at CEA, France in September, at Kashiwa in September, at U. Kyushu, Fukuoka, in February). Tutorials consist of both classroom and hands-on learning

In this year, our team has organized the following workshop and mini-symposiums in SIAM Parallel Processing Symposium held in Waseda, Japan:

- Workshop on PGAS programming models: Experiences and Implementations (PGAS-EI), HPC Asia 2017 workshop, Tokyo, Japan.
- PP18 mini-symposium on “Emerging Programming Models for Extreme-Scale Computing”
- PP18 mini-symposium on “Research Compiler Infrastructure for High Performance Computing”
2.3. RESEARCH RESULTS AND ACHIEVEMENTS

2.3.1 XcalableACC for Lattice QCD

Accelerated clusters, which are distributed memory systems equipped with accelerators, have been used in various fields. For accelerated clusters, programmers often implement their applications by a combination of MPI and CUDA (MPI+CUDA). However, the approach faces programming complexity issues. To solve these issues, we develop the XcalableACC (XACC) language, which is a hybrid model of XcalableMP (XMP) and OpenACC. XACC enables programmers to develop applications on accelerated clusters with ease.

Fig. 2.1 shows an XACC concept where execution units are called “node” all of which execute the same program via the SPMD execution model. Each node consists of a host and some accelerators (For simplicity, Fig. 2.1 shows only one accelerator). A distributed array is declared using a “template”, which is a dummy array that represents a global index space. Fig. 2.1, a green square in each node is a part of the distributed array.

The XACC programming model uses XMP, OpenACC and XACC directives. XMP directives declare distributed arrays, divide loop statements and transfer data among host memory. OpenACC directives transfer the distributed arrays between host memory and accelerator memory on the same node and calculate the loop statements divided by XMP on accelerators. XACC directives transfer data among accelerator memories or between accelerator memory and host memory on different nodes.

2.3.1.1 Implementation of Lattice QCD

To evaluate XACC performance and productivity, we implemented a lattice quantum chromodynamics (Lattice QCD) application. Fig. 2.2 shows how to declare distributed arrays of the quark and gluon. In lines 1-8, the quark and gluon structure arrays are declared. The NT, NZ, NY and NX are the numbers of TZYX axis elements. In lines 10-18, distributed arrays are declared where the macro constant values NODES_T and NODES_Z indicate the number of nodes on the T and Z axes. Thus, the program is parallelized on T and Z axes. In lines 16-18, the shadow directives add halo regions to the arrays because each quark and gluon element is affected by its neighboring orthogonal elements. In line 22, the enter data directive transfers the distributed arrays from host memory to accelerator memory.
Fig. 2.3 shows a part of the Wilson-Dirac operator code, which is a main kernel in Lattice QCD. All arguments in WD() are distributed arrays. In XMP and XACC, distributed arrays which are used as arguments must be redeclared in function to pass their information to a compiler. Thus, the align and shadow directives are used in WD(). In line 10, the loop directive parallelizes the outer two loop statements. In line 11, the parallel loop directive parallelizes all loop statements.

Fig. 2.4 shows how to call WD(). The reflect directives are inserted before WD() in order to update own halo region. In line 2, “1:0” in width clause means only the lower halo region is updated because only it is needed in WD(). The u is not updated before the second WD() function because values of u are not updated in WD(). Moreover, the “orthogonal” clause is added because diagonal updates of the arrays are not required in WD().

2.3.1.2 Performance evaluation

This section evaluates the performance of XACC on the Lattice QCD code. For comparison purposes, those of MPI+CUDA and MPI+OpenACC are also evaluated. For performance evaluation, we use the HA-PACS/TCA system which consists of 64 compute nodes and 256 GPUs.

Fig. 2.5 shows the performance results that indicate the time required to solve one CG iteration as well as the performance ratio values that indicate the comparative performance of XACC and other languages. When the performance ratio value of a language is greater than 1.00, the performance result of the language is better than that of XACC. Fig. 2.5 shows that the performance ratio values of MPI+CUDA are between 1.04 and 1.18, and that those of MPI+OpenACC are between 0.99 and 1.04. Moreover, Fig. 2.5 also shows that the performance results of both MPI+CUDA and MPI+OpenACC become closer to those of XACC as the number of nodes increases.
2.3. RESEARCH RESULTS AND ACHIEVEMENTS

2.3.1.3 Productivity evaluation

We developed three Lattice QCD codes using MPI+CUDA, MPI+OpenACC, and XACC. Fig. 2.6 shows our procedure for developing each code where we first develop the code for an accelerator from the serial code, and then extend it to handle an accelerated cluster. As one of metrics for productivity, Delta Source Lines of Codes (DSLOC) is proposed. The DSLOC indicates how the codes change from a corresponding implementation. The DSLOC is the sum of three components: how many lines are added, deleted and modified. When the DSLOC is small, the programming costs and the possibility of program bugs will be small as well. We use the DSLOC to count the amount of change required to implement an accelerated cluster code from a serial code.

Table I shows the DSLOC where lowercase characters correspond to Fig. 2.6. The DSLOC of XACC (b+e) is smaller than MPI+CUDA (a+c) and MPI+OpenACC (b+d). The difference between XACC and MPI+CUDA is 420.0%, and that between XACC and MPI+OpenACC is 39.4%.

2.3.1.4 Conclusion

In this study, we evaluated performance and productivity of XACC through the implementation of the Lattice QCD code. For comparison purposes, we also implemented the Lattice QCD code using MPI+CUDA and MPI+OpenACC. We began by evaluating all performance on the HA-PACS/TCA system using up to 64 compute nodes and 256 GPUs. The results showed that performance results of XACC is almost equal to those of MPI+CUDA and MPI+OpenACC when large numbers of compute nodes are used. Next, we evaluated productivity using the DSLOC metric. The result indicates that XACC provides the best productivity of all.

2.3.2 XcalableMP 2.0 for task-parallelism in distributed memory systems

XcalableMP (XMP) is a Partitioned Global Address Space (PGAS) style of language for describing large-scale scientific code for parallel systems with distributed memory architecture. Currently, we are working on the next version of XcalableMP specification, called XcalableMP 2.0. The important topics for XcalableMP 2.0 might include the support for manycore clusters, that is multitasking with integrations of PGAS model and synchronization models for dataflow/multitasking executions. For this issue, we propose an extension by a new directive called tasklet to describe multithreaded and dataflow programming
model. We expect this model will enable less overhead of synchronization eliminating expensive global synchronization, overlap between computation and communication in manycore, and lightweight communication by RDMA in PGAS model.

### 2.3.2.2 OpenMP and XMP Tasklet Directive

Although OpenMP originally focuses on work sharing for loops as the parallel for directive, OpenMP 3.0 introduces task parallelism using the task directive. It facilitates the parallelization where work is generated dynamically and irregularly as in recursive structures or unbounded loops. The depend clause on the task directive is supported from OpenMP 4.0 and specifies data dependencies with dependence-type in, out, and inout. Task dependency can reduce the global synchronization of a thread team because it can execute fine-grained synchronization between tasks through user-specified data dependencies.

To support task parallelism in XMP as in OpenMP, the tasklet directive \(^1\) is proposed. Fig. 2.7 describes the syntax of the tasklet, taskletwait, and tasklets directives for the multi-tasking execution in XMP. The tasklet directive generates a task for the associated structured block on the node specified by the on clause, and the task is scheduled and immediately executed by an arbitrary thread in the specified node if there is no task dependency. If it has any task dependencies, the task execution is postponed until all dependencies are resolved. These behaviors occur when these tasks are surrounded by tasklets directive. The taskletwait directive waits on the completion of the generated tasks on each node.

### 2.3.2.2 Multi-tasking Execution Model in XcalableMP between nodes

In OpenMP, the task dependency in a node depends on the order of reading and writing to data based on the sequential execution. Therefore, the OpenMP multi-tasking model cannot be applied to describe the dependency between tasks running in different nodes since threads of each node are running in parallel. In OmpSs, interactions between nodes are described through the MPI task that is executing MPI communications. Task dependency between nodes is guaranteed by the completion of MPI point-to-point communication in tasks. While this approach can satisfy dependencies between nodes, it may cause further productivity degradation because it forces users to use a combination of two programming models that are based on different description formats. Therefore, we propose new directives for communication with tasks in XMP, and they enable users to write easily the multi-tasking execution for clusters by only using language constructs.

\(^1\)There is the task directive in XMP, it is different from OpenMP’s one.
2.3. RESEARCH RESULTS AND ACHIEVEMENTS

```c
int A[3], B;
#pragma xmp nodes P(3)
#pragma xmp template T(0:2)
#pragma xmp distribute T(block) onto P
#pragma xmp align A[i] with T(i)

#pragma xmp tasklets
{
#pragma xmp tasklet out(A[0]) on P(1)
  A[0] = 0; /* taskA */
#pragma xmp tasklet gmove in(A[0]) out(A[1])
  A[1] = A[0]; /* taskB */
#pragma xmp tasklet gmove in(A[1]) out(B) on P(2:3)
  B = A[1]; /* taskC */
}
```

Figure 2.8: Example of the tasklet and tasklet gmove directives.

We propose two new directives, tasklet gmove and tasklet reflect directives to describe interactions between nodes in tasks by point-to-point communication for the inter-node data dependency, and these communications are synchronized only between sender and receiver of communication in each task.

The tasklet gmove directive copies the variable of the right-hand side (rhs) into the left-hand side (lhs) of the associated assignment statement for local or distributed data in tasks. If the variable of the rhs or the lhs is the distributed data, this directive may execute communication. The communication is detected and performed automatically by the XMP runtime system. This directive must be executed by all nodes in the current executing node set specified by the on clause. When in, out, or inout clause presents on the tasklet gmove directive, the generated task have each data dependency in a node similar to the tasklet directive.

Fig. 2.8 shows an example of the tasklet and tasklet gmove directives. In this example, array A[] of length 4 is distributed to 2 nodes as equal-size contiguous blocks; therefore, node 1 and 2 can deal with indices 0 to 1 and 2 to 3, respectively. This code has three tasks, taskA and taskB are executed on node 1, and taskB and taskC are executed on node 2, that these execution nodes are specified by the on clause. There are flow dependency between taskA and taskB by A[0], and taskB and taskC by B. After the execution of taskA, taskB on node 1 sends A[0] to node 2 which is determined by the execution nodes set by the on clause. In node 2, taskB receives A[0] from node 1 in B. When finishing the receive operation in taskB, taskC is immediately started on node 2 because the flow dependency by B is satisfied.

The tasklet reflect directive is a task-version of reflect operation. It updates halo regions of the array specified to array-name in tasks. In this directive, data dependency is automatically added to these tasks based on the communication data because the boundary index of the distributed data is dynamically determined by XMP runtime system.

2.3.2.3 Preliminary Performance Evaluation on Oakforest-PACS

We have designed a simple code translation algorithm from the proposed directives to XMP runtime calls with MPI and OpenMP. The tasklets directive is converted to OpenMP parallel and single directives. The execution node is determined by the on clause that is translated to an if statement. The tasklet gmove and tasklet reflect directives are converted to MPI_Send/Recv() and these MPI functions are executed in OpenMP tasks with data dependency specified by users. However, in case of the MPI blocking call such as MPI_Send/Recv() in these codes, the deadlock may occur depending on the task scheduling mechanism by the combination of MPI and OpenMP. To prevent this problem, we used the MPI asynchronous communications such as MPI_Isend/Irecv(), MPI_Test(), and the OpenMP taskyield directive.

We have evaluated the performance on our Oakforest-PACS. We select the Flat and Quadrant mode for KNL. While the Intel Xeon Phi 7250 has 68 cores, 64 cores usage per a node is recommended in this system. As our benchmark program, we used Blocked Cholesky factorization program which calculates a decomposition of a Hermitian, positive-definite blocked matrix as into the product of a lower triangular matrix and its conjugate
transposed. We implemented two versions in different parallelization approaches: “Parallel Loop” and “Task”, in MPI and OpenMP. The “Parallel Loop” version is the conventional barrier-based implementation described by work sharing for loops by the parallel for directive and independent tasks using the task directive without the depend clause. Although this version of the blocked Cholesky factorization is applied the overlap of the communication and computation at the process level, it performs the global synchronization in work sharing. The “Parallel Loop” version of Laplace equation solver is not include the overlap of the communication and computation. The “Task” version is implemented using our proposed model by task dependency using the depend clause instead of global synchronization.

We measured the performance using from one to 32 nodes, 1 process per node, 64 cores per process, and 1 thread per core. As problem size of these benchmarks, the matrix size is $32768 \times 32768$ and the block size is $512 \times 512$ in double precision arithmetic. The matrix is distributed by two-dimensional block-cyclic data distribution in blocked Cholesky factorization and two-dimensional block data distribution in Laplace equation solver.

![Graphs](image_url)

(A) Performance

(B) Breakdown at 32 nodes execution

Figure 2.9: Performance and breakdown of the blocked Cholesky factorization on Oakforest-PACS system.

Figs. 2.9 show the performance and breakdown of the blocked Cholesky factorization on the Oakforest-PACS. The breakdown indicates the average time of each operations performed on all threads because tasks executed on threads defer each time the program is executed. The “wait” in breakdown represents the waiting time of the thread including the global synchronization. In Fig. 2.9 (A), the “Task” version show better performance than the barrier-based implementations, “Parallel Loop” version. The reason why “Task” version outperforms “Parallel Loop” version is that the global synchronization spends more cost in work sharing of loops and among tasks as the “wait” shown in Fig. 2.9 (B). The relative performances of “Task” version against the barrier-based implementation, “Parallel Loop” version, is 123%.

2.3.2.4 Communication Optimization for Many-core Clusters

In the version reported in the previous subsection, the communication are executed in MPI_THREAD_MULTIPLE as MPI thread-safety level since tasks executed on threads may communicate simultaneously. Our basic performance analysis of point-to-point communication using OSU Micro-Benchmarks indicates that the performance of multi-threaded communication with MPI_THREAD_MULTIPLE degrades when increasing the number of threads, compared to a single-threaded communication.

According to the observation above, we optimize the communication by delegating all communications to the communication thread. To delegate the communication to a single thread, we make a global queue accessible by all threads so that the tasks enqueues the communication requests into its queue and wait for the communication completion. Meanwhile, the communication thread dequeues the requests for communication to perform the requested communications, and check the communication completion, respectively. The communication thread executes only the communication, and the other threads perform computation tasks.

In Figs. 2.9 (A), “Task (opt)” version are higher than that of the multi-tasking execution with MPI_THREAD_MULTIPLE. The reason is that the communication time is reduced compared with the “Task” version as shown in Figs. 2.9 (B). The relative performance against the barrier-based implementation, “Parallel Loop” in Figs. 2.9, are improved to 138% and 141%, respectively.
2.3. RESEARCH RESULTS AND ACHIEVEMENTS

In near future, we expect the performance of MPI_THREAD_MULTIPLE will also be improved as the design and implementation is getting matured for parallel manycore system. Currently, we are investigating a lower-level communication API for efficient one-sided communication of PGAS operations in multithreaded execution environment.

2.3.3 SPPEXA MXY project

MYX (MUST correctness checking for YML and XMP programs) is an international collaboration for correctness checking of exascale parallel programs.

MUST was originally developed by RWTH Aachen as a tool for checking MPI programs, and extended to support emerging PGAS and workflow programming models. Fig. 2.10 shows the overview of MYX collaboration.

In this collaboration, we defined and implemented the XMPT interface, which is an XMP tools interface modeled after the OpenMP tools interface, OMPT. The definition of XMPT will finally be included in the XMP specification.

According to XMPT, the XMP runtime notifies an interested tool about any encountered directive as XMPT events, which are designed to be used by correctness as well as performance analysis as shown in Fig. 2.11.

For correctness analysis, XMPT enables productivity improvements in programming for Exascale by means of scalable correctness checking of XMP-programs, such as one-sided communications and global data accesses, by analyzing the semantics expressed by the XMP control flow directives and identifying semantic issues. For performance analysis, XMPT enables tracing tools like Score-P to log event information and use the data to visualize the performance of execution.

On the other hand, we had proposed a multi SPMD programming framework, where multiple SPMD programs are integrated into a workflow. In our implementation of the multi SPMD programming framework, YML had been employed to realize a workflow programming model, and XMP had been supported to describe
the tasks of the workflow. In order to incorporate XMP checked by MUST into the XMP/YML multi SPMD programming framework, we have investigated how to apply MUST correctness checking to the worker programs invoked by a master scheduler program.

### 2.3.4 Performance study of NEST on K computer using Scalasca

The group on NEST, a brain simulator for Neuronal sciences is engaged in performance analysis of the simulator (5th-generation kernel, in the development community’s terminology) and collaborating, under the MOU with Juelich Supercomputing Centre (JSC), with scientists and Ph.D. students from the ICT group at RIKEN, the computational neuroscience division and JSC at Juelich Research Center in Germany.

To collect the performance data at scale, up to 82,944 nodes that K computer provides, we’ve chosen Scalasca as our tool for investigation (the porting work of Scalasca to K computer is a collaboration work between the Application Support division at JSC and Programing environment research group and it has been actively maintained).

As a first step toward the goal of this project, we’ve collected data by running the well-studied neuronal-network model without instrumentation to use as a base-line date and the new 5th-gen kernel at the time of the experiment, demonstrated its newly implemented and introduced algorithms and data structures improve the expected good scalability up to the full-node scale of K computer. This set of data is used in the original research paper entitled “Extremely scalable spiking neural network simulation code: from laptops to exascale computers”, in Frontiers in Neuroinformatics [1] and the press release on the results were published by the Human Brain Project.

Next, we manually instrumented NEST to minimize the sizable overhead incurred if automatic instrumentation by compiler is chosen, and collected and analyzed the profile data set. (about 20% of execution time increase due to manual instrumentation) From the data, we’ve determined the MPI communication cost is about 1% of wall-clock time even in an experiment where neurons scattered across the 82,944 nodes.

Another finding from the profile data is that the computing cost increase with the increase the number of nodes involved. We plan to come back to this and investigate when the stable simulation code base which includes the 5th generation kernel is released.

### 2.4 Schedule and Future Plan

We have been developing a production-level XcalableMP compiler and make it available for the K computer’s users as well as the users of conventional clusters.

---

Towards exascale computing, there are two major approaches, manycore and GPU. We are already working on XcalableACC as a solution for accelerator-based system, which has been explored in the JST CREST project. As well as dissemination of the current version of XcalableMP, we are now moving forward to the next version, XcalableMP 2.0, which includes new features such as the integration of dynamic tasking with PGAS programming model for large-scale manycore systems, especially for the post-K.

And, for the coming Post-Moore era, we will start a study on parallel programming models for the architecture of the Post-Moore. As a first step, we will exploit the programming models for FPGA cluster as one of the candidates for such architecture.

2.5 Publications

2.5.1 Journal Articles


2.5.2 Conference Papers


2.5.3 Invited Talks


2.5.4 Posters and Presentations


2.5.5 Patents and Deliverables

- Omni XcalableMP compiler ver. 1.3 (registered as an R-CCS-supported software)