Chapter 4

HPC Usability Research Team

4.1 Members

Toshiyuki Maeda (Team Leader)

Masatomo Hashimoto (Research Scientist)

Peter Bryzgalov (Research Scientist)

Itaru Kitayama (Technical Staff I)

Yoshiki Nishikawa (Visiting Scientist, University of Tokyo)

Yves Caniou (Visiting Scientist, Université Claude Bernard Lyon 1)

Judit Gimenez (Visiting Scientist, Barcelona Supercomputing Center)

Sameer Shende (Visiting Scientist, University of Oregon)

Fabien Delalondre (Visiting Scientist, École polytechnique fédérale de Lausanne)

Pramod Kumbhar (Visiting Scientist, École polytechnique fédérale de Lausanne)

Tatsuya Abe (Visiting Scientist, Chiba Institute of Technology)

Sachiko Kikumoto (Assistant)

Yumeno Kusuhara (Assistant)

4.2 Research Activities

The mission of the HPC Usability Research Team is to research and develop a framework and its theories/technologies for liberating large-scale HPC (high-performance computing) to end-users and developers. In order to achieve the goal, we conduct research in the following three fields:

4.2.1 Computing portal

In a conventional HPC usage scenario, users live in a closed world. In other words, users have to play roles of software developers, service providers, data suppliers, and end users. Therefore, a very limited number of skilled HPC elites can enjoy the power of HPC, while the general public sometimes gives a suspicious look to the benefit of HPC. In order to address the problem, we are designing and implementing a computing portal framework that lowers the threshold for using, providing, and aggregating computing/data services on HPC systems, and liberates the power of HPC to the public.

4.2.2 Virtualization

Virtualization is a technology for realizing virtual computers on real (physical) computers. One big problem of the above mentioned computer portal that can be used by wide range of users simultaneously is how to ensure safety, security, and fairness among multiple users and computing/data service providers. In order to solve the problem, we plan to utilize the virtualization technology because virtual computers are isolated from each other, thus it is easier to ensure safety and security. Moreover, resource allocation can be more flexible than the conventional job scheduling because resource can be allocated in a fine-grained and dynamic way. We also study lightweight virtualization techniques for realizing virtual large-scale HPC for test, debug, and verification of computing/data services.

4.2.3 Program analysis/verification

Program analysis/verification is a technology that tries to prove certain properties of programs by analyzing them. By utilizing software verification techniques, we can prove that a program does not contain a certain kind of bug. For example, the byte-code verification of Java VM ensures memory safety of programs. That is, programs that pass the verification never perform illegal memory operations at runtime. Another big problem of the above mentioned computing portal framework is that one computing service can be consists of multiple computing services that are provided by different providers. Therefore, if a bug or malicious attack code is contained in one of the computing services, it may affect the whole computing service (or the entire portal system). In order to address the problem, we plan to research and develop software verification technologies for large-scale parallel programs. In addition, we also plan to research and develop a performance analysis and tuning technology based on source code modification history.

4.3 Research Results and Achievements

4.3.1 Design and Implementation of a Computing Portal Framework for HPC

In FY2015, we enhanced the computing portal framework developed in FY2012-FY2014 so that the users of the K computer are able to build and publish their own computing portal with their own authority and computing resources on the K computer. More specifically, we modified the computing portal framework so that it is able to directly communicate with the login nodes of the K computer. In addition, the framework is able to generate SSH public keys (and hide their corresponding SSH private keys inside) so that the users of the K computer are able to associate the generate public keys to their accounts. Thus the users are able to launch their own computing portal by installing/deploying the framework on their servers and registering the SSH public keys generated by the framework to their K accounts.

In FY2016, we will continue to develop the computing portal framework. Especially, we will create a VM image in which the framework is installed and ready-to-run so that the users of the K computer is able to launch their own computing portal with the framework by simply deploying the VM image to, e.g., AWS. In addition, we will also investigate to develop a Docker (`https://www.docker.com`) container image.

4.3.2 Virtualization Techniques

Lightweight Virtualization for Testing/Debugging Parallel Programs

Writing a program which makes full use of massively parallel HPC environments (e.g., the K computer) is extremely difficult because debugging parallel programs is extremely difficult because of inherent non-determinacy of parallel programs and hard-to-reproduce bugs. Moreover, writing massively parallel programs also tend to suffer from a performance problem. For example, even if a program scales well on a PC cluster system whose size is small-to-moderate, the program may not scale on massively parallel HPC systems. Even worse, the performance may severely degrade and will be worse than on a small PC cluster system or even a single PC. Actually, this is not uncommon and the reason is that communication costs between computing nodes may largely vary and sometimes incurs unacceptable heavy overheads.

In order to address the abovementioned problem, we have been developing a lightweight network virtualization system for testing/debugging programs for massively parallel programs without actually using real massively parallel HPC environments. With our system, users can run several hundreds of virtual computing nodes on a single physical computing node.

There are two key ideas in our system: library-hooking and decentralized management of routing information. Library-hooking is a kind of virtualization technology which intercepts function calls for system operations, and modify their parameters and/or return values in order to trick the programs as if they run on in isolated multiple computing nodes, even though they run on a single physical computing node. More specifically, in our lightweight virtualization system, we mainly hook network related operations (and some file I/O) from user programs. One benefit of the library-hooking approach is that it introduces little overheads to program execution (compared to other virtualization techniques, e.g., CPU level virtualization, OS level virtualization, and so on) because it can be achieved by user-level operations and requires no interaction with OS.

When implementing a lightweight network virtualization system, decentralized management of routing information is necessary in order to avoid maintaining routing information in a single or a few physical nodes. Our lightweight virtualization system has to manage routing information by itself because it virtualizes network environments. If the routing information is managed in a single physical node, all the other physical nodes have to ask the single node in order to correctly route network packets from one virtual node to another. Therefore, when the numbers of virtual nodes and physical nodes are huge, the single node will become a performance bottleneck and severely degrade the overall performance of our lightweight virtualization system.

More specifically, our lightweight virtualization system statically distributes the information which virtual node runs on which physical node before executing user programs on virtual computing nodes. In ordinary HPC systems, it is uncommon that computing nodes are directly allocated during job execution. In addition, in order to virtualize port numbers, our lightweight virtualization system let physical nodes exchange the information about virtualized port numbers when one virtual node on one physical node communicates with another virtual node on another physical node.

Based on the abovementioned approach, in FY2012-FY2014, we implemented the prototype of our lightweight virtualization system and it successfully ran on the K computer under a restricted environment. More specifically, it successfully ran 20000 virtual computing nodes on 1000 physical computing nodes. In theory, however, it must be able to run more virtual computing nodes on a single physical computing node and run on more physical computing nodes, but this was impossible because the K computer restricts the number of user processes on a physical computing node and the operating system kernel of the computing nodes of the K computer has a serious fault which is related to memory management.

In order to avoid the abovementioned restriction of the K computer and run more number of virtual computing nodes, in FY2015, we implemented workarounds for the prototype of our lightweight virtualization system. More specifically, we refactored our prototype implementation and the amount of memory usage were reduced to about 20 % of the original one. We also modified the prototype implementation in order to keep it from accessing the /etc/hosts file excessively when establishing connections, by directly passing IP addresses to mpirun. This modification addressed the problem of memory imbalance among physical computing nodes. In addition, we upgraded the version of OpenMPI in order to reduce the amount of memory usage.

In FY2016, we will continue the development of our system and evaluate it with more numbers of virtual computing nodes and physical computing nodes. In addition, we plan to study an approach of tricking performance profiling tools so that they feel as if they run on real computing nodes and emit profiling data which represents characteristics of real massively-parallel computing environments.

Container Technologies for HPC

Container technologies are a kind of lightweight virtualization technology. Although they tend to be less efficient than the library-hooking approach described in the previous section (Sec. 4.3.2), they provide more complete image of virtual execution environments. For example, Docker (`https://www.docker.com`) provides multiple isolated virtual Linux execution environments on a host Linux system. Because Docker is built and depends on several functionalities provided by the Linux kernel, it is not able to host non-Linux virtual execution environments unlike full-virtualization technologies (e.g., KVM, QEMU, and so on), but far more efficient than them.

One big problem of the current typical HPC systems compared to today's so-called cloud services from viewpoint of software developers/publishers is that the HPC systems are less flexible and/or responsive. For example, they are not allowed to install and/or modify system/middleware programs in the HPC systems, while the cloud services provide fully-virtualized environments to them and they can freely modify the environments. In addition, the typical HPC systems are operated with conventional batch schedulers and it sometimes takes time to launch jobs, while the cloud services launch virtual execution environments instantly when requested by them.

The reason why the conventional HPC systems are less flexible and/or responsive is that their primary purpose is to compute scientific applications efficiently as much as possible, thus the overheads that may be introduced by utilizing full virtualization technologies are unacceptable.

On the other hand, as described above, the recent advance in the container technologies achieves very small overheads yet provides sufficiently flexible virtual execution environments, thus we predict that the container technologies will play important role in forthcoming HPC usage.

In FY2015, we utilized Docker to improve the usability of K-scope, which is a Fortran source code analysis tool developed by Software Development Team of AICS. More specifically, we created a Docker container image in which K-scope is installed so that users are able to use K-scope without manually installing it. In addition, we also extended K-scope so that users are able to analyze their programs seamlessly on the remote server without modifying their source code and/or build scripts (please note that, in FY2014, we implemented the extension of K-scope so that users are able to analyze their programs on the remote server without installing the analysis engine (more specifically, XcalableMP, which is developed of Programming Environment Research Team of AICS), but users may have to modify their source code and/or build scripts because the paths on the local machine of users and the remove server may not match).

4.3.3 Program Verification and Analysis

Memory Consistency Model-Aware Program Verification

A memory consistency model is a formal model that specifies the behavior of the memory that is shared and simultaneously accessed by multiple threads and/or processes. Under the recent multicore CPU architectures and shared memory multithread/distributed programming languages (e.g., Java, C++, UPC, Coarray Fortran, and so on), the shared memory sometimes behaves in an unexpected way because they adopt relaxed memory consistency models. For example, under some relaxed memory consistency models, the effects of the memory operations performed sequentially by one thread (e.g., A $\rightarrow$ B) may be observed in a different order by the other threads (e.g., B $\rightarrow$ A). Moreover, the threads may not agree on the orders of the effects of the memory operations (e.g., one thread observes A $\rightarrow$ B, while the other observes B $\rightarrow$ A, and so on) they observe. The reason why the recent CPUs and shared memory languages adopt relaxed memory consistency models is that enforcing sequential (non-relaxed) memory consistency incurs huge synchronization overheads among a large number of the threads/nodes that share a single address memory space.

From the viewpoint of program verification, there are two problems in handling relaxed memory consistency models. First problem is that the conventional program verification does not consider relaxed memory consistency models. Thus, they cannot be applied directly to relaxed memory consistency models because they may yield false results. Second problem is that there exist various kinds of relaxed memory consistency models and each CPU architecture/each programming language adopts different memory consistency models. Thus, it is very tedious to define and implement program verification for each CPU and programming languages of relaxed memory consistency models.

To address these problems, we have been studying three approaches. First approach is to define a new formal system which is able to represent various relaxed memory consistency models. More specifically, we define a very relaxed memory consistency model as a base model. Then, we define various memory consistency models as additional axioms on the base model. In fact, we are able to define a broad range of memory consistency models from CPUs to shared-memory programming languages (e.g, Intel64, Itanium, UPC, Coarray Fortran, and so on), in the single formal system.

Second approach is to design and implement a model checker that supports various relaxed memory consistency models based on the formal model of the abovementioned first approach. More specifically, we define a non-deterministic state transition system with execution traces where each execution trace represents a possible permutation of instruction executions. Roughly speaking, given a target program, our model checker explores all the reachable states in the non-deterministic transition system of the target problem for all the possible execution traces (that is, permutations of instructions). In our model checker, memory consistency models can be defined as constraint rules on execution traces. For example, the sequential consistency model can be defined as a constraint which allows no permutation on the execution traces. With our model checker, we were able to verify the examples programs of the specification manuals of the memory consistency models of Itanium and UPC.

Third approach is to define a program logic for a shared-memory parallel process calculus under a relaxed memory consistency model. More specifically, we define an operational semantics for the process calculus, and then define a sound (and relatively-complete) logic to the semantics. There are two key ideas in our program logic. First idea is that a program is translated into a dependence graph among instructions in the program, and the operational semantics and the logic are defined in terms of the dependence graph. One advantage of handling dependence graphs is that while loops, branch statements, and parallel composition of processes can be handled in a uniform way. In addition, another advantage is that multiple memory consistency models can be handled by adopting different translation approaches for each memory consistency model. Second idea is that we introduce auxiliary variables in the operational semantics that temporarily buffer the effects of memory operations.

In FY2015, we further improved the implementation of McSPIN (developed from FY2013), and studied the memory consistency model of the programming language Chapel by request from a research developer of Chapel. Moreover, we also studied several memory management algorithms on various memory consistency models with external researchers.

Evidence-Based Performance Tuning

To get the maximum of HPC systems, it is inevitable to optimize the performance of applications. However, performance tuning for massively parallel HPC systems is very difficult because it is not apparent how to improve programs except for highly skilled programmers. In addition, generally speaking, modifying correctly working programs is a bothering task from the viewpoint of developers. Thus, performance tuning requires experienced craftsmanship, and relies on intuition and experience.

In order to address the problem, we have been working on evidence-based performance tuning. More specifically, we store the results of performance profiling in a database where the results are associated with source code modification history. With the database, developers are able to know, for example, what kinds of optimization were applied in the past, what kinds of optimization are effective for improving a certain performance profiling parameter, and so on.

In FY2015, we tried to increase the number of tuning cases in order to conduct detailed evaluation and improve accuracy of the analysis, but it turned out that it is hard to collect data directly because we could not find any researcher/developers who have such the data in and out of AICS. To work around the problem, we studied an approach of predicting performance of programs only from their source code modification history. In fact, we analyzed several thousands of Fortran projects registered in GitHub (`https://github.com`).

In FY2016, we will continue to try to increase the number of tuning cases in order to conduct detailed evaluation and improve accuracy of the analysis. More specifically, we will manually develop a training set by using the results of analysis of the Fortran projects obtained in FY2015, in cooperation with the two members of Software Development Team of AICS.

Python-Based Aggregation of Multiple Software for HPC

In the world of HPC, programs are usually written in somewhat old-fashioned programming languages such as Fortran/C/C++ for historical reasons, thus writing programs for HPC is painful because we cannot use useful features of modern sophisticated programming languages. On the other hand, it is not realistic so far to write a whole program in modern programming languages because of performance problems.

In order to address the problem and achieve both productivity of program development and performance of program execution, we studied an approach of using Python for writing HPC applications. More specifically, we write non-performance critical large part of a program in Python, and performance critical small part in Fortran/C/C++. The reason why we choose Python is that Python provides a rich set of foreign language interfaces. For example, Fortran programs can be interfaced with f2py (NumPy: `http://www.numpy.org/`), C programs can be interfaced with ctypes and Cython, and C++ programs can be interfaced with Boost.Python and Cython.

In FY2013, we modified EigenExa (a high performance Eigen-solver developed by the Large-scale Parallel Numerical Computing Technology research team of AICS) so that it can be used as a shared library and a Python module (these modifications were feedbacked to the upstream). In addition, integration of Lotus (a quantum chemistry library developed by Tomomi Shimazaki, the Computational Molecular Science research team of AICS) and EigenExa were ongoing mainly by Tomomi Shimazaki.

In FY2014 and FY2015, in collaboration with Tomomi Shimazaki, a non-performance critical large part of Lotus were refactored and written in Python. With the refactoring, we were able to utilize various existing libraries (e.g., EigenExa: `http://www.aics.riken.jp/labs/lpnctrt/EigenExa_e.html`, SMASH: `http://smash-qc.sourceforge.net/`, ASE: `https://wiki.fysik.dtu.dk/ase/`, etc.) in Lotus, and demonstrated that the features of Lotus can be easily extended. More specifically, we extended Lotus by request of Yukio Kawashima of the Computational Chemistry research unit of AICS with only several tens of lines of code addition. We further refactored Lotus by using Cython in FY2015.

Porting Performance Analysis Tools to the K computer

Because massively parallel supercomputers, such as the K computer, are very different from single computer systems or small size cluster systems, simply porting existing applications to the K computer typically does not work due to performance problems (many existing conventional applications do not consider massively-parallel systems). Therefore, performance profiling is necessary to understand the behaviors of applications on massively parallel systems and tune the applications.

To address the problem, we are porting/deploying existing performance analysis tools to the K computer, in cooperation with external research institutes. In FY2014, we ported Scalasca (`http://www.scalasca.org/`) and Extrae (`https://www.bsc.es/computer-sciences/extrae`) to the K computer. Scalasca was ported in cooperation with a research team of Jülich Supercomputing Centre, and Extrae was ported in cooperation with a research team of Barcelona Supercomputing Center. Using the ported tools, we actually analyzed the behavior of ABySS, a parallel genome sequence assembler (`http://www.bcgsc.ca/platform/bioinfo/software/abyss`). We also analyzed the behavior of SIONlib, a parallel I/O library, in cooperation with Jülich Supercomputing Centre, and deployed it on the K computer.

In FY2015, we continued to port/deploy existing performance analysis tools to the K computer. Especially, we ported Eclipse PTP (`https://eclipse.org/ptp/`), which is an extension framework of Eclipse (`https://eclipse.org/`) for parallel program development/execution, to the K computer, in cooperation with a research team of University of Oregon. In addition, we modified and integrated Extrae and SIONlib so that Extrae is able to use SIONlib for its I/O processing (this should be useful for handling very large trace data).

4.4 Schedule and Future Plan

In FY2016, we will continue to develop the computing portal framework. Especially, we will create a VM image in which the framework is installed and ready-to-run so that the users of the K computer is able to launch their own computing portal with the framework by simply deploying the VM image to, e.g., AWS. In addition, we will also investigate to develop a Docker (`https://www.docker.com`) container image.

Regarding the virtualization technologies, we will continue the development of our lightweight network virtualization system and evaluate it with more numbers of virtual computing nodes and physical computing nodes. In addition, we plan to study an approach of tricking performance profiling tools so that they feel as if they run on real computing nodes and emit profiling data which represents characteristics of real massively-parallel computing environments.

Regarding the program verification and analysis, we will pursue our evidence-based performance tuning approach. More specifically, we will continue to try to increase the number of tuning cases in order to conduct detailed evaluation and improve accuracy of the analysis. More specifically, we will manually develop a training set by using the results of analysis of the Fortran projects obtained in FY2015, in cooperation with the two members of Software Development Team of AICS.

In addition to the above mentioned individual research topics, we also design/implement integration of the research results of the virtualization technologies and the software verification with the computing portal.

4.5 Publications

Journal Articles

[1] Tatsuya Abe and Toshiyuki Maeda. "A General Model Checking Framework for Various Memory Consistency Models". In: *International Journal on Software Tools for Technology Transfer* (2016). To be published.

Conference Papers

[2] Tatsuya Abe and Toshiyuki Maeda. "Towards a Unified Verification Theory for Various Memory Consistency Models". In: *Proc. of the 6th Workshop on Syntax and Semantics of Low-Level Languages (LOLA 2015).* 2015.

[3] Masatomo Hashimoto et al. "Extracting Facts from Performance Tuning History of Scientific Applications for Predicting Effective Optimization Patterns". In: *Proc. of the 12th Working Conference on Mining Software Repositories (MSR 2015).* 2015.

[4] Itaru Kitayama, Brian J. Wylie, and Toshiyuki Maeda. "Execution Performance Analysis of the ABySS Genome Sequence Assembler using Scalasca on the K computer". In: *Proc. of ParCo2015.* 2015.

[5] Tomomi Shimazaki, Masatomo Hashimoto, and Toshiyuki Maeda. "Developing a High Performance Quantum Chemistry Program with a Dynamic Scripting Language". In: *Proc. of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SEHPCCSE15).* 2015.

Invited Talks

[6] Toshiyuki Maeda. *Trial Study on Development of a Quantum Chemistry Library with Modern Programming Techniques.* High Performance Computing Chemistry (HPCC) 2015. 2015.

Posters and Presentations

[7] Tatsuya Abe and Toshiyuki Maeda. *Research on Program Verification Considering Various Memory Consistency Models.* The 13rd Dependable System Workshop (DSW2015). 2015.

[8] RIKEN AICS HPC Usability Research Team. *HPC Usabiilty Research Team.* Exhibition of International Supercomputing Conference (ISC) High Performance. 2015.

Patents and Deliverables

[9] Tatsuya Abe. *McSPIN.* `https://bitbucket.org/abet/mcspin`.

[10] Peter Bryzgalov. *Eclipse PTP ported for the K computer.* `https://github.com/pyotr777/EclipsePTP_PJM_TSC/`.