# Chapter 5

# HPC Usability Research Team

## 5.1 Members

Hiroya Matsuba (Team Leader)

Motohiko Matsuda (Research Scientist)

Masatoshi Kawai (Postdosctoral Researcher)

## 5.2 Research Activities

The mission of the HPC Usability Team is to develop a software platform that contributes to increasing the number of uses of K computer. This team focuses on Cyber-physical systems as potential new use cases of supercomputers, expecting that simulation of infrastructure facilities will provide valuable information for its operation. For example, if a simulator of a cooling facility is available, the operator can search for an energy-efficient combination of parameters such as a temperature of cooling water, power of pumps, and positions of valves. Simulation is especially valuable because it enables operators to test parameters that are impossible to apply to the real facilities because of safety reasons.

The main obstacles in utilizing simulation technologies in cyber-physical systems are the difficulties in developing simulation programs. Although there are many simulator products, many of them focus on a specific technical aspects such as fluid dynamics or structural analysis, whereas simulation of industrial facilities requires a combination of various simulation techniques, such as a 1-D circuit, discrete event, or agent simulations as well as fluid or structural simulations. No one commercial software covers such a wide range of simulation techniques. The HPC Usability Research Team aims to develop a new programming framework that enables rapid development of simulation programs or programs that connect existing simulators, especially those for parallel computers. We assume engineers of companies who provide, for example, consulting service for factory management, use this programming framework.

We also aim to contribute to improving operational efficiencies of the K computer by actually adopting the techniques of the cyber-physical systems. By developing a simulator of the cooling and electric facilities of the supercomputer K or Fugaku, we expect we can improve operational efficiencies by, for example, reducing safety margins while ensuring the safety of the operation by using the simulator.

### 5.2.1 Development of the Pyne Parallel Programming Framework

We are developing a Pyne framework, which is a programming framework that enables ones to develop parallel programs as if they were sequential programs. The main objective of this framework is to improve the productivity of parallel programs, including simulation programs or programs for connecting existing simulators with another simulator or machine learning frameworks.

#### 5.2.1.1 Basic Concepts of Pyne

Pyne enables higher productivity of parallel programs by enabling programmers to write parallel programs as if they were sequential ones. Because parallelizing arbitrary sequential programs efficiently is almost impossible,

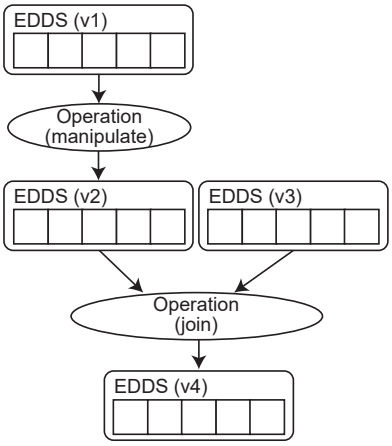


Figure 5.1: Example of Pyne's abstraction of a parallel program

```
 1  def main:
 2    v1 = edds.createVector(init=readStream)
 3    v2 = v1.manipulate(neighSize=1, func=avgFunc)
 4    v3 = edds.createVector(init=init3)
 5    v4 = edds.join(operands=(v2, v3),
 6                   neighSize=(0,0), func=addFunc,
 7                   dist=[1000])
 8    v5 = v4.out(func=writeStream)
 9    edds.final(v5)
10
11  def avgFunc(neigh):
12    return (neigh[0] + neigh[1] + neigh[2]) / 3.0
13
14  def addFunc(neigh):
15    return neigh[0][0] + neigh[1][0]
```

Figure 5.2: Example source code of a Pyne program

Pyne introduces an appropriate restriction on the type of supported application while maintaining as much flexibility as possible. To be specific, Pyne applications are required to be a series of operations on the data structures called the implementation independent data structure (IIDS). Figure 5.1 shows an example of the structure of a simple Pyne program. In this program, after the creation of an IIDS called v1, which is a vector (creation is also an operation but omitted from the figure), manipulate operation is applied to this IIDS. This operation generates the resulting IIDS v2. What is done in this manipulate operation is specified by the programmer as a sequential procedure. Then, after the creation of another IIDS called v3, this program performs the join operation. Finally, the resulting IIDS, which is v4, is generated by this operation.

Figure 5.2 shows the source code to express this program (unimportant details are omitted). The above-described operations are written at Lines 2 to 7, which is the main part of the program. The functions avgFunc and addFunc are to specify the concrete calculation that should be done during the manipulate and join operations, respectively. The important point is the fact that this program can be run with a huge size of vectors. In such a case, vectors are automatically distributed across multiple nodes. How to distribute the vector is determined by Pyne, not by programmers. Therefore, programmers do not have to care about the parallel implementation of the vector, which is why the data structure of Pyne is named implementation independent data structure.

The problem of Pyne is that it is practically valuable only when many kinds of IIDS operations are provided as a library. If this were possible, Pyne would have become an attractive parallel application development framework. However, the team has been terminated due to management problems, and has not developed an actual library. Nevertheless, we think it is worthwhile to have designed a novel interface for the simple development of parallel applications.

#### 5.2.1.2 Interface Improvement

This year, we tried to improve productivity of Pyne by allowing users to write pure Python programs. Last year, we developed Pyne using Cython, which is a technique to compile Python-like programs into native code. We used Cython, not pure Python, to obtain acceptable performance as an HPC infrastructure software. However, writing Cython programs requires knowledge of C language, which diminishes the productivity of Python.

The improved version of Pyne uses Numba, which is a Just-in-Time (JIT) compiler for Python. By specifying Python functions to be compiled at run-time, Numba generates native code using LLVM. The advantage of Numba over Cython is that it does not require uses to rewrite their programs. Although Numba only supports a subset of Python language, its functionality is generally enough to compile the user-defined part of IIDS operations.

Numba is usually used by adding `@jit` annotation to Python functions that should be compiled. However, using Numba with Pyne is not straightforward. Adding the annotation to the user-defined part of IIDS operations provides no performance improvement because the overhead to pass Python objects to native code is involved at the innermost loop. Python specific objects are necessary when Pyne runtime library calls the user-defined functions. Specifically, because the number of arguments and return values varies with the function, a generic logic to call user-defined functions will have to manipulate variable-length lists, which incurs handling of Python-specific objects.

To avoid using Python-specific objects when Pyne runtime calls user-defined functions, Pyne dynamically generates an interface function, which is specific to each user-define function. Because the interface function is generated for a particular user-defined function, the number of arguments and return values is also fixed, and no variable-length data structure is required. Such an interface function can be compiled by Numba, and fast execution is realized.

Figure 5.3 shows the example of IIDS usage and generated code. Pyne generates the function named `caller_calcTentVelocity_1013`. This generated function embeds the information of, for instance, how many operands are necessary for user-defined part, or how many values are returned from the function. The generated function and the user-defined function are optimized by Numba (JIT). Because these functions generally include only simple loops and numerical calculations, these functions are efficiently converted to native code at run-time.

#### 5.2.1.3 Performance Evaluation

We ran a fluid simulation application at two different sizes, $2700 \times 900$ and $900 \times 300$, for large and small configurations, respectively. There is an obstacle in the space. Fluid flows around this object. We ran this simulation program for 10,000 time-steps so that we can see meaningful simulation results, the appearance of Kármán's vortexes. We used a 16-node cluster. Each node had an 8-core Xeon Platinum 8280 processor, with 376 GB of memory, and an Intel Omini-Path (100Gbps) interconnect.

Figure 5.4 shows the strong scaling performance obtained by executing the Pyne and PETSc versions with the two problem sizes and with 3, 12, 48, 75, and 108 processes. As shown in the figure, the Pyne version performed as well as the PETSc version when the number of processes is small, but Pyne performs worth than the PETSc version when the number of processes is large. This performance difference comes from the overhead of code generation. Because Pyne generates Python code as described above, and Numba generates native code at run-time, the time for code generation is included in the benchmark results.

### 5.2.2 A Digital Twin for Operation Planning of the Cooling Facility

HPC Usability Research Team is collaborating with Operations and Computer Technologies Division on creating a real cyber-physical system. Our target is the cooling facility of Fugaku. By reproducing the behavior of the cooling facility of Fugaku with a simulator, we can do trial and error in searching for efficient operation parameters, such as temperature settings of cooling water. The cyber-physical system that virtually experiment potentially dangerous operations is often called a digital twin. This project is to utilize a digital twin for contribution to the efficient operation of supercomputers. We also collaborate with AIST (National Institute of Advanced Industrial Science and Technology), who operates ABCI, to utilize the experiences of both parties for the efficient operation of the supercompuers.

#### 5.2.2.1 Simulation Modeling of the Cooling Facility

The digital twin of the cooling facility will facilitate the mission of delivering a continuous operation of a supercomputer. It allows operation planning for efficiency optimizations and decision making at failure situations

```
# Global part
NX=900; NY=300
vT = gridInLoop.setNodeValue(
             func=calcTentVelocity,
             range=((1, 1), (NX-2,NY-2)),
             neigh=["v"], out=["vT"])

# Local part
def calcTentVelocity(cdn,myself,neigh):
  i, j = cdn
  uc, vc = neigh[0]
  ue, ve = neigh[1]
  .... (calculation) ...
  return (u, v)

# Generated code
def caller_calcTentVelocity_1013(
                 exeFunc,n_0,n_1,o_0,o_1):
  for j in range(1, 299):
    for i in range(1, 899):
      r=exeFunc((i+0,j+0),(),((
        (n_0[j,i],n_1[j,i]),
        (n_0[j,i+1],n_1[j,i+1]),
        (n_0[j+1,i],n_1[j+1,i]),
        (n_0[j,i-1],n_1[j,i-1]),
        (n_0[j-1,i],n_1[j-1,i]))))
      o_0[j,i]=r[0]
      o_1[j,i]=r[1]
```

Figure 5.3: IIDS Usage and Generated Code

which require a study on the behaviors of the cooling system. The key points of using the digital twin are the following.

- Cooling devices are slow and autonomous. That causes overshooting/undershooting of water temperatures. Minimizing deviations reduces energy consumption.
- Human operators need to prepare for faulty situations for prompt responses. Time bounds of tolerance are the keys to an operation protocol design.
- The facility designers, however, mainly concerns static, capacity-based limits. An analysis of dynamic behaviors is the responsibility of the operators.

We are working on simulation modeling with the focus on the dynamic behaviors, and performed some analysis on some artificial faulty situations. Figure 5.5 shows the simulation model of the cooling system modeled in Modelica. The model is still for K computer, currently. The enhancements of the cooling system from K to Fugaku are relatively minor, and the model will be updated for Fugaku after some actual operation data of Fugaku is collected.

#### 5.2.2.2 Simulation result validation in failure situations

Figures 5.6 show temperature variations of the cooling water at an event of turning off a power generator at a time around 2,000 seconds. In the event, the absorption chillers stopped working due to the lack of the steam from the power generator. During the event, a human operator promptly responded by starting additional chillers, and canceled the all submitted jobs as an emergency safety measure. The figures show good matches between the model simulation (solid lines) and the actual measurement (dotted lines).

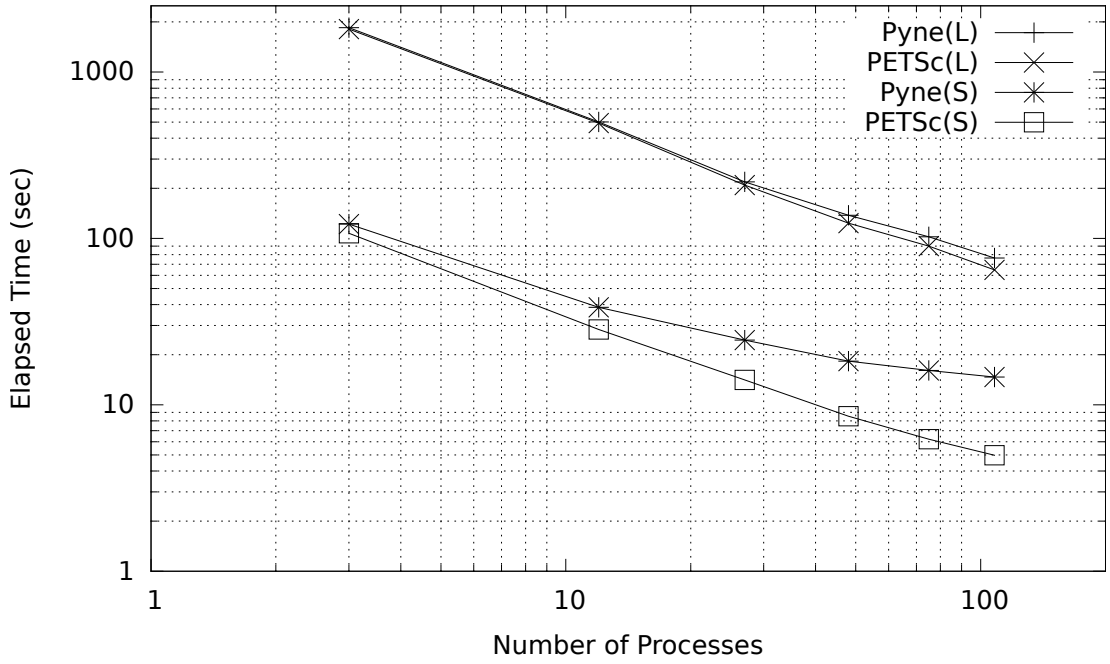


Figure 5.4: Performance (fluid simulation)

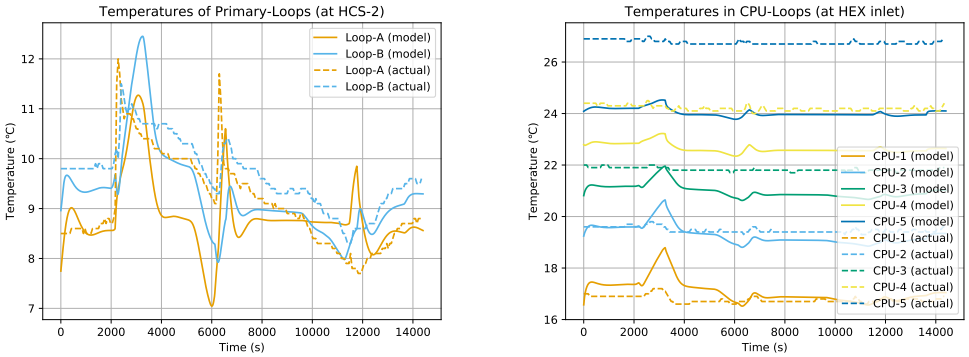


Figure 5.6: Water temperatures at a turning off event of a power generator

The model has well reproduced the behaviors in the situations of an event on a power generator as well as normal operations, where the temperature variations are unexpectedly complex. The next step is artificially sets up failure situations.

#### 5.2.2.3 Case: An operator would not cancel jobs

The first artificially set-up case is that the jobs are not canceled at turning off a power generator. In the validation case above, a human operator took a safety measure of canceling the all submitted jobs. But it was not necessary as a hindsight. The temperature variations are very similar without a cancellation. The behaviors of the water temperatures when the job cancellation is skipped are shown in Figures 5.7.

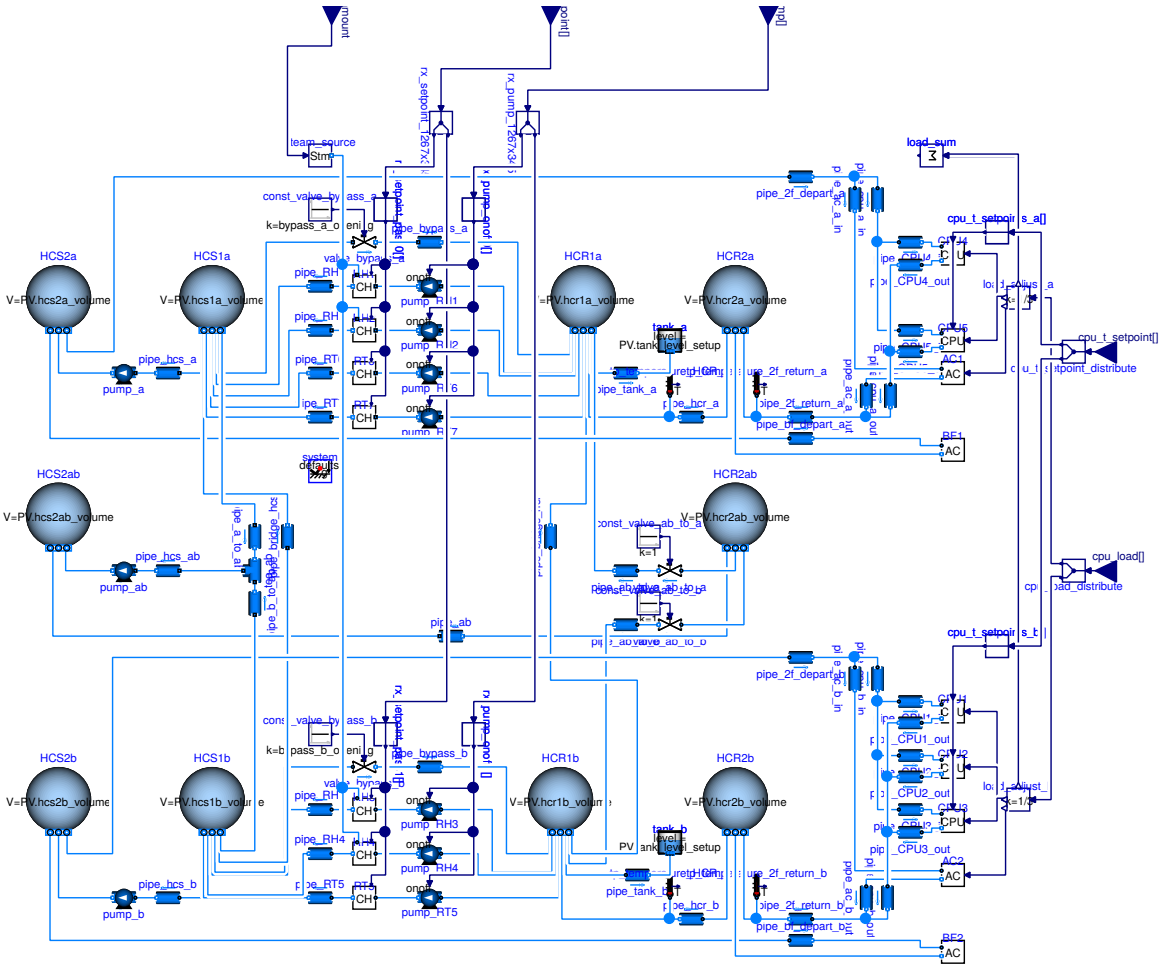


Figure 5.5: The model of the primary water loops in Modelica

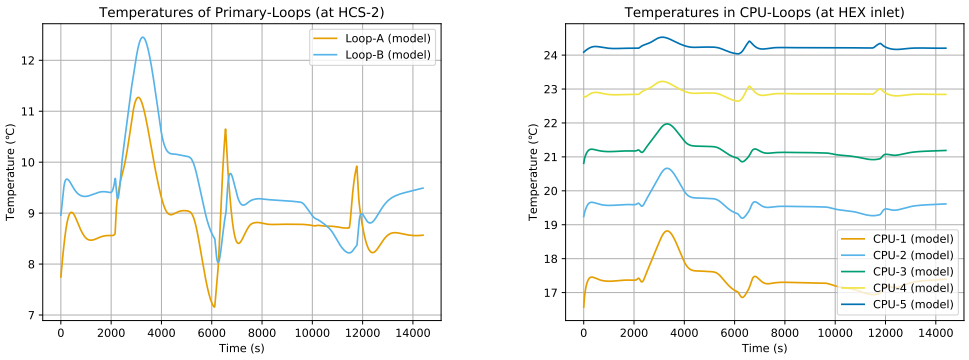


Figure 5.7: No cancellation of jobs at a turn off event

#### 5.2.2.4 Case: An operator would not react forever

The second artificially set-up case is that a human operator would not not react forever. In the validation case above, a human operator reacted very quickly in less than 10 minutes. The loss of a power generator is a fatal event for the cooling system, and the temperatures went higher quickly when an operator did not react at all. The simulation was stopped at a certain time before reaching the presupposed upper limit of the temperatures because that causes a simulation error (about 30 degrees Celsius). The behaviors of the water temperatures with no operator reactions are shown in Figures 5.8.

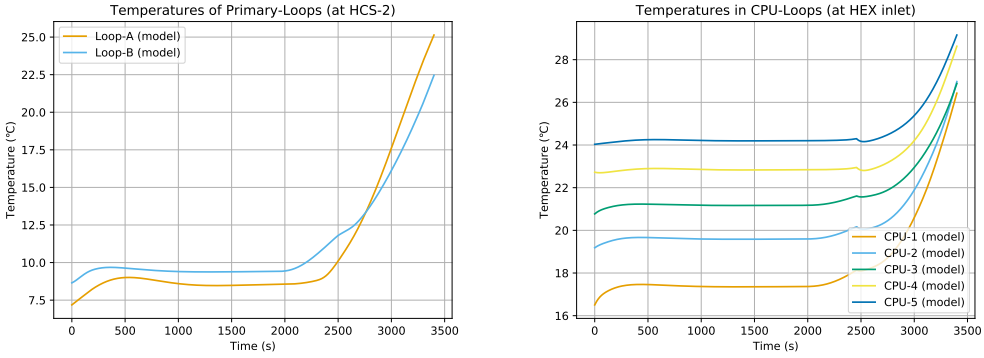


Figure 5.8: A human operator would not react

#### 5.2.2.5 Case: An operator would react 30 minutes late

The third artificially set-up case is that a human operator reacted very late. Actually, the system could tolerate the delay of 30 minutes, keeping the water temperatures of CPU cooling below 30 degrees Celsius. The behaviors of the water temperatures with late operator reactions are shown in Figures 5.9.

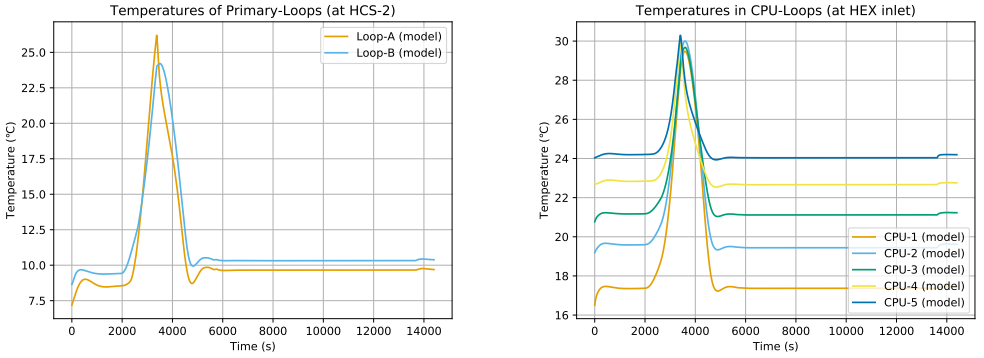


Figure 5.9: A human operator would react 30 minutes late

## 5.3 Schedule and Future Plan

This teams has been closed at the end of FY2019. There are no future plans.

## 5.4 Publications

### 5.4.1 Papers (refereed)

[1] M. Kawai, A. Ida, H. Matsuba, K. Nakajima, M. Bolten: "Multiplicative Schwartz-Type Block Multi-Color Gauss-Seidel Smoother for Algebraic Multigrid Methods". International Conference on High Performance Computing in Asia-Pacific Region 2020.

[2] H. Matsuba, M. Matsuda and M. Kawai: "Pyne: A programming framework for parallel simulation development". Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2) 2019.

[3] M. Matsuda, H. Matsuba, J. Nonaka, K. Yamamoto, H. Shibata and T. Tsukamoto: "Modeling the Existing Cooling System to Learn its Behavior for Post-K Supercomputer at RIKEN R-CCS". Energy Efficient HPC State of the Practice Workshop (EE HPC SOP) 2019.

[4] K. Hayashi, N. Sakamoto, J. Nonaka, M. Matsuda, and F. Shoji: "An In-Situ Visualization Approach for the K computer using Mesa 3D and KVS". WOIV 2018 (ISC Workshop on In Situ Visualization http://woiv.org/).

### 5.4.2 Posters (refereed)

[5] J. Nonaka, K. Ono, N. Sakamoto, K. Hayashi, M. Matsuda, F. Shoji, K. Oku, M. Fujita, K. Hatta: "A Large Data Visualization Framework for SPARC64 fx HPC Systems - Case Study: K Computer Operational Environment -", 2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV).
[6] M. Kawai, A. Ida and G. Wellein, "ppOpen-SOL: Robust ILU Preconditioner for Exascale", HPC in Asia, ISC2018, Germany, 2018
[7] M. Matsuda, J. Nonaka, K. Yamamoto H. Shibata, T. Tsukamoto, and H. Naemura, "R-CCS Facility Simulation Modeling for Assisting Operation Planning and Decision Making", The 2nd R-CCS International Symposium, 2020.

### 5.4.3 Presentations

[8] N. Nomura, K. Nakajima, M. Kawai, A. Fujii, "The Analysis of SA-AMG Method by Applying Hybrid MPI/OpenMP Parallelization on Cluster Supercomputer System", 15th Copper Mountain Conference On Iterative Methods, US, 2019
[9] N. Nomura, K. Nakajima, M. Kawai, A. Fujii, "The Evaluation of The SA-AMG Method By Applying Hybrid Parallelizaionon Cluster Computing System", ASE Seminar, Tokyo, 2019

### 5.4.4 Patents

[10] M. Mase, T. Sakurai, and H. Matsuba, "Simulation execution method and computer system", Joint application by Riken and Hitachi, Ltd., May, 2018